

IV B.Tech. I Semester
10BT71204: CRYPTOGRAPHY AND NETWORK SECURITY

UNIT-I: INTRODUCTION

Security Attacks - Interruption, Interception, Modification and Fabrication. Security Services Confidentiality, Authentication, Integrity, Non-repudiation, Access Control and Availability. Security Mechanisms. A model for Internet network security, Internet Standards and RFCs, Conventional Encryption Principles, Caesar Cipher, Hill cipher, Poly and Mono Alphabetic Cipher.

UNIT-II: ENCRYPTION PRINCIPLES

Conventional encryption algorithms: Feistel structure, DES algorithm, S-Boxes, Triple DES, Advanced Data Encryption Standard (AES), Cipher block modes of operation, location of encryption devices, key distribution Approaches of Message Authentication, Secure Hash Functions and HMAC.

UNIT-III: CRYPTOGRAPHY AND APPLICATIONS

Public key cryptography principles, public key cryptography algorithms, Digital signatures, RSA, Elliptic Algorithms, Digital Certificates, Certificate Authority and key management, Kerberos, X.509 Directory Authentication Service.

UNIT-IV: ELECTRONIC MAIL SECURITY

Email privacy: PGP operations, Radix-64 Conversion, Key Management for PGP, PGP Trust Model, Multipurpose Internet Mail Extension (MIME), Secure MIME (S-MIME).

UNIT-V: IP SECURITY ARCHITECTURE AND SERVICES

IP Security Overview, IP Security Architecture, Security Association, Authentication Header, Encapsulating Security Payload, Combining Security Associations and Key Management: OAKLEY key determination protocol, ISAKMP.

UNIT-VI: WEB SECURITY

Web Security Considerations, Secure Socket Layer (SSL) and Transport Layer Security (TLS), Secure Electronic Transaction (SET).

UNIT-VII: NETWORK MANAGEMENT SECURITY

Basic concepts of SNMP, SNMPv1 Community facility and SNMPv3. System Security: Intruders-Intrusion techniques, Intrusion Detection, Password Management, Bot nets. Malicious Software: Viruses and related threats, Virus Counter Measures, Distributed Denial of Service Attacks.

UNIT-VIII: FIREWALLS

Firewall Design principles, Trusted Systems, Common Criteria for Information Technology Security Evolution.

TEXT BOOKS:

1. William Stallings, Network Security Essentials Applications and Standards, 3rd edition, Pearson Education.
2. Stallings, Cryptography and network Security, 3rd edition, PHI/Pearson.

REFERENCE BOOKS:

- 0 Eric Maiwald, Fundamentals of Network Security , Dreamtech press, 2004.
- 1 Charlie Kaufman, Radia Perlman and Mike Speciner, Network Security - Private Communication in a Public World , 2nd edition, Pearson/PHI.
- 2 Robert Bragg, Mark Rhodes, Network Security: The complete reference , TMH, 2004.
- 3 Buchmann, Introduction to Cryptography , 2nd edition, Springer, 2004.

1.1 Security services:

Interception:

An **interception** means that some unauthorized party has gained access to an asset. The outside party can be a person, a program, or a computing system. Examples of this type of failure are illicit copying of program or data files, or wiretapping to obtain data in a network. Although a loss may be discovered fairly quickly, a silent interceptor may leave no traces by which the interception can be readily detected.

Attack: Interception



This is an attack on **Confidentiality**

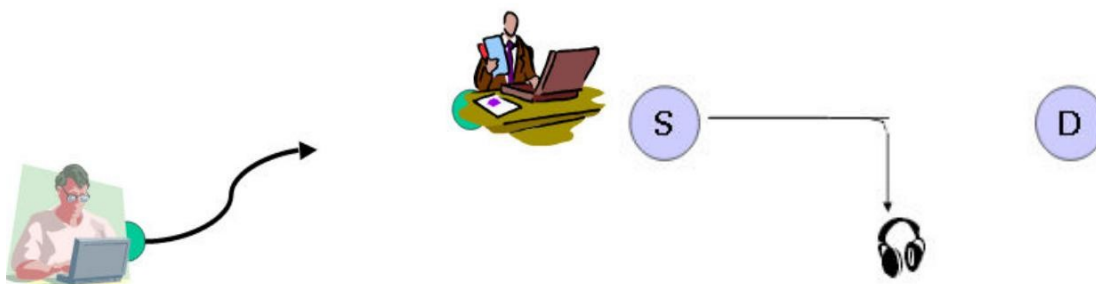
Approaches

Eavesdropping (listen), Link monitoring, Packet capturing, System compromise

Interruption:

In an **interruption**, an asset of the system becomes lost, unavailable, or unusable. An example is malicious destruction of a hardware device, erasure of a program or data file, or malfunction of an operating system file manager so that it cannot find a particular disk file.

Attack: Interruption



This is an attack on **Availability**

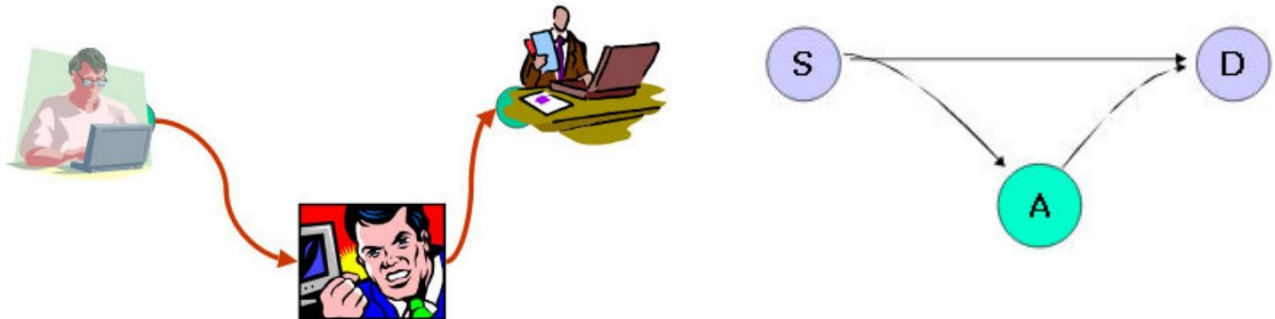
Approaches

Destruction of hardware, Physical damages to communication links, Introduction of noise, Removal of routing, Erase of a program or a file, DoS attacks

Modification:

If an unauthorized party not only accesses but tampers with an asset, the threat is a **modification**. For example, someone might change the values in a database, alter a program so that it performs an additional computation, or modify data being transmitted electronically. It is even possible to modify hardware. Some cases of modification can be detected with simple measures, but other, more subtle, changes may be almost impossible to detect.

Attack: Modification



This is an attack on **Integrity**

Approaches

Changing a record in a database, System compromise, making use of delays in communication, Modify hardware

Fabrication:

Finally, an unauthorized party might create a **fabrication** of counterfeit objects on a computing system. The intruder may insert spurious transactions to a network communication system or add records to an existing database. Sometimes these additions can be detected as forgeries, but if skillfully done, they are virtually indistinguishable from the real thing.

Attack: Fabrication



This is an attack on **Authenticity**

Approaches

Adding a new record to a database, Insertion of new network packet, Make use of IP spoofing.

1.2 Security Services:

Definition: 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.

Confidentiality: Confidentiality is the protection of transmitted data from passive attacks. The other aspect of confidentiality is the protection of traffic flow from analysis. 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. Different aspects of confidentiality:

5888 Connection Confidentiality

The protection of all user data on a connection.

5889 Connectionless Confidentiality

The protection of all user data in a single data block.

5890 Selective-Field Confidentiality

The confidentiality of selected fields within the user data on a connection or in a single data block.

5891 Traffic-Flow Confidentiality

The protection of the information that might be derived from observation of traffic flows.

Authentication: The authentication service is concerned with assuring that a communication is authentic.

Two specific authentication services are

Peer Entity Authentication

Used in association with a logical connection to provide confidence in the identity of the entities connected.

Data-Origin Authentication

In a connectionless transfer, provides assurance that the source of received data is as claimed.

Integrity: The assurance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).

Connection Integrity with Recovery

Provides for the integrity of all user data on a connection and detects any modification, insertion, deletion, or replay of any data within an entire data sequence, with recovery attempted.

Connection Integrity without Recovery

As above, but provides only detection without recovery.

Selective-Field Connection Integrity

Provides for the integrity of selected fields within the user data of a data block transferred over a connection and takes the form of determination of whether the selected fields have been modified, inserted, deleted, or replayed.

Connectionless Integrity

Provides for the integrity of a single connectionless data block and may take the form of detection of data modification. Additionally, a limited form of replay detection may be provided.

Selective-Field Connectionless Integrity

Provides for the integrity of selected fields within a single connectionless data block; takes the form of determination of whether the selected fields have been modified.

Non-repudiation: Provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication.

Nonrepudiation, Origin

Proof that the message was sent by the specified party.

Nonrepudiation, Destination

Proof that the message was received by the specified party.

Access Control:

The prevention of unauthorized use of a resource (i.e., this service controls who can have access to a resource, under what conditions access can occur, and what those accessing the resource are allowed to do).

Availability:

Availability to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity.

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.

1.3 Security Mechanisms.

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.

Encipherment

The use of 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.

Digital Signature

Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).

Access Control

A variety of mechanisms that enforce access rights to resources.

Data Integrity

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

Authentication Exchange

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

Traffic Padding

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

Routing Control

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

Notarization

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

PERVASIVE SECURITY MECHANISMS

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

Trusted Functionality

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

Security Label

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

Event Detection

Detection of security-relevant events.

Security Audit Trail

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

Security Recovery

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

1.4 A model for Internet network security

A model for Internet network security is shown in Figure 1.4. A message is to be transferred from one party to another across some sort of Internet service. The two parties, who are the **principals** in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the Internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

Security aspects come into play when it is necessary or desirable to protect the information transmission from an opponent who may present a threat to confidentiality, authenticity, and so on. All of the techniques for providing security have two components:

23 A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of a code based on the contents of the message, which can be used to verify the identity of the sender.

24 Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception.

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

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

5888 Design an algorithm for performing the security-related transformation. The algorithm should be such that an opponent cannot defeat its purpose.

5889 Generate the secret information to be used with the algorithm.

5890 Develop methods for the distribution and sharing of the secret information.

5891 Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

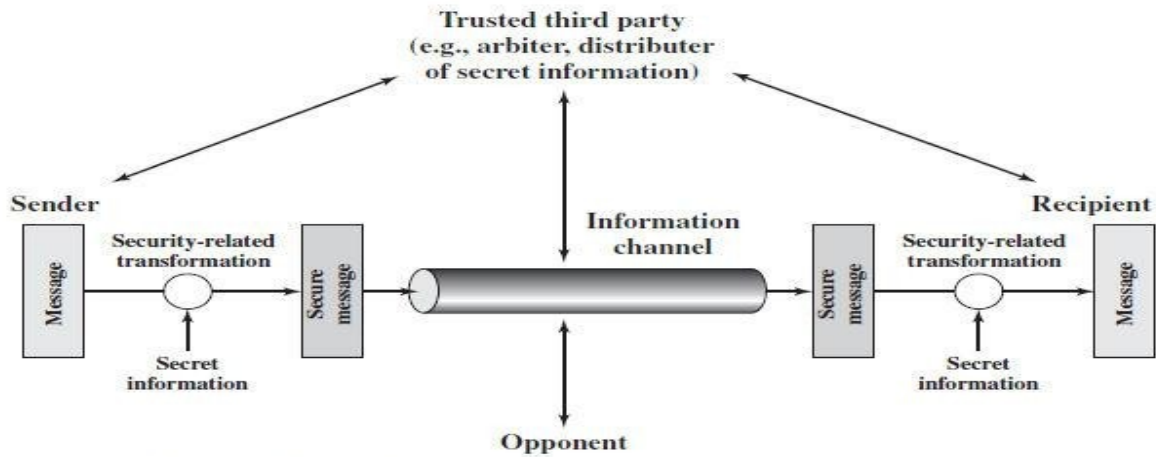


Figure 1.4 Model for Network Security

The **security mechanisms** needed to cope with unwanted access fall into two broad categories (see Figure 1.5). The first category might be termed a gatekeeper function. It includes password-based login procedures that are designed to deny access to all but authorized users and screening logic that is designed to detect and reject worms, viruses, and other similar attacks. Once either an unwanted user or unwanted software gains access, the second line of defense consists of a variety of internal controls that monitor activity and analyze stored information in an attempt to detect the presence of unwanted intruders.

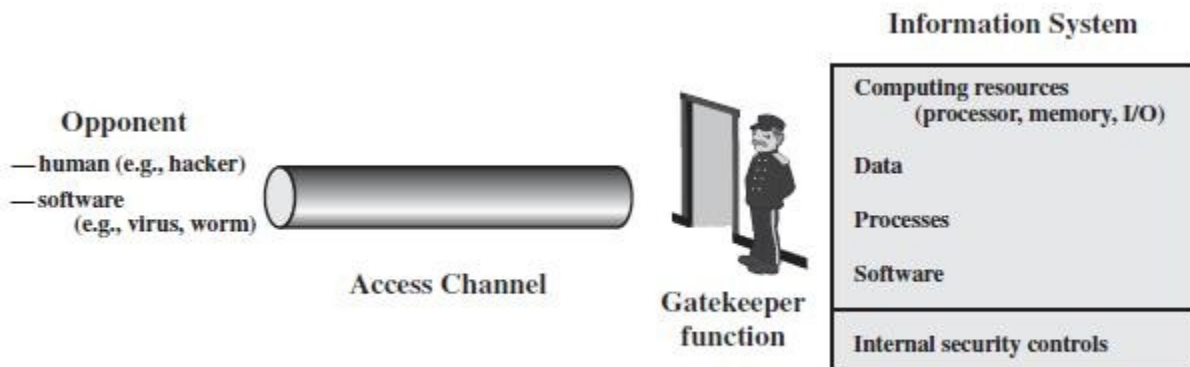


Figure 1.5 Network Access Security Model

1.5 INTERNET STANDARDS AND THE INTERNET SOCIETY

Many of the protocols that make up the TCP/IP protocol suite have been standardized or are in the process of standardization. By universal agreement, an organization known as the Internet Society is responsible for the development and publication of these standards. The internet society is a professional membership organization that oversees a number of boards and task forces involved in Internet development and standardization.

The Internet Organizations and RFCs Publication

The Internet Society is the coordinating committee for Internet design, engineering, and management. Areas covered include the operation of the internet itself and the standardization of protocols used by end systems on the Internet for interoperability. Three organizations under the Internet Society are responsible for the actual work of standards development and publication:

Internet Architecture Board (IAB): Responsible for defining the overall architecture of the Internet, providing guidance and board direction to the IETF.

Internet Engineering Task Force (IETF): the Protocol engineering and development arm of the Internet.

Internet Engineering task Force (IESG): responsible for technical management of IETF activities and the Internet standards process.

Working groups chartered by the IETF carry out the actual development of new standards and protocols for the Internet. During the development of a specification, a working group will make a draft version of the document available as an Internet Draft, which is placed in the IETF's "Internet Drafts" online directory. The document may remain as an Inter Draft for up to six months, and interested parties may review and comment on the draft. During that time, the IESG may approve publication of the draft as an RFC (Request for Comment). If the draft has not progressed to the status of an RFC during the Six-month period, it is withdrawn from the directory. The working group may subsequently publish a revised version of the draft.

The IETF is responsible for publishing the RFCs, with approval of the IESG. The RFCs are the working notes of the Internet research and development community.

The work of the IETF is divided into eight areas, each with an area director and each composed of numerous working groups. Table below shows the IETF areas and their focus.

Table IETF Areas

IETF Area	Theme	Example Working Groups
General	IETF processes and procedures	Policy Framework Process for Organization of Internet Standards
Applications	Internet applications	Web-related protocols (HTTP) EDI-Internet integration LDAP
Internet	Internet infrastructure	IPv6 PPP extensions
Operations and management	Standards and definitions for network operations	SNMPv3 Remote Network Monitoring
Routing	Protocols and management for routing information	multicast routing OSPF QoS routing
Security	Security protocols and technologies	Kerberos IPSec X.509 S/MIME TLS
Transport	Transport layer protocols	Differentiated services IP telephony NFS RSVP
User services	Methods to improve the quality of information available to users of the Internet	Responsible Use of the Internet User Services FYI documents

The Standardization Process

The decision of which TFCs become Internet standards is made by the IESG, on the recommendation of the IETF. To become a standard, a specification must meet the following criteria:

- 0** Be stable and well understood
- 1** Be technically competent
- 2** Have multiple, independent, and interoperable implementations with substantial operational experience
- 3** Enjoy significant public support
- 4** Be recognizably useful in some or all parts of the internet.

The figure below shows the series of steps, called the *standards track*, that a specification goes through to become a standard. The steps involve increasing amounts of scrutiny and testing. At each step, the IETF must make a recommendation for advancement of the protocol, and the IESG must ratify it. The process begins when the IESG approves the publication of an internet Draft document as an RFC with the status of proposed Standard.

The white boxes in fig represent temporary states, which should be occupied for the minimum practical time. A document must remain a proposed standard for at least six months and a Draft Standard for at least four months to allow time for review and comment. The gray boxes represent long term states that may be occupied for years.

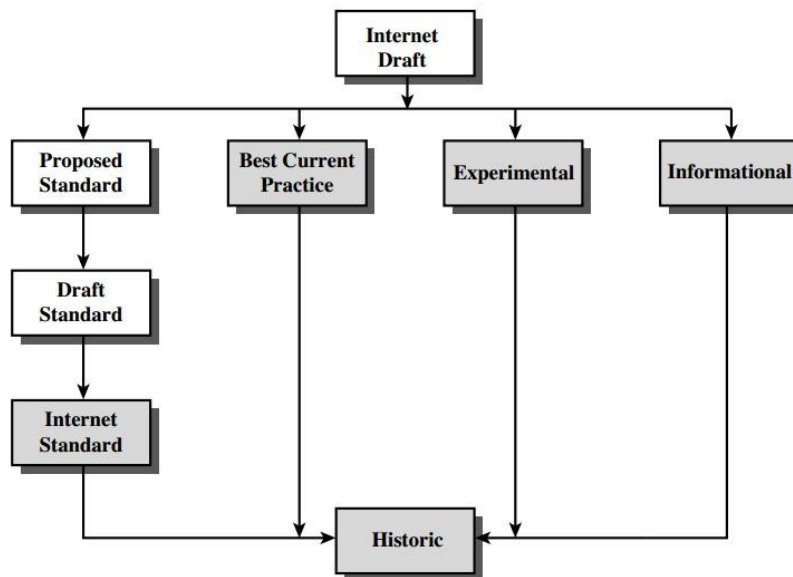


Figure 1.7 Internet RFC Publication Process

For a specification to be advanced to Draft Standard status there must be at least two independent and interoperable implementations from which adequate operational experience has been obtained.

After significant and operational experience has been obtained, a specification may be evolved to Internet Standard. At this point, the specification is assigned an STD number as well as an RFC number.

Finally, when a protocol becomes obsolete, it is assigned to the Historic state.

Internet Standards Categories

- 0 **Technical specification (TS):** a TS defines a protocol, service, procedure, convention, or format. Most Internet standards are TSs.
- 1 **Applicability statement (AS):** An AS specifies how, and under what circumstances, one or more TSs may be applied to support a particular Internet capability. An AS identifies one or more TS that are relevant to the capability, and may specify values or ranges for particular parameters associated with a TS or Functional subsets of a TS that are relevant for the capacity.

1.6 CONVENTIONAL ENCRYPTION PRINCIPLES OR SYMMETRIC ENCRYPTION PRINCIPLES

A symmetric encryption scheme has five ingredients (Figure below):

0 Plaintext: This is the original message or data that is fed into the algorithm as input.

1 Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

0 Secret key: The secret key is also input to the algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.

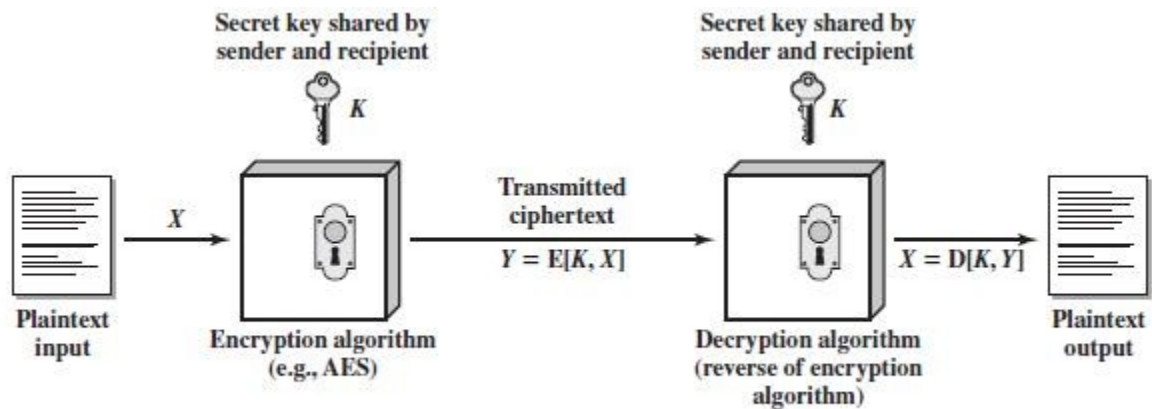


Figure Simplified Model of Symmetric Encryption

Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.

Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the cipher text and the same secret key and produces the original plaintext.

There are two requirements for secure use of symmetric encryption:

- 0 A strong encryption algorithm. At a minimum, the algorithm should be such that the opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
- 1 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.

Cryptography

Cryptographic systems are generically classified along three independent dimensions:

23 The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: **substitution**, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and **transposition**, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations be reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.

24 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 each use a different key, the system is referred to as asymmetric, two-key, or public-key encryption.

25 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

The process of attempting to discover the plaintext or key is known as cryptanalysis. The strategy used by the cryptanalyst depends on the nature of the encryption scheme and the information available to the cryptanalyst.

Table below summarizes the various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst.

Type of Attack	Known to Cryptanalyst
Ciphertext only	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded
Known plaintext	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • One or more plaintext–ciphertext pairs formed with the secret key
Chosen plaintext	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

The most difficult problem is presented when all that is available is the *ciphertext only*. 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.

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 of 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 **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. 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. 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 above lists two other types of attack: chosen ciphertext and chosen text.

Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

An encryption scheme is **computationally secure** if the ciphertext generated by the scheme meets one or both of the following criteria:

23 The cost of breaking the cipher exceeds the value of the encrypted information.

24 The time required to break the cipher exceeds the useful lifetime of the information.

A brute-force approach involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table below shows how much time is involved for various key sizes.

Table Average Time Required for Exhaustive Key Search

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

The 56-bit key size is used with the DES (Data Encryption Standard) algorithm. For each key size, the results are shown assuming that it takes 1 μ s to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater. The final column of Table considers the results for a system that can process 1 million keys per microsecond.

Caesar Cipher

This is the earliest and the simplest substitution cipher designed by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

For example,

plain: meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB

Note that the alphabet is wrapped around, so that the letter following Z is A.

Then the algorithm can be expressed as follows. For each plaintext letter p , substitute the ciphertext letter C :

$$C = E(3, p) = (p + 3) \bmod 26$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = E(k, p) = (p + k) \bmod 26$$

where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$$p = D(k, C) = (C - k) \bmod 26$$

If it is known that a given cipher text is a Caesar cipher, then a brute-force cryptanalysis is easily performed: simply try all the 25 possible keys. Three important characteristics of this problem enabled us to use a brute-force cryptanalysis:

5888 The encryption and decryption algorithms are known.

2. There are only 25 keys to try.

3. The language of the plaintext is known and easily recognizable.

Hill Cipher

Another interesting multiletter cipher is the Hill cipher, developed by the mathematician Lester Hill in 1929.

This encryption algorithm takes successive plaintext letters and substitutes for them ciphertext letters. The substitution is determined by linear equations in which each character is assigned a numerical value

($a = 0, b = 1, \dots, z = 25$). For $m = 3$, the system can be described as

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$

$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \bmod 26$$

$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \bmod 26$$

This can be expressed in terms of row vectors and matrices:⁷

$$(c_1 \ c_2 \ c_3) = (p_1 \ p_2 \ p_3) \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \bmod 26$$

or

$$\mathbf{C} = \mathbf{PK} \bmod 26$$

where \mathbf{C} and \mathbf{P} are row vectors of length 3 representing the plaintext and ciphertext, and \mathbf{K} is a 3×3 matrix representing the encryption key. Operations are performed mod 26.

For example, consider the plaintext “paymoremoney” and use the encryption key

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

Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack.

The first three letters of the plaintext are represented by the vector (15 0 24). Then $(15\ 0\ 24)\mathbf{K} = (303\ 303\ 531) \bmod 26 = (17\ 17\ 11) = \text{RRL}$. Continuing in this fashion, the ciphertext for the entire plaintext is RRLMWBKASPDH.

Decryption requires using the inverse of the matrix \mathbf{K} . We can compute $\det \mathbf{K} = 23$, and therefore, $(\det \mathbf{K})^{-1} \bmod 26 = 17$. We can then compute the inverse as

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix \mathbf{K}^{-1} is applied to the ciphertext, then the plaintext is recovered.

In general terms, the Hill system can be expressed as

$$\begin{aligned} \mathbf{C} &= \mathbf{E}(\mathbf{K}, \mathbf{P}) = \mathbf{PK} \bmod 26 \\ \mathbf{P} &= \mathbf{D}(\mathbf{K}, \mathbf{C}) = \mathbf{CK}^{-1} \bmod 26 = \mathbf{PKK}^{-1} = \mathbf{P} \end{aligned}$$

Consider this example. Suppose that the plaintext “hillcipher” is encrypted using a 2×2 Hill cipher to yield the ciphertext HCRZSSXNSP. Thus, we know that $(7\ 8)\mathbf{K} \bmod 26 = (7\ 2)$; $(11\ 11)\mathbf{K} \bmod 26 = (17\ 25)$; and so on. Using the first two plaintext–ciphertext pairs, we have

$$\begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix} \mathbf{K} \bmod 26$$

The inverse of \mathbf{X} can be computed:

$$\begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix}^{-1} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix}$$

so

$$\mathbf{K} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix} \begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 549 & 600 \\ 398 & 577 \end{pmatrix} \bmod 26 = \begin{pmatrix} 3 & 2 \\ 8 & 5 \end{pmatrix}$$

This result is verified by testing the remaining plaintext–ciphertext pairs.

Mono and Poly Alphabetic Cipher.

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. .

If, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4×10^{26} possible keys. it eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher**.

There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., non compressed English text), then the analyst can exploit the regularities of the language. As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 2.5. If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match.

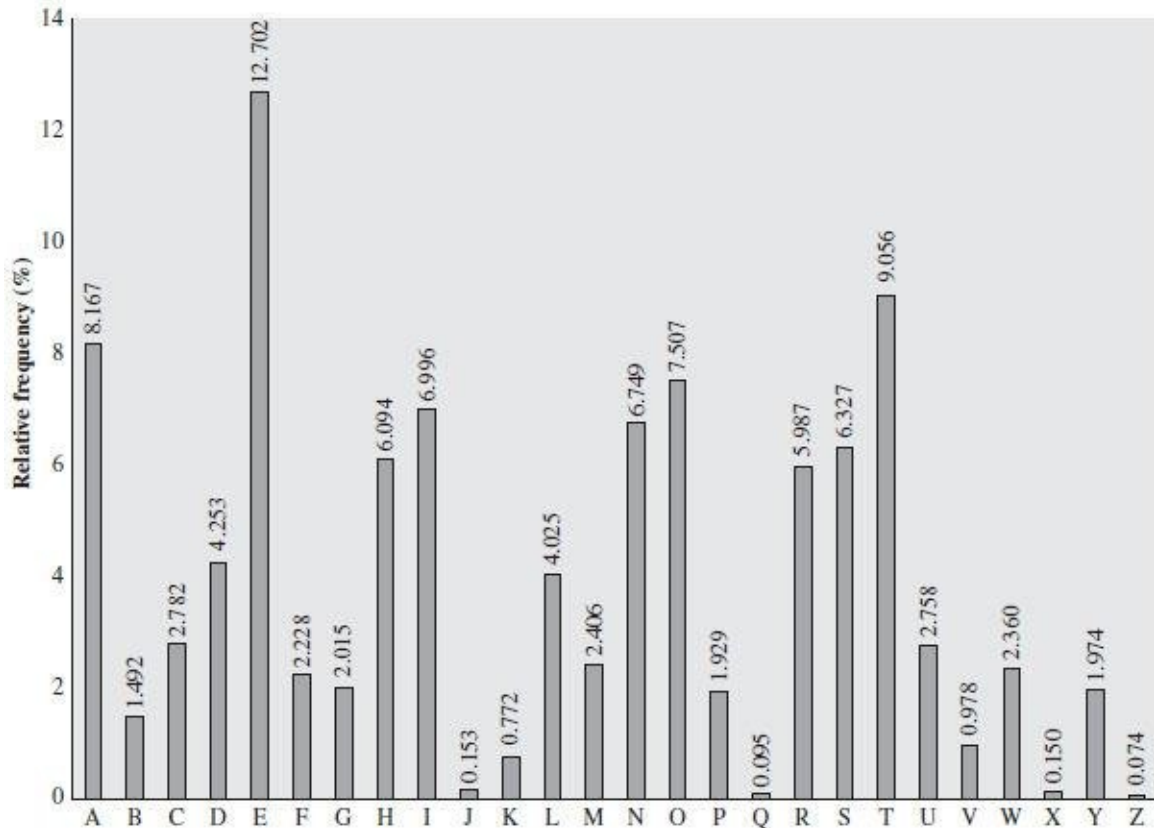


Figure 2.5 Relative Frequency of Letters in English Text

Exapmle:

Cipher text:

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDSHZOWSFPAPPDTSVPQZWMXUZHUSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

Relative frequency of characters in cipher text:

P 13.33	H 5.83	F 3.33	B 1.67	C 0.00
Z 11.67	D 5.00	W 3.33	G 1.67	K 0.00
S 8.33	E 5.00	Q 2.50	Y 1.67	L 0.00
U 8.33	V 4.17	T 2.50	I 0.83	N 0.00
O 7.50	X 4.17	A 1.67	J 0.83	R 0.00
M 6.67				

Relative Frequency of Letters in English Text:

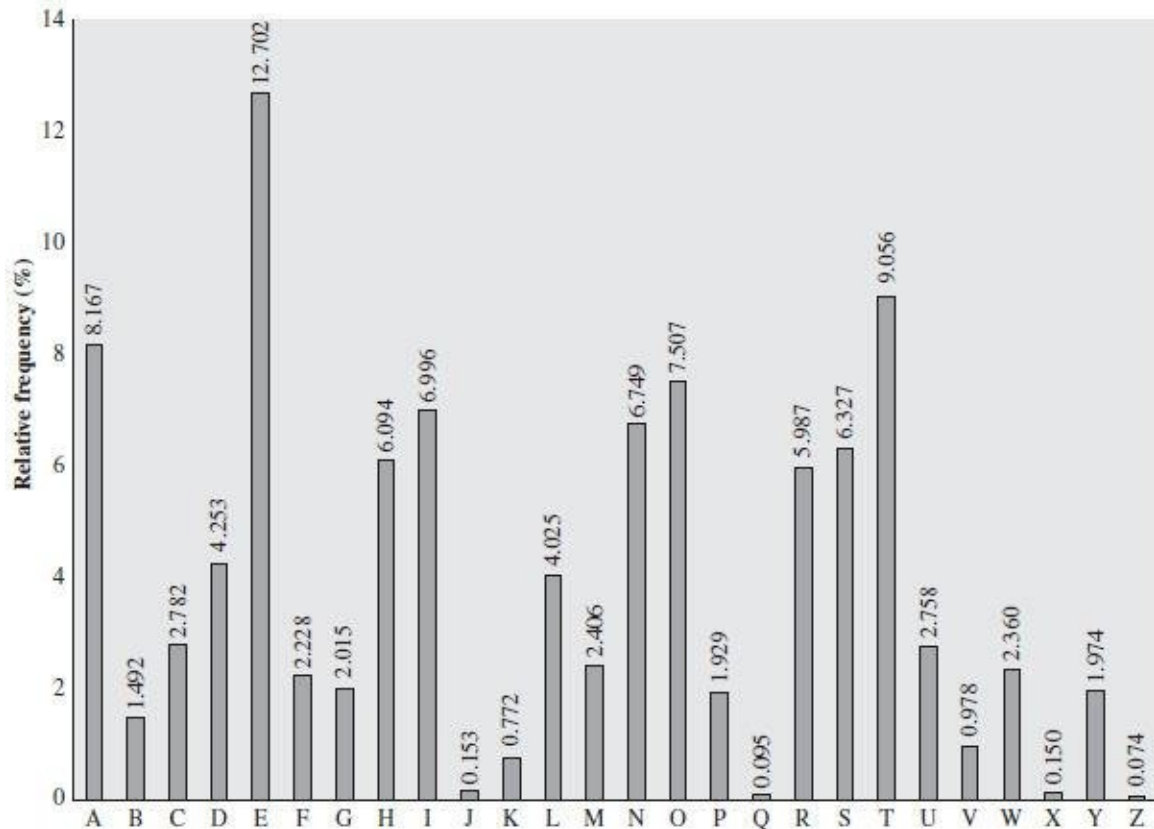


Figure 2.5 Relative Frequency of Letters in English Text

Crypt analysis:

```

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
t a e e t e a t h a t e e a a
VUEPHZHMDZSHZOWSFPAPDTSVPQUZWMYXUZHXS
e t t a t h a e e e a e t h t a
EPYEPDPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
e e e t a t e t h e t

```

A powerful tool is to look at the frequency of two-letter combinations, known as **digrams**. A table similar to Figure 2.5 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our ciphertext, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with

h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the ciphertext, and we can translate that sequence as "the." This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track. Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th_t. If so, S equates with a.

Only four letters have been identified, but already we have quite a bit of the message. Continued analysis of frequencies plus trial and error should easily yield a solution from this point. The complete plaintext, with spaces added between words, follows:

```
it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow
```

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

Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is **polyalphabetic substitution cipher**. All these techniques have the following features in common:

5888 A set of related monoalphabetic substitution rules is used.

5889 A key determines which particular rule is chosen for a given transformation.

VIGENÈRE CIPHER The best known, and one of the simplest, polyalphabetic ciphers is the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter a. Thus, a Caesar cipher with a shift of 3 is denoted by the key value.

The process of encryption is simple: Given a key letter x and a plaintext letter y, the ciphertext letter is at the intersection of the row labeled x and the column labeled y; in this case the ciphertext is V. To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

key:	deceptivedeceptivedeceptive
plaintext:	wearediscoveredsaveyourself
ciphertext:	ZICVTWQNGRZGVTVAVZHCQYGLMGJ

Decryption is equally simple. The key letter again identifies the row. The position of the ciphertext letter in that row determines the column, and the plaintext letter is at the top of that column.

The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost.

Table 2.4 The Modern Vigenère Tableau

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

UNIT-II: ENCRYPTION PRINCIPLES

Conventional encryption algorithms: Feistel structure, DES algorithm, S-Boxes, Triple DES, Advanced Data Encryption Standard (AES), Cipher block modes of operation, location of encryption devices, key distribution, Approaches of Message Authentication, Secure Hash Functions and HMAC.

XXXX-----

2.1 Feistel Cipher Structure

Many symmetric block encryption algorithms, including DES, have a structure first described by Horst Feistel of IBM in 1973 and shown in Figure below.

The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K . The plaintext block is divided into two halves, $LE0$ and $RE0$. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs LE_{i-1} and RE_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K . In general, the subkeys K_i are different from K and from each other and are generated from the key by a subkey generation algorithm.

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 (XOR) 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

5888 Block size: Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed. A block size of 128 bits is a reasonable tradeoff and is nearly universal among recent block cipher designs.

5889 Key size: Larger key size means greater security but may decrease encryption/decryption speed. The most common key length in modern algorithms is 128 bits.

5890 Number of rounds: The essence of a symmetric block cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

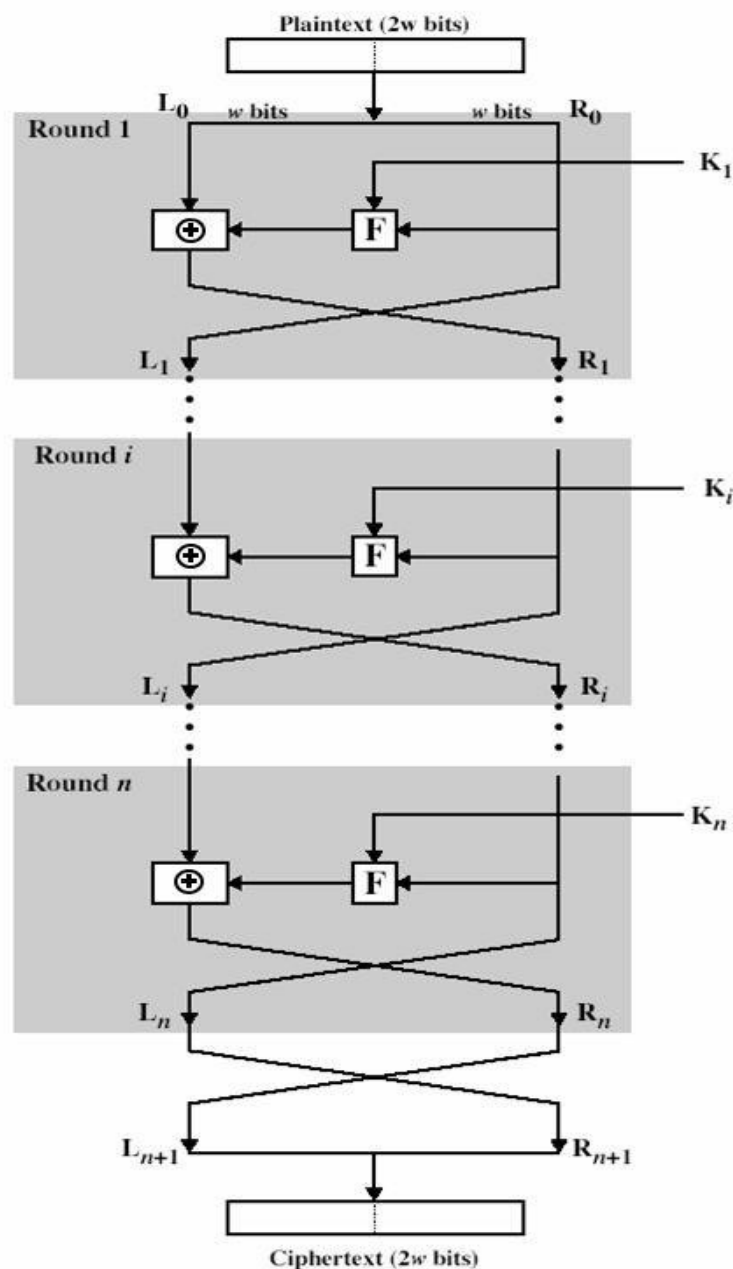
5891 Subkey generation algorithm: Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

5892 Round function: Again, greater complexity generally means greater resistance to cryptanalysis. There are two other considerations in the design of a symmetric block cipher:

5889 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.

5890 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 with a symmetric block cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys 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.



2.2 DES algorithm:

- 5888 The most widely used encryption scheme
- 5889 The algorithm is referred to the Data Encryption Algorithm (DEA)
- 5890 DES is a block cipher
- 5891 The plaintext is processed in 64-bit blocks
- 5892 The key is 56-bits in length

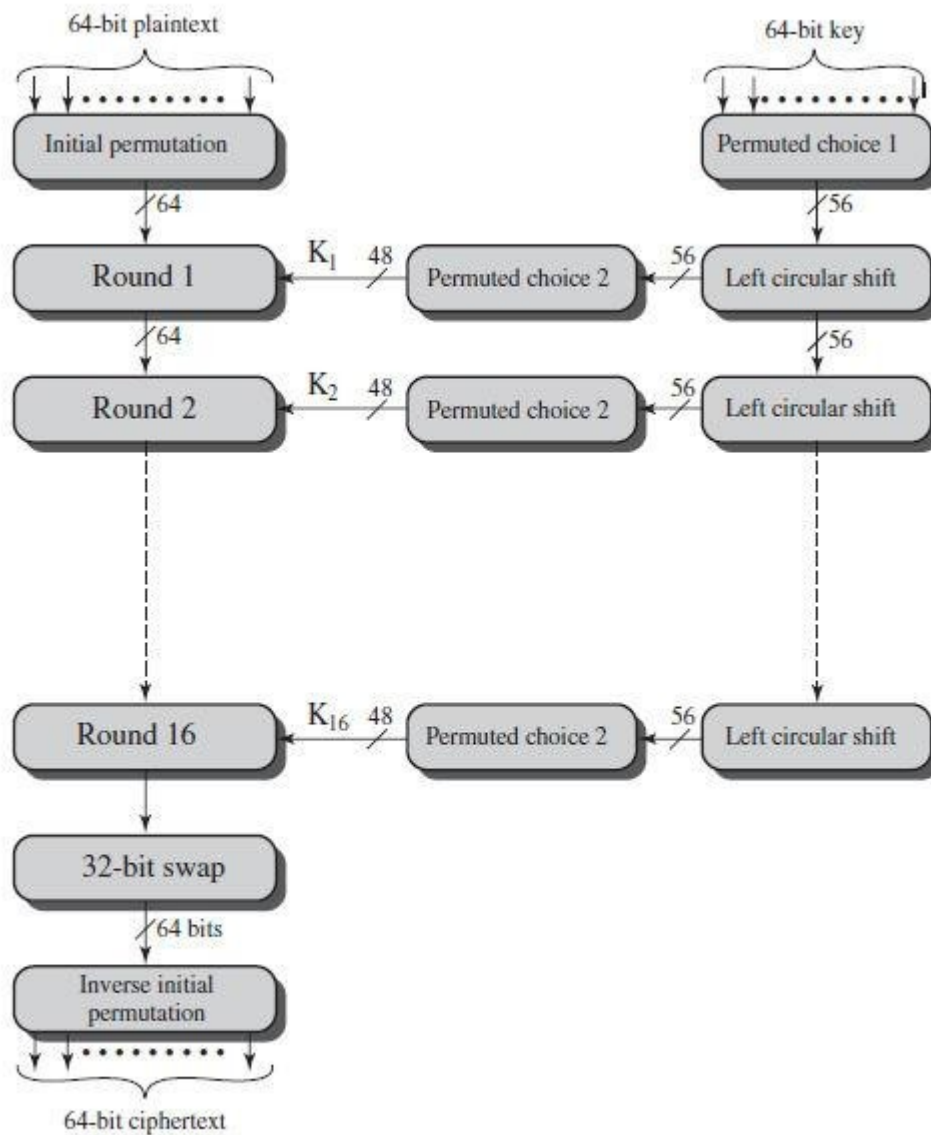


Figure 3.5 General Depiction of DES Encryption Algorithm

Initial Permutation (IP):

The plaintext block undergoes an initial permutation. 64 bits of the block are permuted.

A Complex Transformation:

64 bit permuted block undergoes 16 rounds of complex transformation. (Using subkeys).

32-bit swap:

32 bit left and right halves of the output of the 16th round are swapped.

Inverse Initial Permutation (IP⁻¹):

The 64 bit output undergoes a permutation that is inverse of the initial permutation. The 64 bit output is the ciphertext.

Table 3.2 Permutation Tables for DES**(a) Initial Permutation (IP)**

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

(b) Inverse Initial Permutation (IP⁻¹)

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

(c) Expansion Permutation (E)

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

(d) Permutation Function (P)

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

Single round DES Algorithm

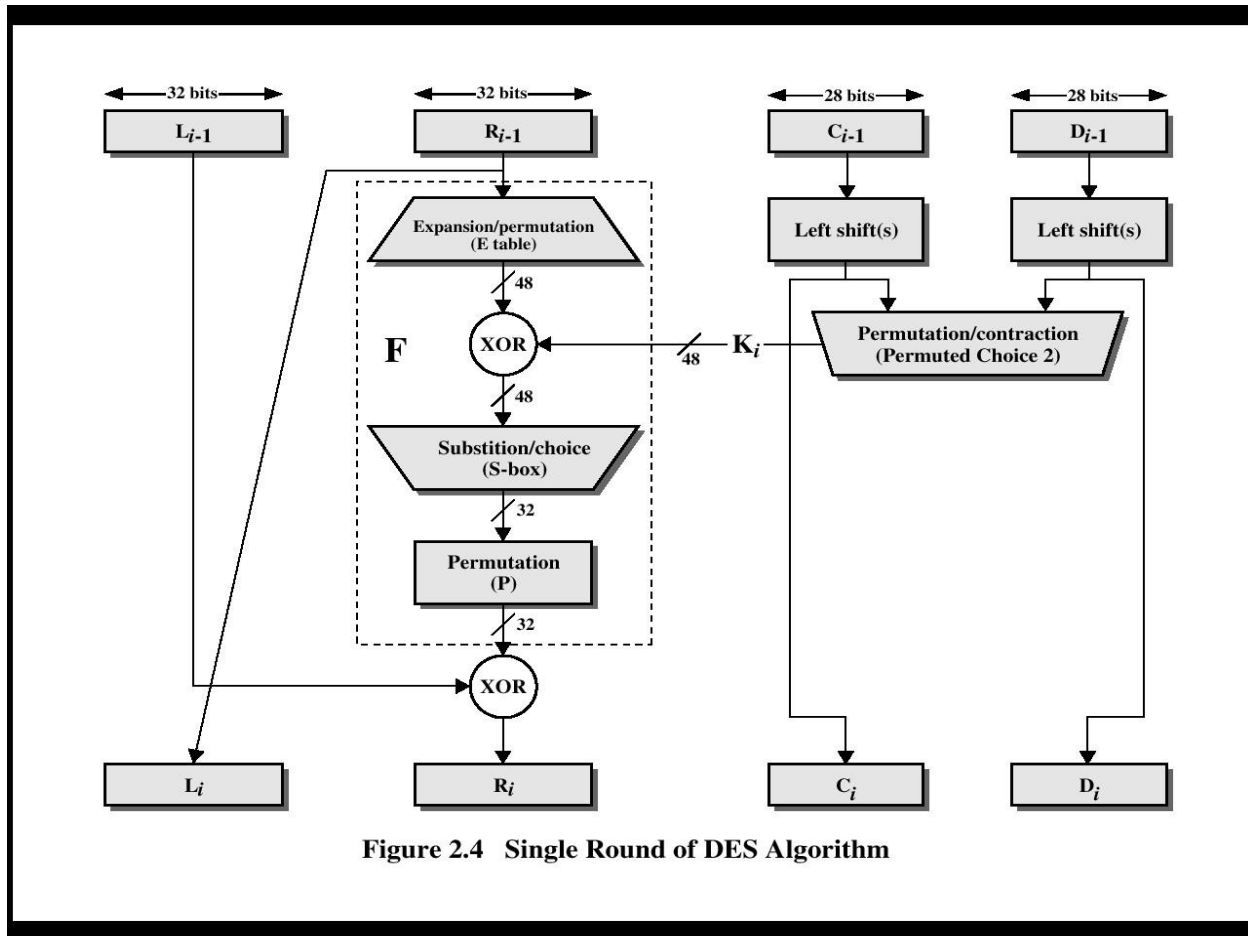


Figure 2.4 Single Round of DES Algorithm

The complex processing at each iteration/round:

$L_i = R_{i-1}$

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

Details of function F:

It takes 32 bits input and produces a 32 bit output.

Details of function F:

>32 bit input is expanded into 48 bits.

-This is done by permuting and duplicating some bits of 32 bits.

>Exclusive OR operation is performed between these 48 bits and 48 bit subkey.

Details of function F:...

23 48 bit output of the Exclusive OR operation is grouped into 8 groups of 6 bits each.

24 Each 6 bit group is fed into a 6-to-4 substitution box that transforms 6 bits to 4 bits.

Details of function F:...

25 32 bit output of 8 substitution boxes is fed into a permutation box.

26 The 32 bit output of the permutation box is $F(R_{i-1}, K_i)$.

Concerns about:

The key length (56-bits)

5888 56 bit key was adequate in 70s.

5889 With faster processors, this encryption method is no longer safe.

Calculation of $F(R, K)$

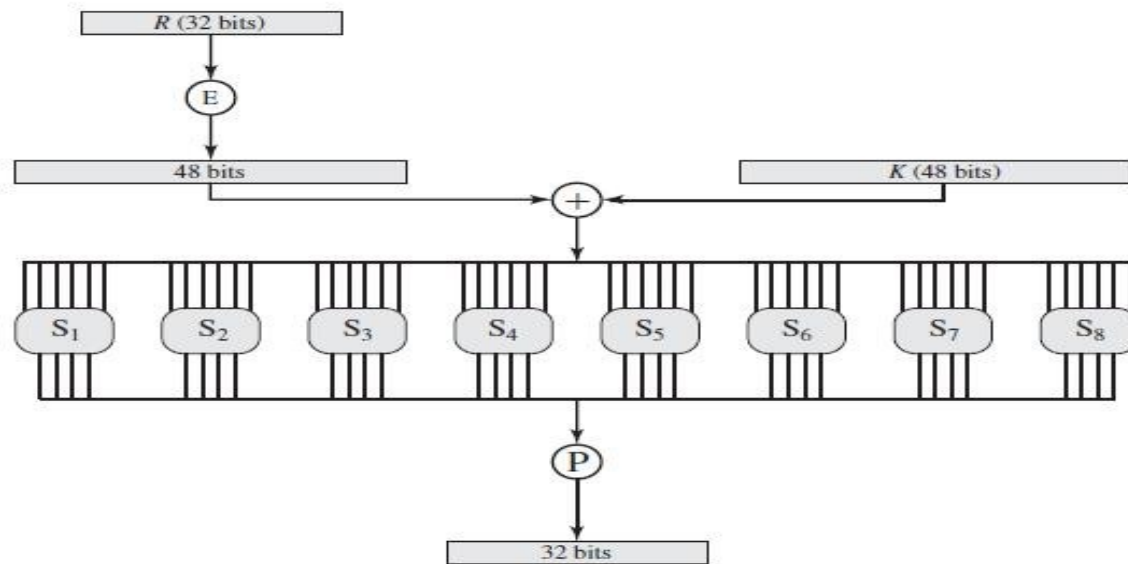


Figure 3.7 Calculation of $F(R, K)$

Properties:

Two desired properties of a block cipher are the **avalanche effect** and the **completeness**. To check the avalanche effect in DES, let us encrypt two plaintext blocks (with the same key) that differ only in one bit and observe the differences in the number of bits in each round.

Plaintext: 0000000000000000	Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1	
Plaintext: 0000000000000000 1	Key: 22234512987ABB23
Ciphertext: 0A4ED5C15A63FEA3	

Although the two plaintext blocks differ only in the rightmost bit, the ciphertext blocks differ in 29 bits. This means that changing approximately 1.5 percent of the plaintext creates a change of approximately 45 percent in the ciphertext.

Completeness effect

Completeness effect means that each bit of the ciphertext needs to depend on many bits on the plaintext.

The strength (security issues) of des

These concerns, fall into two areas: key size and the nature of the algorithm.

- The Use of 56-Bit Keys

- The Nature of the DES Algorithm

- Timing Attacks

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16} keys. Thus, on the face of it, a brute-force attack appears impractical. But it is broken using a special-purpose “DES

cracker” machine by Electronic Frontier Foundation (EFF) in July 1998. Fortunately, there are a number of alternatives to DES, the most important of which are AES and triple DES.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, 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 supposed 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 a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. This is a long way from knowing the actual key, but it is an intriguing first step.

2.3 S-box design criteria

- P1. No S-box is a linear or affine function of the input.
- P2. Changing 1 input bit to an S-box results in changing at least 2 output bits.
- P3. $S(x)$ and $S(x+001100)$ must differ in at least 2 bits.

The following were labelled by the NSA as “caused by design criteria”:

- P4. $S(x) \neq S(x+11ab00)$ for any choice of a and b .
 - P5. The S-boxes were chosen to minimize the difference between the number of 1's and 0's in any S-box output when any single input bit is held constant.
 - P6. The S-boxes chosen required significantly more logical minterms to implement than a random choice would require. A minterm is a logical AND (boolean product) of input bits (and their negations), which form a necessary (but not sufficient) condition for a particular output bit to be asserted. An output bit may have its value completely described by the logical OR (boolean sum) of all its minterms. The number of minterms in such an expression (after simplification) is a measure of its complexity. In the worst case, for n input bits, 2^n minterms may be needed to describe each output bit.
-

After the invention of differential cryptanalysis by Biham and Shamir [2], Don Coppersmith [9, 10] revealed the criteria¹ used in the S-box design two decades earlier:

1. Each S-box should have six bits of input and four bits of output. (In 1974 this was the largest size S-box that could be accommodated if DES were to fit on a single chip.)
2. No output bit of an S-box should be too close to a linear function of the input bits. (The S-boxes are the only nonlinear part of DES. Their nonlinearity is the algorithm's strength.)
3. Each "row" of an S-box should contain all possible outputs. (This randomizes the output.)
4. If two inputs to an S-box differ in exactly one bit, their outputs should differ in at least two bits.
5. If two inputs to an S-box differ exactly in the middle two bits, their outputs must differ by at least two bits. (Criteria (4) and (5) provide some diffusion.)
6. If two inputs to an S-box differ in their first two bits and agree on their last two, the two outputs must differ.
7. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.

2.4 Triple DES

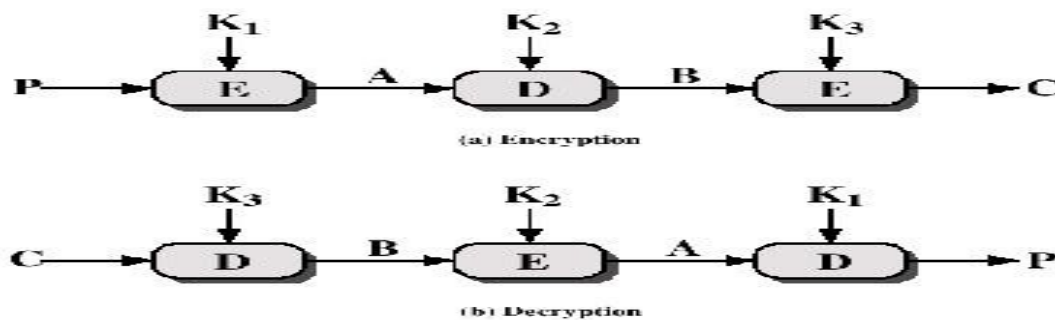


Figure 2.6 Triple DEA

3DES uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure 2.4a):

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

where

C = ciphertext

P = plaintext

$E[K, X]$ = encryption of X using key K

$D[K, Y]$ = decryption of Y using key K

Decryption is simply the same operation with the keys reversed (Figure 2.4b):

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

2.5 Advanced Data Encryption Standard (AES)

- 23 The principal drawback of 3DES is that the algorithm is relatively slow in software.
- 24 The original DEA was designed for mid-1970s hardware implementation and does not produce efficient software code.
- 25 3DES, which has three times as many rounds as DEA, is correspondingly slower.
- 26 A secondary drawback is that both DEA and 3DES use a 64-bit block size.
- 27 For reasons of both efficiency and security, a larger block size is desirable.
- 28 As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have a security strength equal to or better than 3DES and significantly improved efficiency.
- 29 In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.
- 30 Evaluation criteria included security, computational efficiency, memory requirements, hardware and software suitability, and flexibility.
- 31 The two researchers who developed AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.

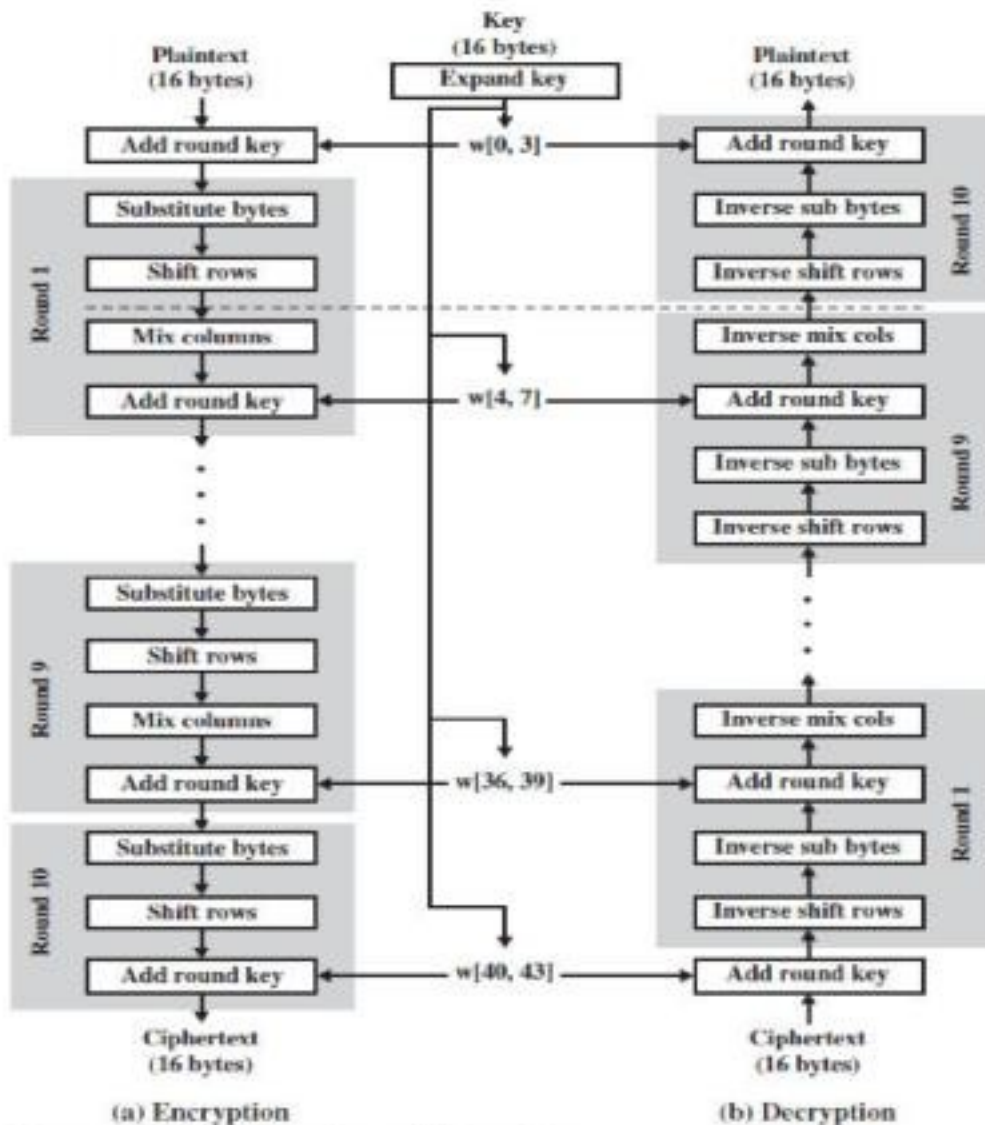


Figure 2.5 AES Encryption and Decryption

5888 The following comments give some insight into AES.

5888 One noteworthy feature of this structure is that it is not a Feistel structure, but processes the entire data block in parallel during each round using substitutions and permutation.

5889 The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$. Four distinct words (128 bits) serve as a round key for each round.

5889 Four different stages are used, one of permutation and three of substitution:

5888 **Substitute bytes:** Uses a table, referred to as an S-box, to perform a byte-by-byte substitution of the block.

5889 **Shift rows:** A simple permutation that is performed row by row.

5890 **Mix columns:** A substitution that alters each byte in a column as a function of all of the bytes in the column.

23 Add round key: A simple bitwise XOR of the current block with a portion of the expanded key.

The structure is quite simple. For both encryption and decryption, the cipher begins with an Add Round Key stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.

Only the **Add Round Key** stage makes use of the key. For this reason, the cipher begins and ends with an Add Round Key stage.

The Add Round Key stage by itself would not be tough. The other three stages together scramble the bits, but by themselves, they would provide no security because they do not use the key.

Each stage is easily reversible. For the Substitute Byte, Shift Row, and Mix Columns stages, an inverse function is used in the decryption algorithm.

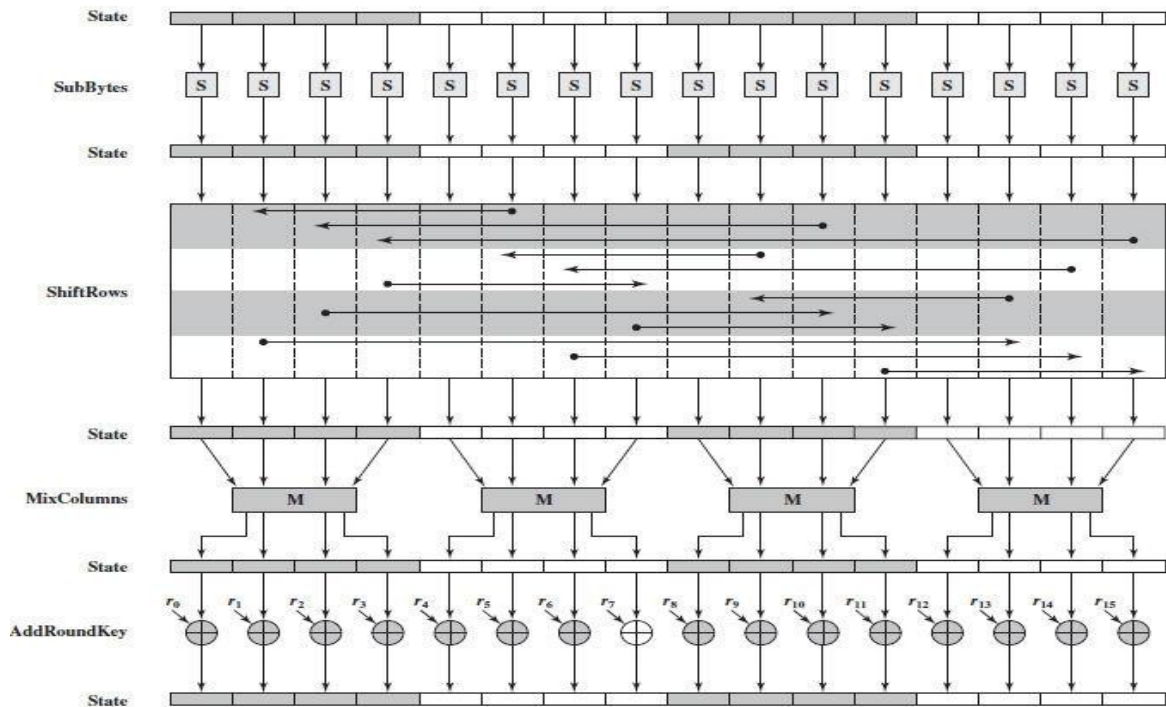


Figure 2.6 AES Encryption Round

Other Symmetric Block Ciphers:

23 International Data Encryption Algorithm (IDEA)

23 128-bit key

24 Used in PGP

24 Blowfish

23 Easy to implement

24 High execution speed

25 Run in less than 5K of memory

25 RC5

- 5888** Suitable for hardware and software
- 5889** Fast, simple
- 5890** Adaptable to processors of different word lengths
- 5891** Variable number of rounds
- 5892** Variable-length key
- 5893** Low memory requirement
- 5894** High security
- 5895** Data-dependent rotations
- 5889 Cast-128
- 5888** Key size from 40 to 128 bits
- 5889** The round function differs from round to round

2.6 Cipher block modes of operation:

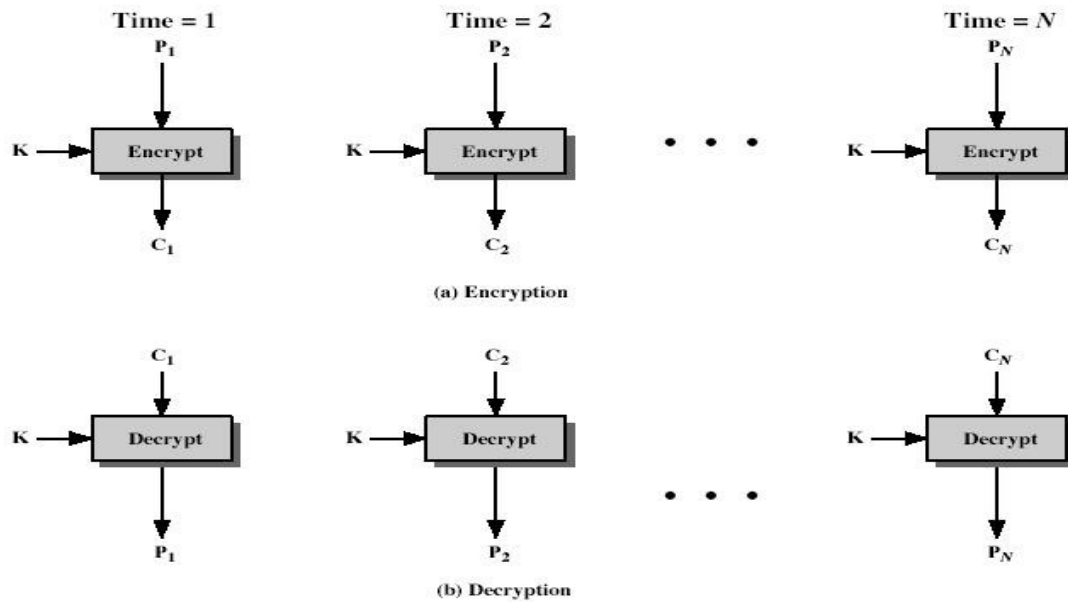
- 23 block ciphers encrypt fixed size blocks
 - 23 eg. DES encrypts 64-bit blocks, with 56-bit key
- 24 To apply DES in variety of applications, five “modes of application” have been defined.
 - 23 Electronic Code book(ECB)
 - 24 Cipher Block chaining(CBC)
 - 25 Cipher Feedback(CFB)
 - 26 Output Feedback(OFB)
 - 27 Counter(CTR)
- 25 These modes are intended for use with any symmetric block cipher including DES and AES.

Electronic Code Book:

- 5888 Message is broken into independent blocks which are encrypted.
- 5889 Each block is a value which is substituted, like a codebook, hence name.
- 5890 Each block is encoded independently of the other blocks.
 - $C_i = DES_{K1}(P_i)$
- 23 uses: secure transmission of single values.

Advantages and Limitations of ECB:

- 5888 Repetitions in message may show in cipher text
- 5889 if aligned with message block
- 5890 particularly with data such as graphics
- 5891 or with messages that change very little, which become a code-book analysis problem
- 5892 Weakness due to encrypted message blocks being Independent.
- 5893 main use is sending a few blocks of data

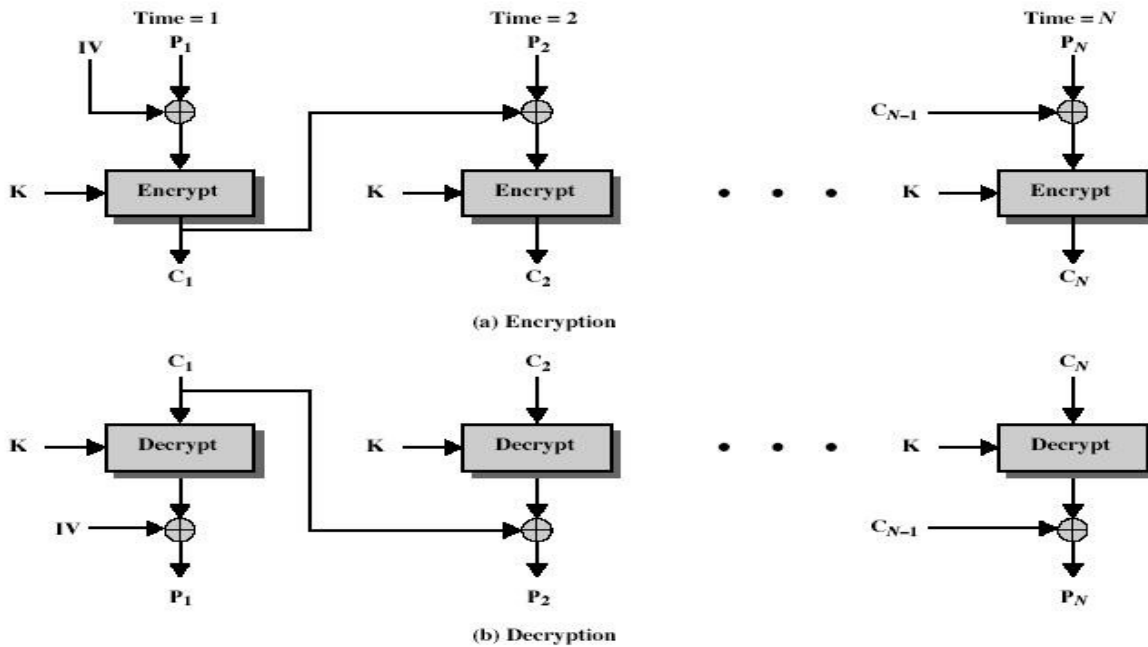


Cipher Block Chaining (CBC):

- 0 Message is broken into blocks.
- 1 but these are linked together in the encryption operation.
- 2 Each previous cipher blocks is chained with current plaintext block, hence name.
- 3 Use Initial Vector (IV) to start process.
- 4 $C_i = \text{DES}_K(P_i \oplus C_{i-1})$
- 5 $C_0 = \text{IV}$
- 6 Uses: bulk data encryption, authentication.

Advantages and Limitations of CBC:

- 0 each ciphertext block depends on **all** message blocks
- 1 thus a change in the message affects all ciphertext blocks after the change as well as the original block
- 2 need **Initial Value** (IV) known to sender & receiver
 - 2.0 however if IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
 - 2.1 hence either IV must be a fixed value or it must be sent encrypted in ECB mode before rest of message
- 3 at end of message, handle possible last short block
 - 3.0 by padding either with known non-data value (eg nulls)
 - 3.1 or pad last block with count of pad size
 - 1.0 eg. [b1 b2 b3 0 0 0 0 5] ← 3 data bytes, then 5 bytes pad+count



Cipher Feedback (CFB):

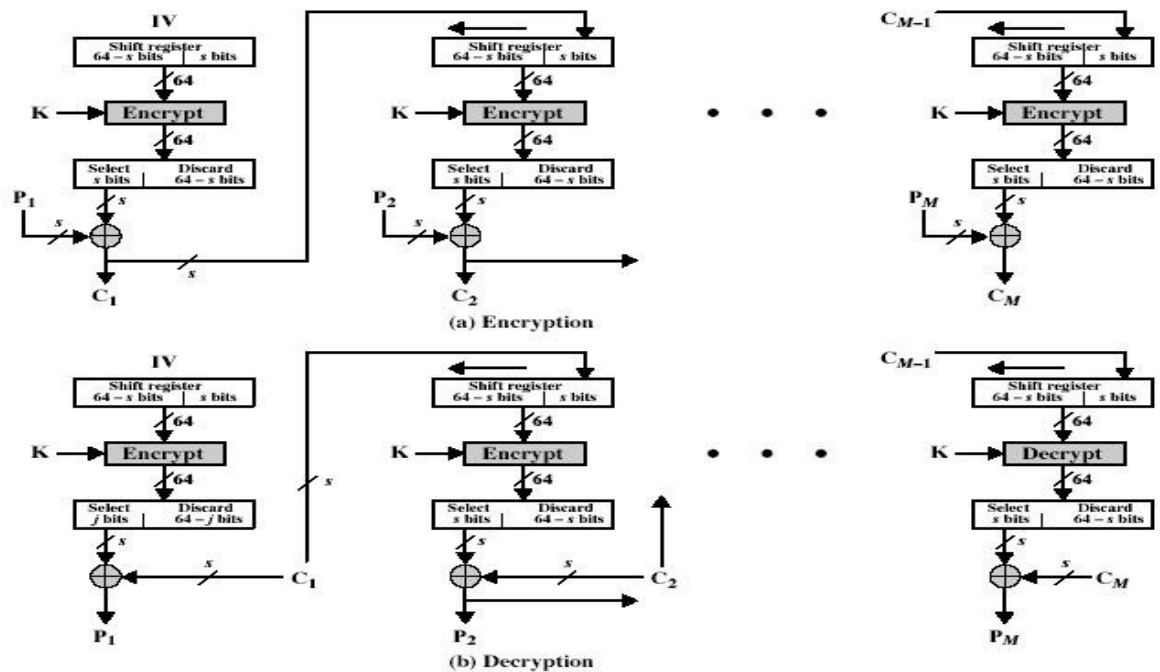
- 0 Message is treated as a stream of bits.
- 1 Added to the output of the block cipher .
- 2 Result is feed back for next stage (hence name).
- 3 Standard allows any number of bit (1,8 or 64 or whatever) to be feed back
 - 0 denoted CFB-1, CFB-8, CFB-64 etc
- 4 Is most efficient to use all 64 bits (CFB-64).

$$C_i = P_i \text{ XOR } \text{DES}_{K1}(C_{i-1})$$

$$C_{-1} = \text{IV}$$
- 0 Uses: stream data encryption, authentication.

Advantages and Limitations of CFB:

- 23 Appropriate when data arrives in bits/bytes .
- 24 Most common stream mode.
- 25 Note that the block cipher is used in encryption mode at both ends.
- 26 Errors propagate for several blocks after the error.



Output FeedBack (OFB):

The alternative to CFB is OFB. Here the generation of the "random" bits is independent of the message being encrypted. The advantage is that firstly, they can be computed in advance, good for bursty traffic, and secondly, any bit error only affects a single bit. Thus this is good for noisy links (eg satellite TV transmissions etc).

Message is treated as a stream of bits
 Output of cipher is added to message
 Output is then feedback (hence name)
 Feedback is independent of message
 Can be computed in advance

$$C_i = P_i \text{ XOR } O_i$$

$$O_i = \text{DES}_{K1}(O_{i-1})$$

$$O_{-1} = \text{IV}$$

Uses: stream encryption over noisy channels.

Advantages and Limitations of OFB:

Used when error feedback a problem or where need to encryptions before message is available.

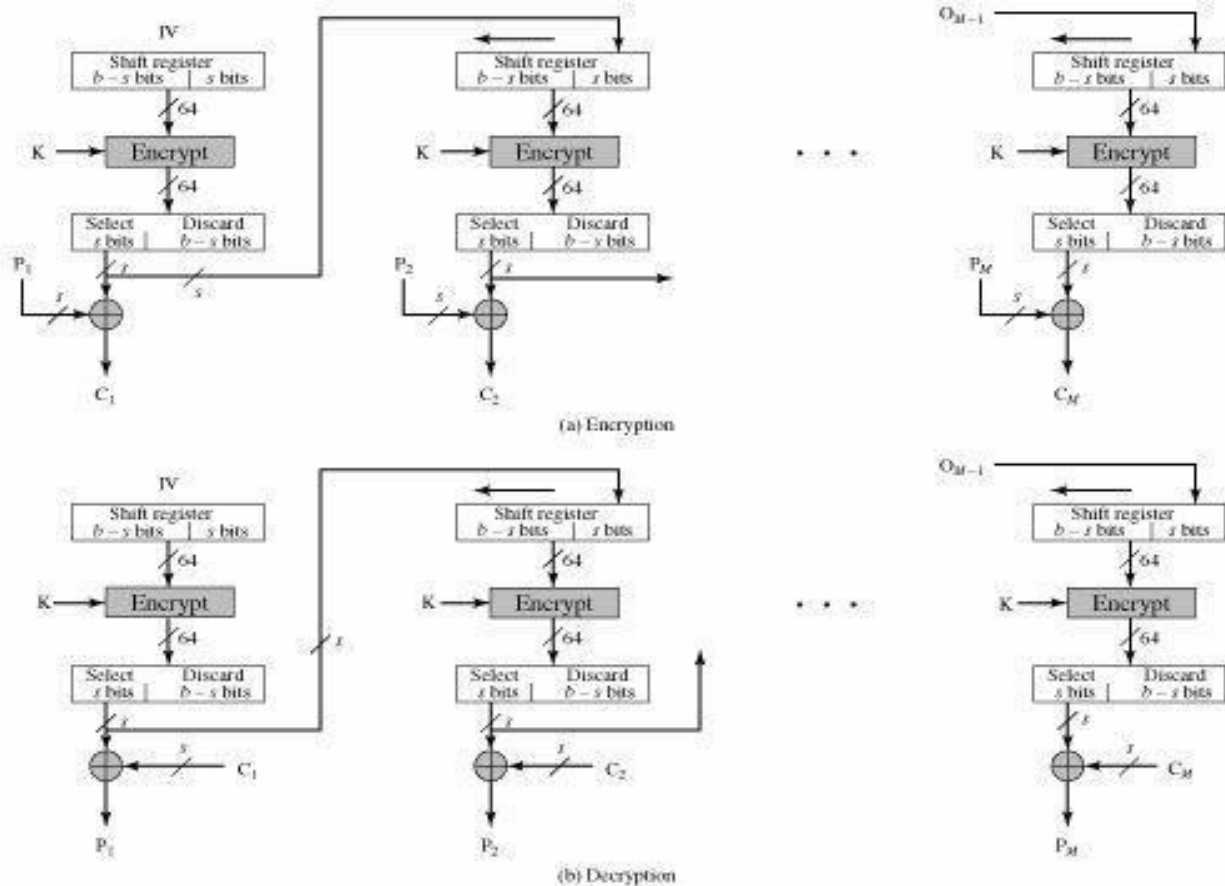
Superficially similar to CFB.

But feedback is from the output of cipher and is independent of message..

sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs

Originally specified with m-bit feedback in the standards.

Subsequent research has shown that only OFB-64 should ever be used.



Comparison of Different Modes

Operation Mode	Description	Type of Result	Data Unit Size
ECB	Each n -bit block is encrypted independently with the same cipher key.	Block cipher	n
CBC	Same as ECB, but each block is first exclusive-ored with the previous ciphertext.	Block cipher	n
CFB	Each r -bit block is exclusive-ored with an r -bit key, which is part of previous cipher text	Stream cipher	$r \leq n$
OFB	Same as CFB, but the shift register is updated by the previous r -bit key.	Stream cipher	$r \leq n$
CTR	Same as OFB, but a counter is used instead of a shift register.	Stream cipher	n

2.8 Location of encryption devices:

The most powerful, and most common, approach to countering the threats to network security is encryption. In using encryption, need to decide what encrypt and where the encryption gear should be located. There are two fundamental alternatives:

1. link encryption

0 End -to end encryption.

Link encryption:

With the link encryption, each vulnerable communications link is equipped on both ends with an encryption devices. Thus, all traffic over all communications link is secured. It provides high level of security. One disadvantage of this approach is that the message must be encrypted each time it enters a packet switch. Thus, the message is vulnerable at each switch. If this is public packet-switching network, the user has no control over the security.

End -to end encryption:

With the end-to-end encryption, the encryption process is carried out at the two end systems. The source host or terminal encrypts the data. The data, in encrypted form, are then transmitted unaltered across the network to the destination terminal or host. The destination shares a key with the source and so is able to decrypt the data. This approach would seem to secure the transmission against attacks on the network links or switches. There is still a weak spot. The packet switching node will receive an encrypted packet and be unable to read the header. Therefore, it will not be able to route the packet. It follows that the host may have only encrypt the user data portion of the packet and must leave the header in the clear, so that it can be read by the network.

Thus, with end-to-end encryption, the user data are secure. However the traffic pattern is not, because packet headers are transmitted in the clear. To achieve greater security, both link and end-to-end encryption are needed, as shown below.

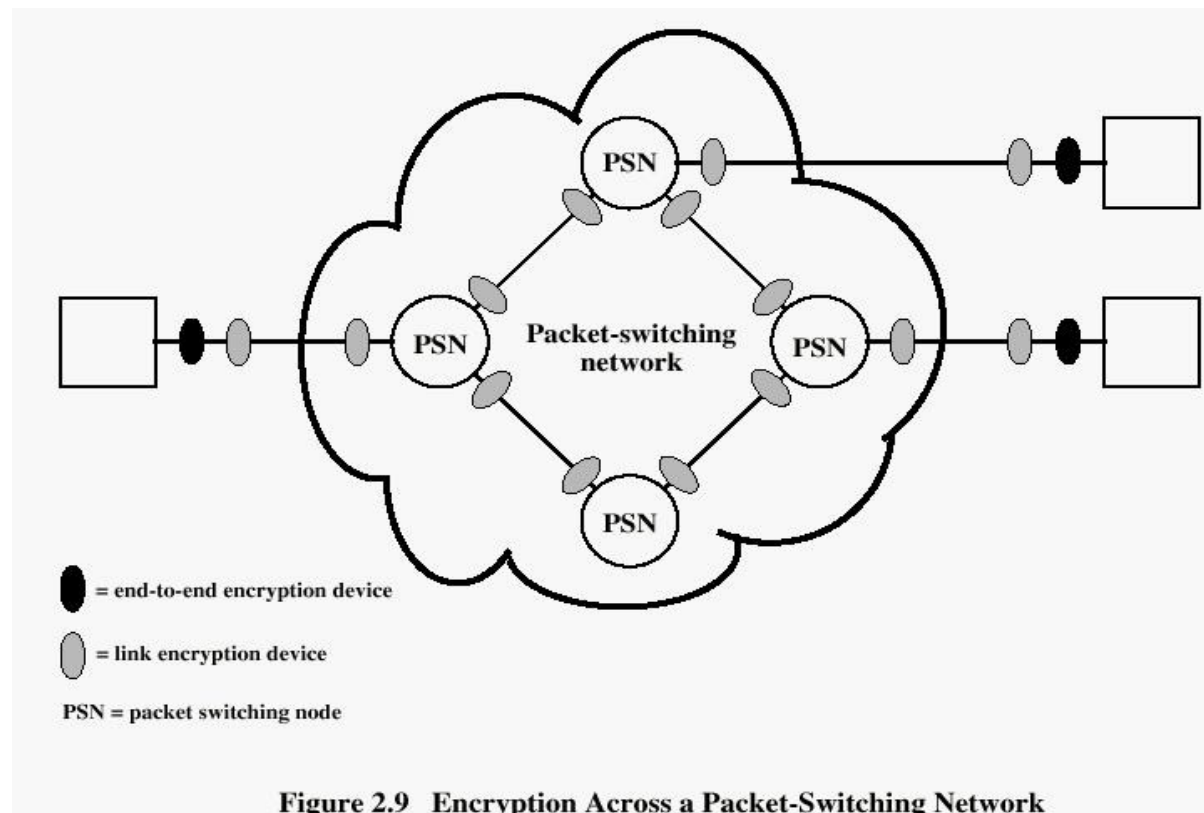


Figure 2.9 Encryption Across a Packet-Switching Network

Key distribution:

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others.

Key distribution can be achieved in a number of ways. For parties A and B:

A key could be selected by A and physically delivered to B.

A third party could select the key and physically deliver it to A and B.

If A and B have previously used a key, one party could transmit the new key to the other, encrypted using the old key.

If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. But, delivering a key manually is not always possible.

Option 3 is a possibility for either link or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys are revealed.

To provide keys for end-to-end encryption, option 4 is preferable. Figure 2.9 shows an implementation which satisfies option 4. For this scheme two kinds of keys are identified:

0 Session key:

23 Data encrypted with a one-time session key. At the conclusion of the session the key is destroyed

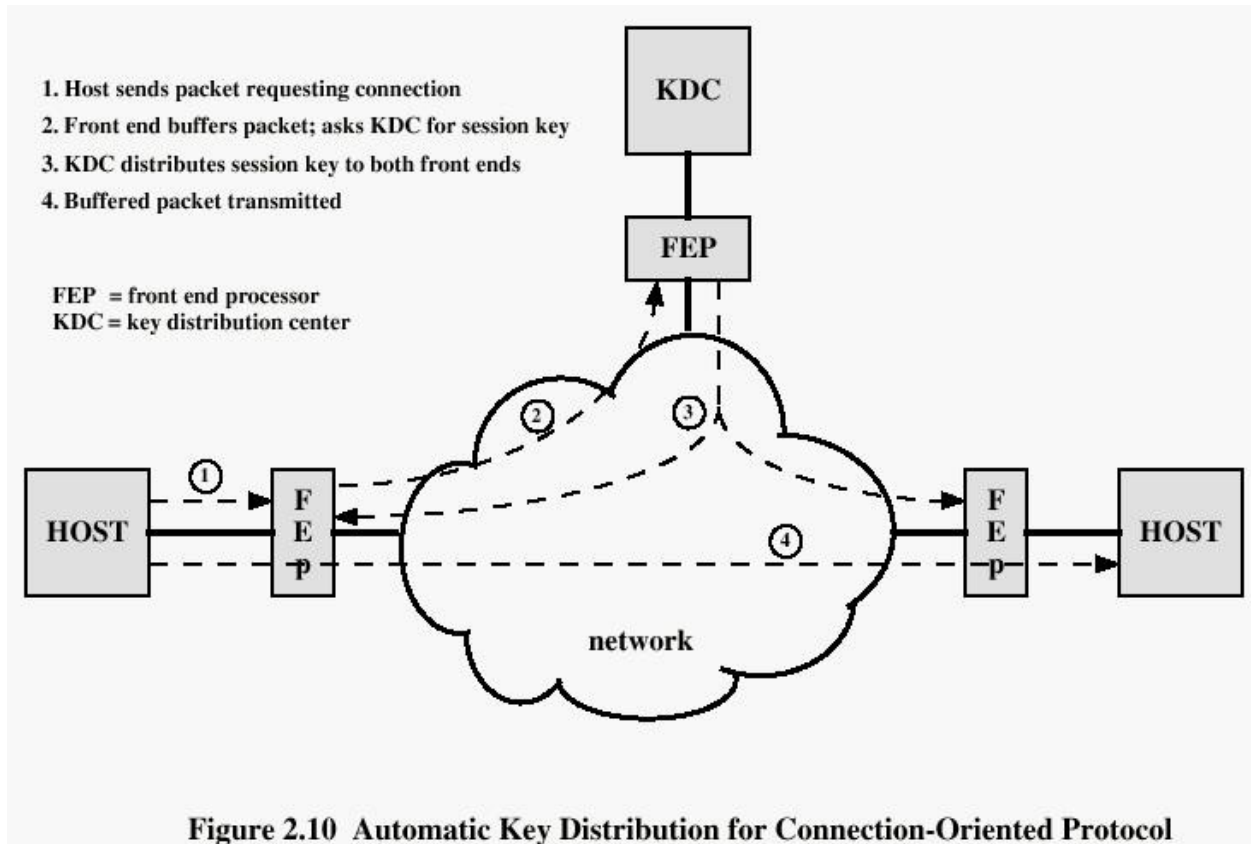
1 Permanent key:

23 Used between entities for the purpose of distributing session keys

The configuration consists of the following elements:

Key distribution center: the key distribution center (KDC) determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the distribution center provides a one-time session key for that connection.

Security service module (SSM): This module, which may consist of functionality at one protocol layer, performs end-to-end encryption and obtains session keys on behalf of users.



The steps involved in establishing a connection are shown in fig above.

2.9 Approaches of Message Authentication:

Encryption protects against passive attack (eavesdropping). A different requirement is to protect against active attack (falsification of data and transactions). Protection against such attacks is known as message authentication.

A message, file, document, or other collection of data is said to be authentic when it is genuine and comes from its alleged source. Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. We may also wish to verify a message's timeliness (it has not been artificially delayed and replayed) and sequence relative to other messages flowing between two parties.

5888 Authentication Using Conventional Encryption

It would seem possible to perform authentication simply by the use of symmetric encryption. If we assume that only the sender and receiver share a key (which is as it should be), then only the genuine sender would be able to encrypt a message successfully for the other participant, provided the receiver can recognize a valid message. Furthermore, if the message includes an error-detection code and a sequence number, the receiver is assured that no alterations have been made and that sequencing is proper. If the message also includes a timestamp, the receiver is assured that the message has not been delayed beyond that normally expected for network transit.

5888 Message Authentication without Message Encryption

In this section, we examine several approaches to message authentication that do not rely on encryption. In all of these approaches, an authentication tag is generated and appended to each message for transmission. The message itself is not encrypted and can be read at the destination independent of the authentication function at the destination.

Three situations in which message authentication without confidentiality is preferable:

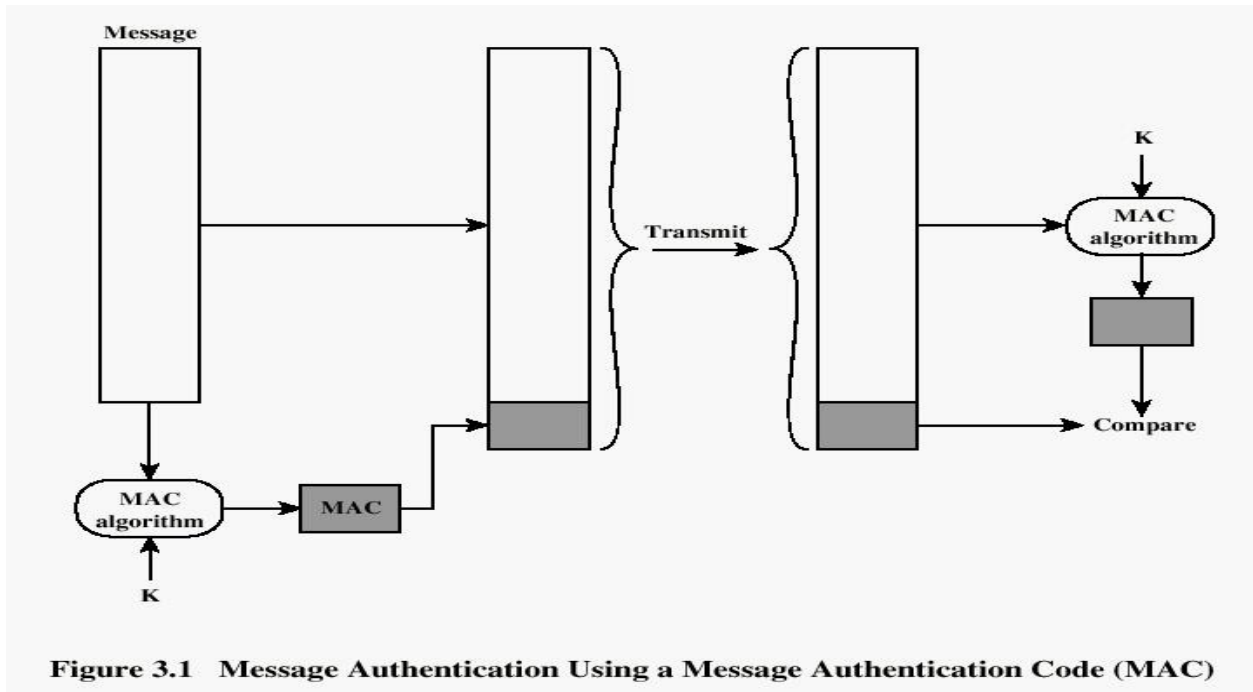
- 4 There are a number of applications in which the same message is broadcast to a number of destinations.
- 5 Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis with messages being chosen at random for checking.
- 6 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 tag were attached to the program, it could be checked when ever assurance is required of the integrity of the program.

MESSAGE AUTHENTICATION CODE One authentication technique involves the use of a secret key to generate a small block of data, known as a **message authentication code (MAC)**, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K_{AB} . When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: $MAC_M = F(K_{AB}, M)$. The message plus code 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 message authentication code. The received code is compared to the calculated code (Figure 3.1). If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then the following statements apply:

5888 The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker cannot alter the code to correspond to the alterations in the message.

5889 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 code.

5890 If the message includes a sequence number (such as is used with HDLC and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.



A number of algorithms could be used to generate the code. DES is used to generate an encrypted version of the message, and the last number of bits of ciphertext is used as the code. A 16- or 32-bit code is typical.

The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible, as it must for decryption. Because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

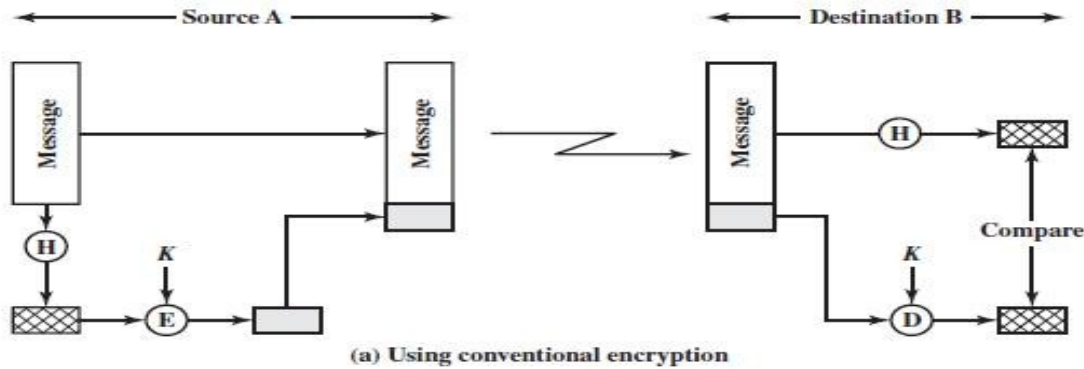
ONE-WAY HASH FUNCTION An alternative to the message authentication code is the **one-way hash function**. As with the message authentication code, a hash function accepts a variable-size message M as input and produces a fixed-size message digest $H(M)$ as output. Unlike the MAC, a hash function does not take a secret key as input. To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic.

Figure below represents three ways in which the message can be authenticated.

- 23 Using conventional encryption.
- 24 Using public key encryption.
- 25 Message Authentication Using a One-Way Hash Function

23 Using conventional encryption.

The message digest can be encrypted using conventional encryption; if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured.



5888 Using public key encryption.

The message digest can be encrypted using public-key encryption. The public-key approach has two advantages:

23 It provides a digital signature as well as message authentication.

24 It does not require the distribution of keys to communicating parties.

These two approaches also have an advantage over approaches that encrypt the entire message in that less computation is required. Nevertheless, there has been interest in developing a technique that avoids encryption altogether. Several reasons for this interest are pointed out in:

5888 Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.

5889 Encryption hardware costs are nonnegligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.

5890 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.

5891 An encryption algorithm may be protected by a patent.

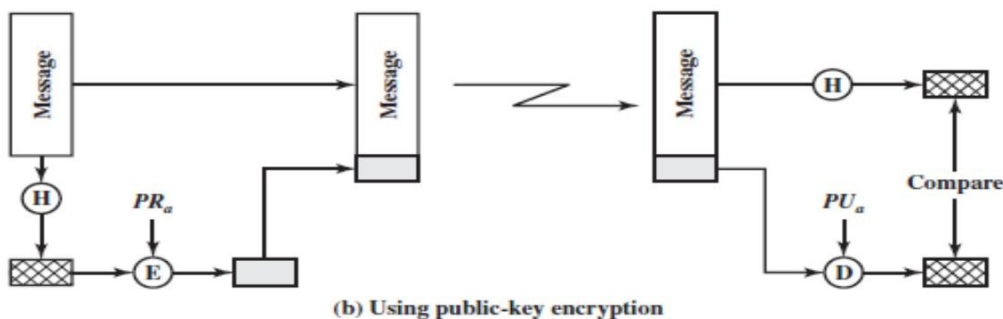


Figure 3.2c shows a technique that uses a hash function but no encryption for message authentication. This technique assumes that two communicating parties, say A and B, share a common secret value S_{AB} . When A has a message to send to B, it calculates the hash function over the concatenation of the secret value and the message: $MD_M = H(S_{AB}||M)$.² It then sends $[M||MD_M]$ to B. Because B possesses S_{AB} , it can recompute $H(S_{AB}||M)$ and verify MD_M . Because the secret value itself is not sent, it is not possible for an attacker to modify an intercepted message. As long as the secret value remains secret, it is also not possible for an attacker to generate a false message.

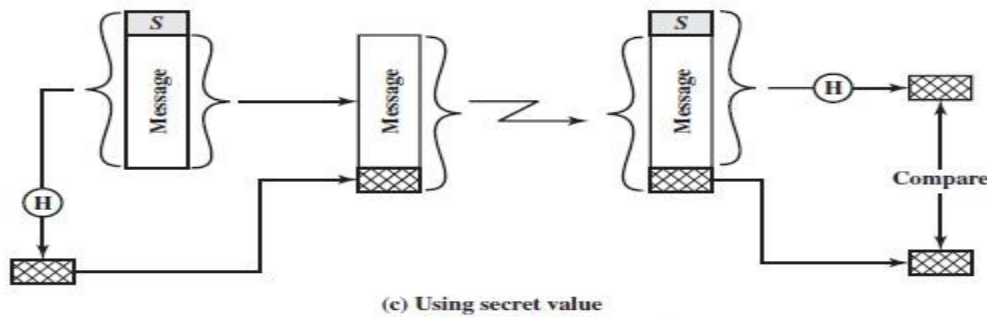


Figure 3.2 Message Authentication Using a One-Way Hash Function

2.9 Secure HASH Functions:

- 5888 Purpose of the HASH function is to produce a "fingerprint".
- 5889 Properties of a HASH function H :
- 5890 H can be applied to a block of data at any size
- 5891 H produces a fixed length output
- 5892 $H(x)$ is easy to compute for any given x .
- 5893 For any given block x , it is computationally infeasible to find x such that $H(x) = h$. this property is referred to as one-way or preimage resistant
- ▶ For any given block x , it is computationally infeasible to find with $H(y) = H(x)$. This property is referred to as second preimage resistant/weak collision resistant.
- ◻◻◻ It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This property is referred to as collision resistant/strong collision resistant.

Simple Hash Function

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

where

- C_i = i th bit of the hash code, $1 \leq i \leq n$
 m = number of n -bit blocks in the input
 b_{ij} = i th bit in j th block
 \oplus = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n -bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n} . With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{-128} , the hash function on this type of data has an effectiveness of 2^{-112} .

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.

1. Initially set the n -bit hash value to zero.
2. Process each successive n -bit block of data as follows:
 - a. Rotate the current hash value to the left by one bit.
 - b. XOR the block into the hash value.

This has the effect of “randomizing” the input more completely and overcoming any regularities that appear in the input. Figure 11.4 illustrates these two types of hash functions for 16-bit hash values.

	bit 1	bit 2	• • •	bit n
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block m	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

2.10 Secure Hash Algorithm:

- 23 SHA originally designed by NIST in 1993
- 24 was revised in 1995 as SHA-1
- 25 US standard for use with DSA signature scheme
- 26 based on design of MD4 with key differences
- 27 produces 160-bit hash values
- 28 Recent 2005 results on security of SHA-1 have raised concerns on its use in future applications.
- 29 NIST issued revision FIPS 180-2 in 2002
- 30 adds 3 additional versions of SHA
 - 23 SHA-256, SHA-384, SHA-512
- 31 designed for compatibility with increased security provided by the AES cipher
- 32 structure & detail is similar to SHA-1
- 33 hence analysis should be similar
- 34 but security levels are rather higher

Comparison of SHA Parameters

Table 3.1 Comparison of SHA Parameters

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.

2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$.

Message Digest Generation Using SHA-1:

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Figure below depicts the overall processing of a message to produce a digest. The processing consists of the following steps.

Step 1 Append padding bits: The message is padded so that its length is congruent to 896 modulo 1024 [$\text{length} = 896 \pmod{1024}$]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. The padding consists of a single 1 bit followed by the necessary number of 0 bits.

Step 2 Append length: A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).

Step 3 Initialize hash buffer: A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

$a = 6A09E667F3BCC908$	$e = 510E527FADE682D1$
$b = BB67AE8584CAA73B$	$f = 9B05688C2B3E6C1F$
$c = 3C6EF372FE94F82B$	$g = 1F83D9ABFB41BD6B$
$d = A54FF53A5F1D36F1$	$h = 5BE0CD19137E2179$

These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

Step 4 Process message in 1024-bit (128-word) blocks: The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 3.4. The logic is illustrated in Figure 3.5.

Each round takes as input the 512-bit buffer value, $abcdefgh$, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1} . Each round t makes use of a 64-bit value W_t , derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t , where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a “randomized” set of 64-bit patterns, which should eliminate any regularities in the input data. Table 11.4 shows these constants in hexadecimal format (from left to right).

The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i . The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1} , using addition modulo 2^{64} .

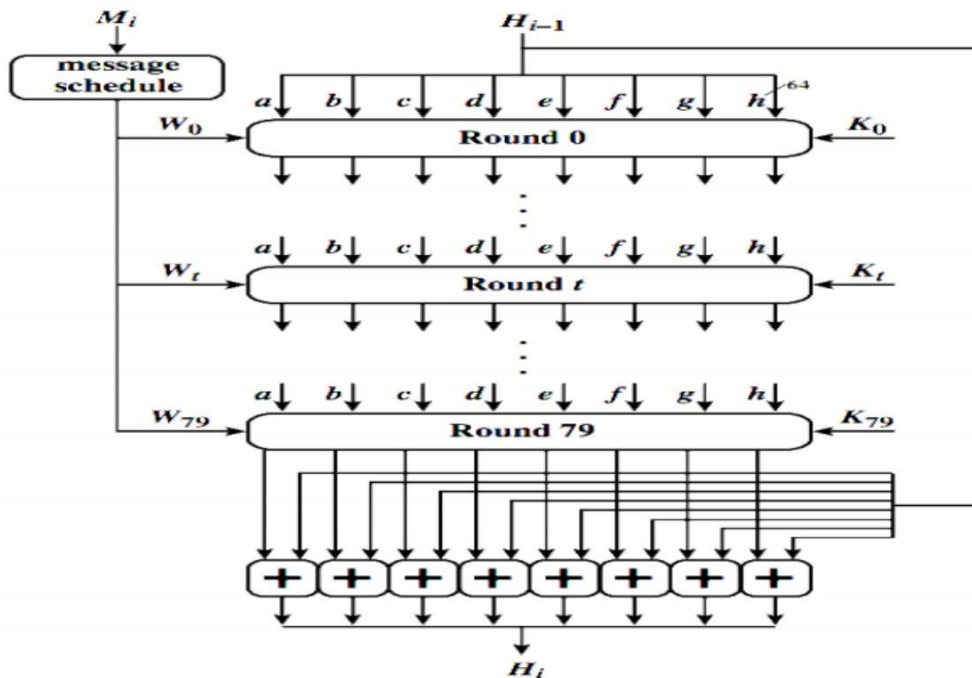


Figure 3: compression function

Step 5 Output: After all N 1024-bit blocks have been processed, the output from the N th stage is the 512-bit message digest.

We can summarize the behavior of SHA-512 as follows:

$$H_0 = IV$$

$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$

$$MD = H_N$$

where

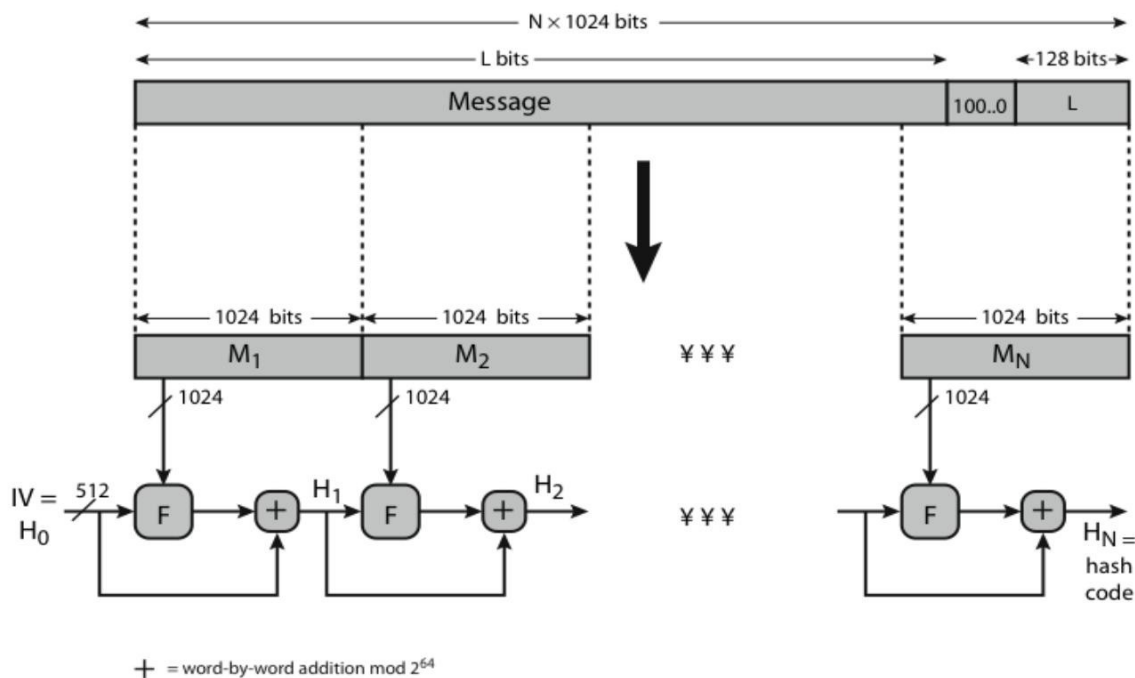
IV = initial value of the abcdefgh buffer, defined in step 3

abcdefgh_i = the output of the last round of processing of the i th message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value



2.10 Hash and MAC Algorithms

5888 Hash Functions

5888 condense arbitrary size message to fixed size

5889 Processing message in blocks.

5890 Some compression function.

5891 Either custom or block cipher based.

5889 Message Authentication Code (MAC)

- 23 Fixed sized authenticator for some message.
- 24 To provide authentication for message.
- 25 By using block cipher mode or hash function.

HMAC

23 Motivations:

- 23 Cryptographic hash functions executes faster in software than encryption algorithms such as DES.
- 24 Library code for cryptographic hash functions is widely available.
- 25 No export restrictions from the US.

HMAC DESIGN OBJECTIVES

- 5888 RFC 2104 lists the following design objectives for HMAC.
- 5889 To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.
- 5890 To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- 5891 To preserve the original performance of the hash function without incurring a significant degradation.
- 5892 To use and handle keys in a simple way.
- 5893 To have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function.

HMAC ALGORITHM Figure 3.6 illustrates the overall operation of HMAC. The following terms are defined:

H = embedded hash function (e.g., SHA-1)

M = message input to HMAC (including the padding specified in the embedded hash function)

Y_i = i th block of M , $0 \leq i \leq (L - 1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key; if key length is greater than b , the key is input to the hash function to produce an n -bit key; recommended length is $> n$

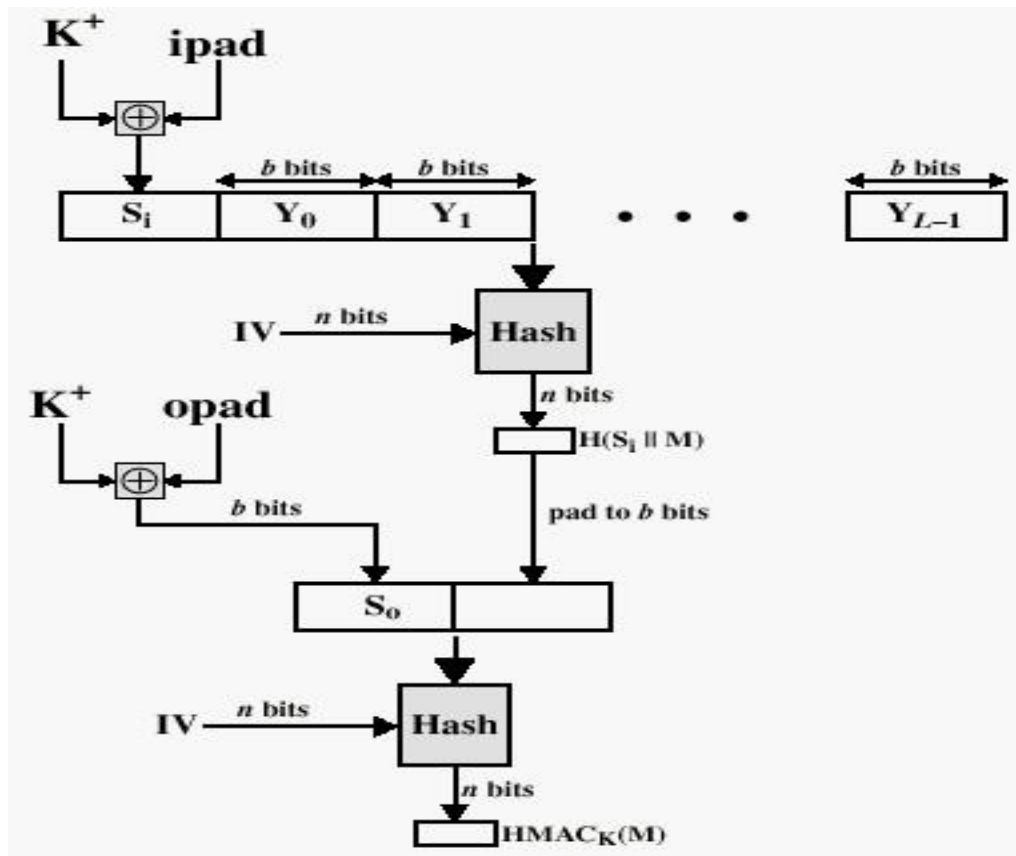
K^+ = K padded with zeros on the left so that the result is b bits in length

ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times

opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

Then HMAC can be expressed as

$$\text{HMAC}(K, M) = H[(K^+ \oplus \text{opad}) \parallel H[(K^+ \oplus \text{ipad}) \parallel M]]$$



In words, HMAC is defined as follows:

1. Append zeros to the left end of K to create a b -bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zero bytes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b -bit block S_i .
3. Append M to S_i .
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b -bit block S_o .
6. Append the hash result from step 4 to S_o .
7. Apply H to the stream generated in step 6 and output the result.

UNIT-III: CRYPTOGRAPHY AND APPLICATIONS

Public key cryptography principles, public key cryptography algorithms, Digital signatures, RSA, Elliptic Algorithms, Digital Certificates, Certificate Authority and key management, Kerberos, X.509 Directory Authentication Service.

Introduction:

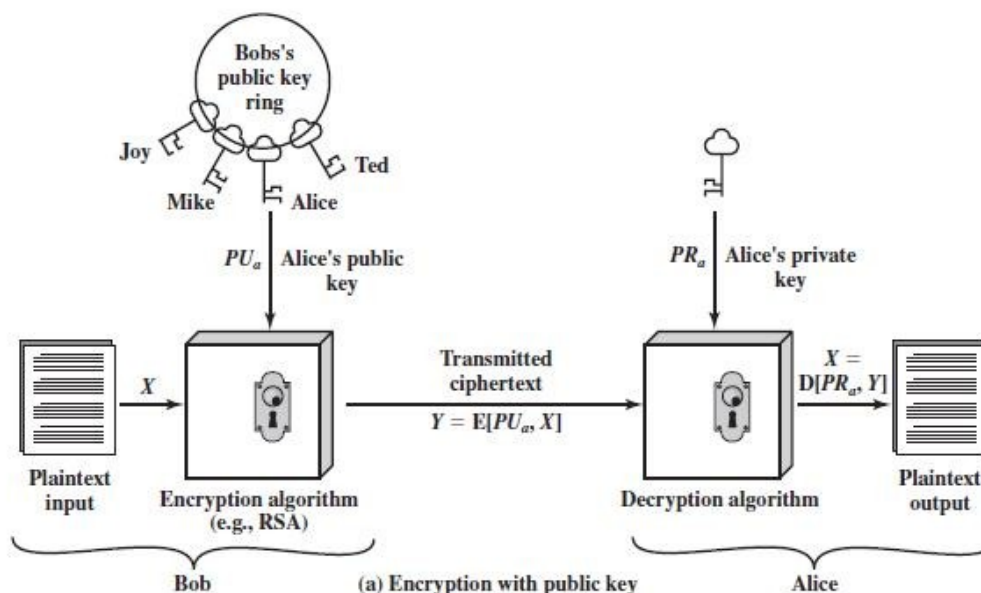
3.1 Public key cryptography principles

Public-key encryption, which finds use in message authentication and key distribution.

1. Public-Key Encryption Structure:

Public-key algorithms are based on mathematical functions rather than on simple operations on bit patterns, such as are used in symmetric encryption algorithms. More important, public-key cryptography is asymmetric, involving the use of two separate keys—in contrast to the symmetric conventional encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

Before proceeding, we should first mention several common misconceptions concerning public-key encryption. One is that public-key encryption is more secure from cryptanalysis than conventional encryption. In fact, the security of any encryption scheme depends on (1) the length of the key and (2) the computational work involved in breaking a cipher. There is nothing in principle about either conventional 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 conventional encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that conventional encryption will be abandoned. 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 conventional encryption. In fact, some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for conventional encryption. A public-key encryption scheme has six ingredients:



Plaintext: This is the readable message or data that is fed into the algorithm as input.

Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

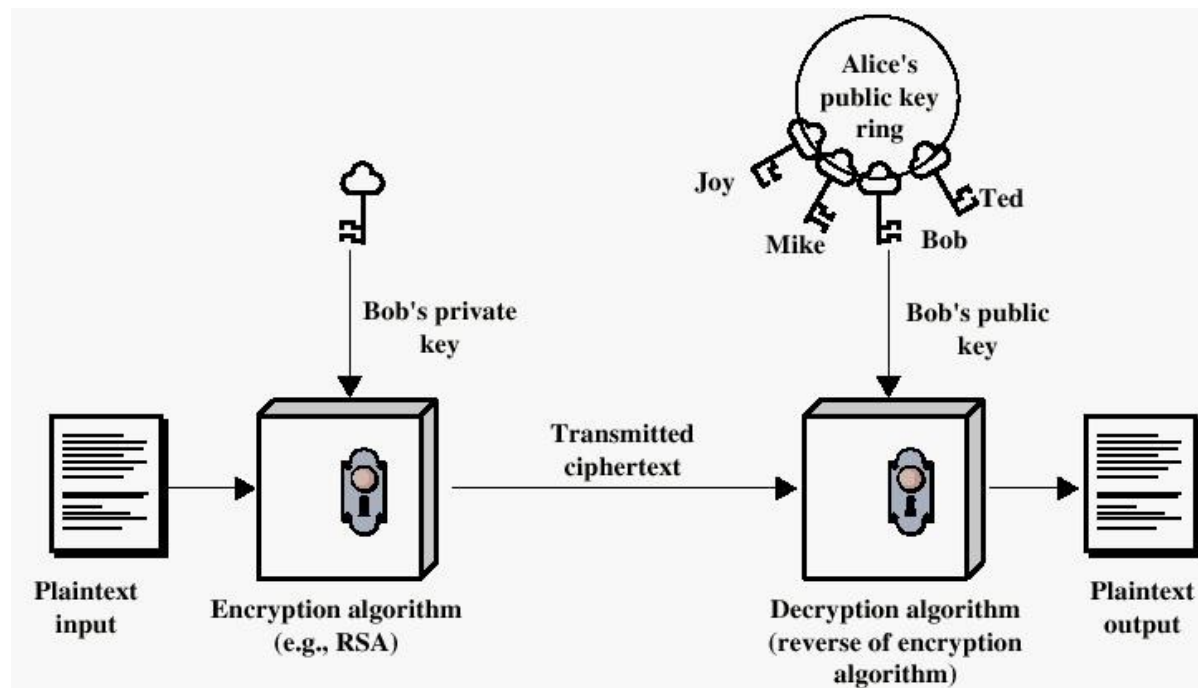
Public and private key: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

As the names suggest, the public key of the pair is made public for others to use, while the private key is known only to its owner. A general-purpose public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption.

Authentication using Public-Key System



The essential steps are the following:

5888 Each user generates a pair of keys to be used for the encryption and decryption of messages.

5889 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 Figure (a) above suggests, each user maintains a collection of public keys obtained from others.

5890 If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice's public key.

5891 When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

The key used in conventional encryption is typically referred to as a **secret key**. The two keys used for public-key encryption are referred to as the **public key** and the **private key**.

2. Applications for Public-Key Cryptosystems

In broad terms, we can classify the use of public-key cryptosystems into three categories:

Encryption/decryption: The sender encrypts a message with the recipient's public key.

Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications.

3. Requirements for Public-Key Cryptography

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

$$C = E(PU_b, M)$$

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

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

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

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

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

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

Difference between conventional and public key algorithms:

Conventional Encryption	Public-Key Encryption
<p><i>Needed to Work:</i></p> <ol style="list-style-type: none"> 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. <p><i>Needed for Security:</i></p> <ol style="list-style-type: none"> 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key. 	<p><i>Needed to Work:</i></p> <ol style="list-style-type: none"> 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). <p><i>Needed for Security:</i></p> <ol style="list-style-type: none"> 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

3.2 Publickey cryptography algorithms

- 5888 RSA and Diffie-Hellman
- 5889 RSA - Ron Rives, Adi Shamir and Len Adleman at MIT, in 1977.
- 5888 RSA is a block cipher.
- 5889 The most widely implemented.
- 5890 Diffie-Hellman
- 5888 Echange a secret key securely.
- 5889 Compute discrete logarithms.

The RSA Algorithm – Key Generation

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Example:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \pmod{160} = 1$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$.

Pu={7,187} Pr={23,187}

The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \pmod{187}$. Exploiting the properties of modular arithmetic, we can do this as follows:

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$$

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 \\ = 79,720,245 \bmod 187 = 88$$

23 There are four possible approaches to defeating the RSA algorithm.

Brute force: This involves trying all possible private keys.

Mathematical attacks: There are several approaches, all equivalent in effect to factoring the product of two primes.

Timing attacks: These depend on the running time of the decryption algorithm.

Chosen ciphertext attacks: This type of attack exploits properties of the RSA algorithm.

Exercise:

5888 Perform encryption and decryption using the RSA algorithm, as in Figure 9.5, for the following:

$$5888 \quad p = 3; q = 11, e = 7; M = 5$$

$$5889 \quad p = 5; q = 11, e = 3; M = 9$$

3.3 Diffie-Hellman Key Exchange

23 The purpose of the algorithm is to enable two users to exchange a secret key securely that then can be used for subsequent encryption of messages.

24 Briefly, we can define the discrete logarithm in the following way.

25 First, we define a primitive root of a prime number p as one whose powers generate all the integers from 1 to $p - 1$.

26 That is, if a is a primitive root of the prime number p , then the numbers.

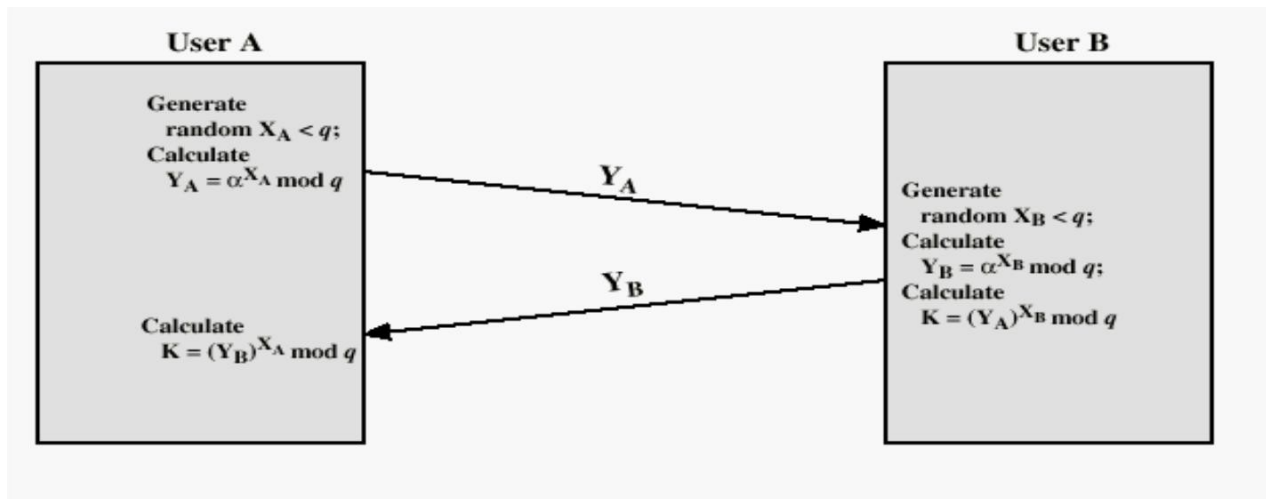
$$a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p$$

are distinct and consist of the integers from 1 through $p - 1$ in some permutation.

For any integer b less than p and a primitive root a of prime number p , one can find a unique exponent i such that

$$b = a^i \bmod p \quad 0 \leq i \leq (p - 1)$$

The exponent i is referred to as the discrete logarithm, or index, of b for the base a , mod p .



Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A	
$K = (Y_B)^{X_A} \bmod q$	

Generation of Secret Key by User B	
$K = (Y_A)^{X_B} \bmod q$	

Example:

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

$$\text{A computes } Y_A = 3^{97} \bmod 353 = 40.$$

$$\text{B computes } Y_B = 3^{233} \bmod 353 = 248.$$

After they exchange public keys, each can compute the common secret key:

$$\text{A computes } K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$$

$$\text{B computes } K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160.$$

We assume an attacker would have available the following information:

$$q = 353; \quad \alpha = 3; \quad Y_A = 40; \quad Y_B = 248$$

Man-In-The-Middle Attack:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} , and then computing the corresponding public keys Y_{D1} and Y_{D2} .
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

3.4 Elliptic Algorithms:

- 5888 Global public elements
 - 5888 $E(a,b)$ - elliptic curve with parameters a , b and q . where q is a prime or an integer of the form 2^m .
 - 5889 G - point on elliptic curve whose order is large value n .
- 5889 User A key generation.
 - Select private n_A $n_A < n$
 - Calculate public P_A $P_A = n_A \times G$
- 23 User B key generation.
 - Select private n_B $n_B < n$
 - Calculate public P_B $P_B = n_B \times G$
- 5888 Generation of secret key by user A
 - 23 $K = n_A \times P_B$

5888 Generation of secret key by user A

5888 $K = n_B \times P_A$

Encryption/Decryption

23 Select a point G and an elliptic Group $E(a,b)$ as parameters.

24 Each user A selects a private key n_A and generates a public key $P_A = n_A \times G$.

25 To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the cipher text C_m consisting of the pair of points.

23 $C_m = \{KG, P_m + KP_B\}$

26 To decrypt the cipher text, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

23 $P_m + KP_B - n_B(KG) = P_m + K(n_B G) - n_B(KG) = P_m$

Merits:

23 A 160 bit ECC has roughly the same security as 1024 bit RSA.

24 Limited memory and computational power.

3.5 Digital signatures:

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

Suppose that Bob wants to send a message to Alice, and although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. In this case, Bob uses his own private key to encrypt the message. When Alice receives the ciphertext, she finds that she can decrypt it with Bob's public key, thus proving that the message must have been encrypted by Bob. No one else has Bob's private key, and therefore no one else could have created a ciphertext that could be decrypted with Bob's public key. Therefore, the entire encrypted message serves as a digital signature. In addition, it is impossible to alter the message without access to Bob'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. Although validating both author and contents, this requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits

that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. A secure hash code such as SHA-1 can serve this function.

Properties:

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other.

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:

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

Thus, the digital signature function includes the authentication function.

Encrypting using private key of their own.

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.

It 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.

A secure hash code such as SHA-1 can serve this function.

►

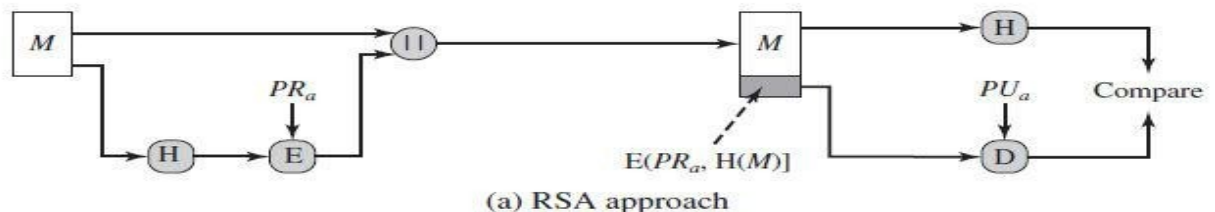
Digital Signature Standard (DSS)

- 0 The National Institute of Standards and Technology (NIST) has published Federal
- 1 Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS).
- 2 The DSS makes use of the Secure Hash Algorithm (SHA).
- 3 The DSS was originally proposed in 1991 and revised in 1993, 1996, 2000, 2009.
- 4 The DSS uses an algorithm that is designed to provide only the digital signature function.
- 5 There are two approaches to digital signatures.

0 RSA approach

1 DSS approach

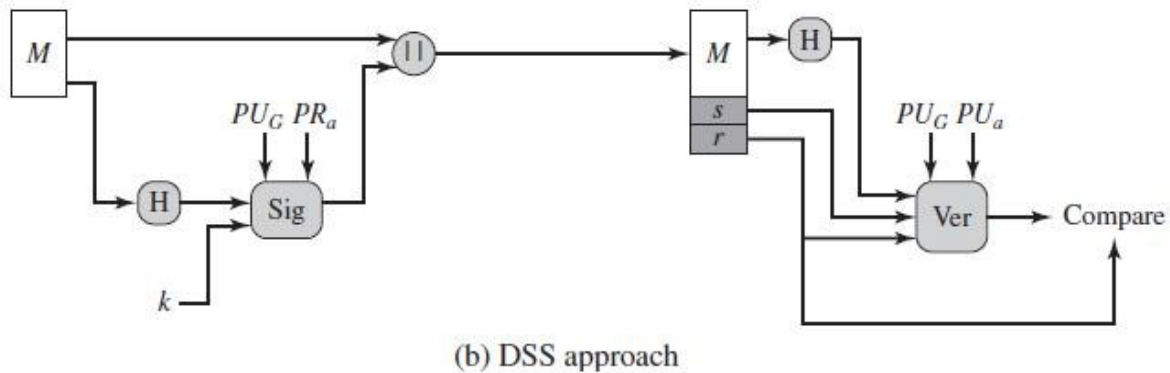
RSA approach:



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.

DSS approach



Where

- Random number (K).
- Sender's private key (KRa).
- Set of parameters known to a group of communicating principals. this set to constitute a global public key (PUG).
- The result is a signature consisting of two components, labeled s and r .

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 generated for this particular signature. The signature function also depends on the sender's private key (PRa) and a set of parameters known to a group of communicating principals. 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 (PUa), 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 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.

We turn now to the details of the algorithm.

Global Public-Key Components

- p prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length of between 512 and 1024 bits
in increments of 64 bits
- q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$;
i.e., bit length of 160 bits
- $g = h^{(p-1)/q} \bmod p$,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

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

User's Public Key

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

User's Per-Message Secret Number

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

Figure 13.4 The Digital Signature Algorithm (DSA)

Signing

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

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

$$\text{Signature} = (r, s)$$

Verifying

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

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

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

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

$$\text{TEST: } v = r'$$

M = message to be signed

$H(M)$ = hash of M using SHA-1

M', r', s' = received versions of M, r, s

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

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

Figure below depicts the functions of signing and verifying.

The structure of the algorithm, as revealed in Figure below, is quite interesting.

Note that the test at the end is on the value r , which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of $k \bmod q$ is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 13.4 or Figure 13.5 that such a scheme would work.

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

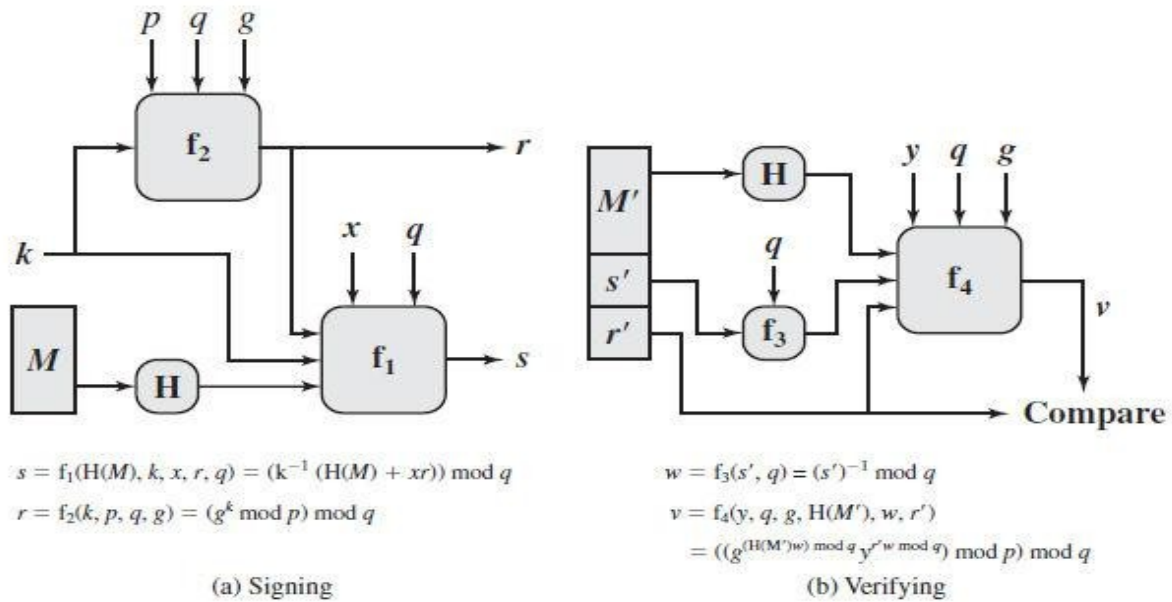
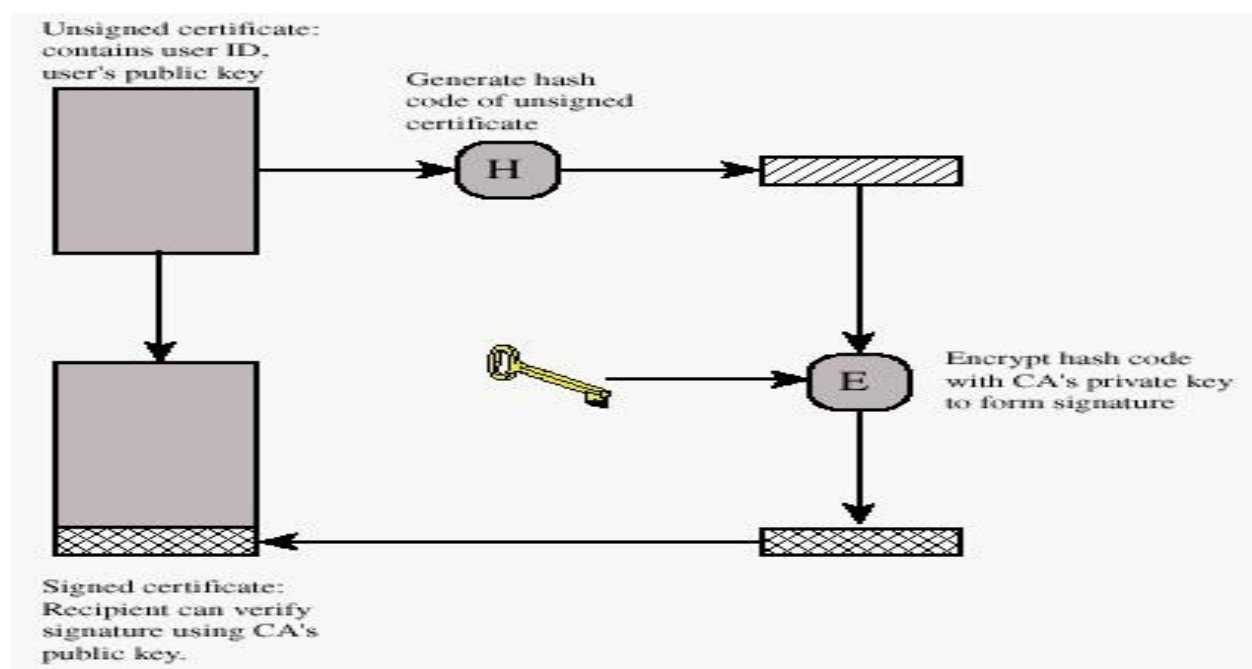


Figure 13.5 DSS Signing and Verifying

3.6 Certificate Authority and key management

- 0 One of the major role of public -key encryption is to address the problem of key distribution.
- 1 There are two distinct aspects to the use of public key encryption in this regard:
 - 1.0** The distribution of public keys.
 - 1.1** The use of public-key encryption to distribute secret keys.

Public key Certificates



Public key distribution of secret keys

One approach is to use RSA

Another approach to use Diffie-Hellman key exchange.

A powerful alternative is the use of public –key certificates. When Bob wishes to communicate with Alice, Bob can do the following.

- Prepare a message.

- Encrypt that message using conventional encryption with a one-time conventional session key.

- Encrypt the session key using public key encryption with Alice's public key.

- Attach the encrypted session key to the message and send it to Alice.

3.7 Kerberos:

What is Kerberos?

- 0 Network Authentication Protocol.
- 0 Developed at MIT in the mid-1980s.
- 1 Available as open source or in supported commercial software requires that each client (each request for service) prove its identity.
- 2 Does not require user to enter password every time a service is requested!
- It provides for strong_ authentication for client-server applications.
- 0 Uses secret-key cryptography to provide this strong authentication.
- ▢ Authentication service for interactive services like telnet, ftp etc.
- ▢ It is fast and allows real time authentication.

Functions of Kerberos:

- ▢ **Authentication**- is the verification of the identity of an involved party and the integrity of the data that the involved party generates.
- ▢ **Integrity** – Is the assurance that the data received is the same as generated.
- ▢ **Confidentiality** – is the protection of info from disclosure to those not intended to receive it.
- ▢ **Authorization** – is the process by which one determines whether a principal is allowed to perform an operation. Authorization is done usually after principal has been authenticated or based on authenticated statements by others.

Motivation

5888 Without knowledge of identity of person requesting an operation difficult to decide if it should be allowed.

5889 Traditional authentication methods are not suitable for use in computer networks where attackers can monitor network traffic and intercept passwords.

5890 Use of strong authentication methods is imperative.

In a common distributed architecture,

Three approaches to security are envisaged:

Rely on individual client work stations to assure identity of user.

Require client systems to authenticate themselves to servers.

Require user to prove identity for each service invoked.

23 In a closed environment where all systems owned and operated by single organization first or second approach may suffice.

24 But in an open environment third approach (supported by Kerberos) needed to protect user information and resources housed on server.

Kerberos Design

☐←☐☐ User must identify himself once at the beginning of a workstation session (login session).

☐←☐☐ Passwords are never sent across the network in clear text (or stored in memory)

☐←☐☐ Every user has a password.

☐←☐☐ Every service has a password.

☐←☐☐ The only entity that knows all the passwords is the *authentication server*.

Provides a centralized authentication server to authenticate users to servers and servers to users.

5888 Relies on conventional encryption, making no use of public-key encryption

5889 Two versions: version 4 and 5

5890 Version 4 makes use of DES.

Kerberos Version 4:

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. This is the protocol with several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

Before discuss kerberos v4. Let's start with simple protocol.

A Simple Authentication Dialogue In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

(1) $C \rightarrow AS: ID_C \| P_C \| ID_V$

(2) $AS \rightarrow C: Ticket$

(3) $C \rightarrow V: ID_C \| Ticket$

$Ticket = E(K_v, [ID_C \| AD_C \| ID_V])$

where

C = client

AS = authentication server

V = server

ID_C = identifier of user on C

ID_V = identifier of V

P_C = password of user on C

AD_C = network address of C

K_v = secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V . The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V . If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C .

Because the ticket is encrypted, it cannot be altered by C or by an opponent. With this ticket, C can now apply to V for service. C sends a message to V containing C 's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

A More Secure Authentication Dialogue: Although the foregoing scenario solvesome of the problems of authentication in an open network environment, problemsremain. Two in particular stand out. First, we would like to minimize the number oftimes that a user has to enter a password. Suppose each ticket can be used only once.If user C logs on to a

workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail-server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme, it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password. An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The new (but still hypothetical) scenario is as follows.

Once per user logon session:

$$(1) C \rightarrow AS: ID_C \parallel ID_{TGS}$$

$$(2) AS \rightarrow C: E(K_C, Ticket_{TGS})$$

Once per type of service:

$$(3) C \rightarrow TGS: ID_C \parallel ID_V \parallel Ticket_{TGS}$$

$$(4) TGS \rightarrow C: Ticket_V$$

Once per service session:

$$(5) C \rightarrow V: ID_C \parallel Ticket_V$$

$$Ticket_{TGS} = E(K_{TGS}, [ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_1 \parallel Lifetime_1])$$

$$Ticket_V = E(K_V, [ID_C \parallel AD_C \parallel ID_V \parallel TS_2 \parallel Lifetime_2])$$

Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password (K_C), which is already stored at the AS. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
4. The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (K_{tgs}) and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

This protocol is having two problems:

- 23 The life associated with the ticket granting ticket. If the life time is very short, then the user will be repeatedly asked for a password. If the life time is long, then an opponent has a greater opportunity for replay.
- 24 There may be requirements for servers to authenticate themselves to users.

These two problems are solved in Kerberosv4.

Kerberos v4 dialogue:

(1) $C \rightarrow AS \quad ID_C \parallel ID_{TGS} \parallel TS_1$

(2) $AS \rightarrow C \quad E(K_c, [K_{c,tgs} \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{TGS}])$

$$Ticket_{TGS} = E(K_{TGS}, [K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) $C \rightarrow TGS \quad ID_v \parallel Ticket_{TGS} \parallel Authenticator_c$

(4) $TGS \rightarrow C \quad E(K_{c,tgs}, [K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v])$

$$Ticket_{TGS} = E(K_{TGS}, [K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_2 \parallel Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \parallel AD_C \parallel TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) $C \rightarrow V \quad Ticket_v \parallel Authenticator_c$

(6) $V \rightarrow C \quad E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \parallel AD_C \parallel TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{TGS}	Tells AS that user requests 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.
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.

Overview of Kerberos:

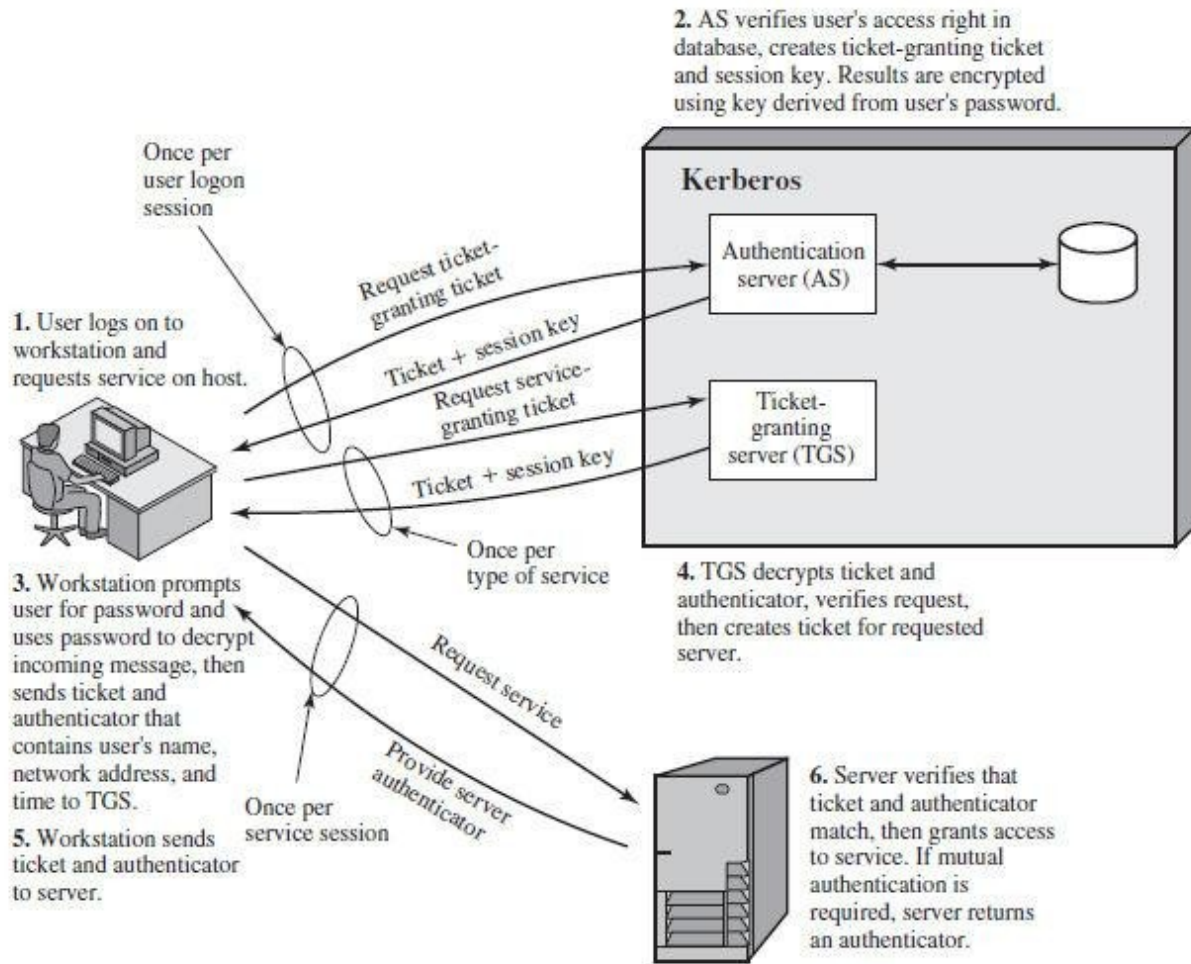


Figure 4.1 Overview of Kerberos

Kerberos realms and multiple kerber:

☞☞☞ A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

5889 The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.

5890 The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

☞☞☞ Such an environment is referred to as a **Kerberos realm**.

The concept of **realm** can be explained as follows.

A Kerberos realm is a set of managed nodes that share the same Kerberos database.

The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room.

A read-only copy of the Kerberos database might also reside on other Kerberos Computer systems.

However, all changes to the database must be made on the master computer system.

Changing or accessing the contents of a Kerberos database requires the Kerberos master password.

A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system.

Each Kerberos principal is identified by its principal name.

Principal names consist of three parts: a service or user name, an instance name, and a realm name.

Networks of clients and servers under different administrative organizations typically constitute different realms.

Users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication.

For two realms to support interrealm authentication, a third requirement is added:

- 23 The Kerberos server in each interoperating realm shares a secret key with The server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users.

With these ground rules in place, we can describe the mechanism as follows :

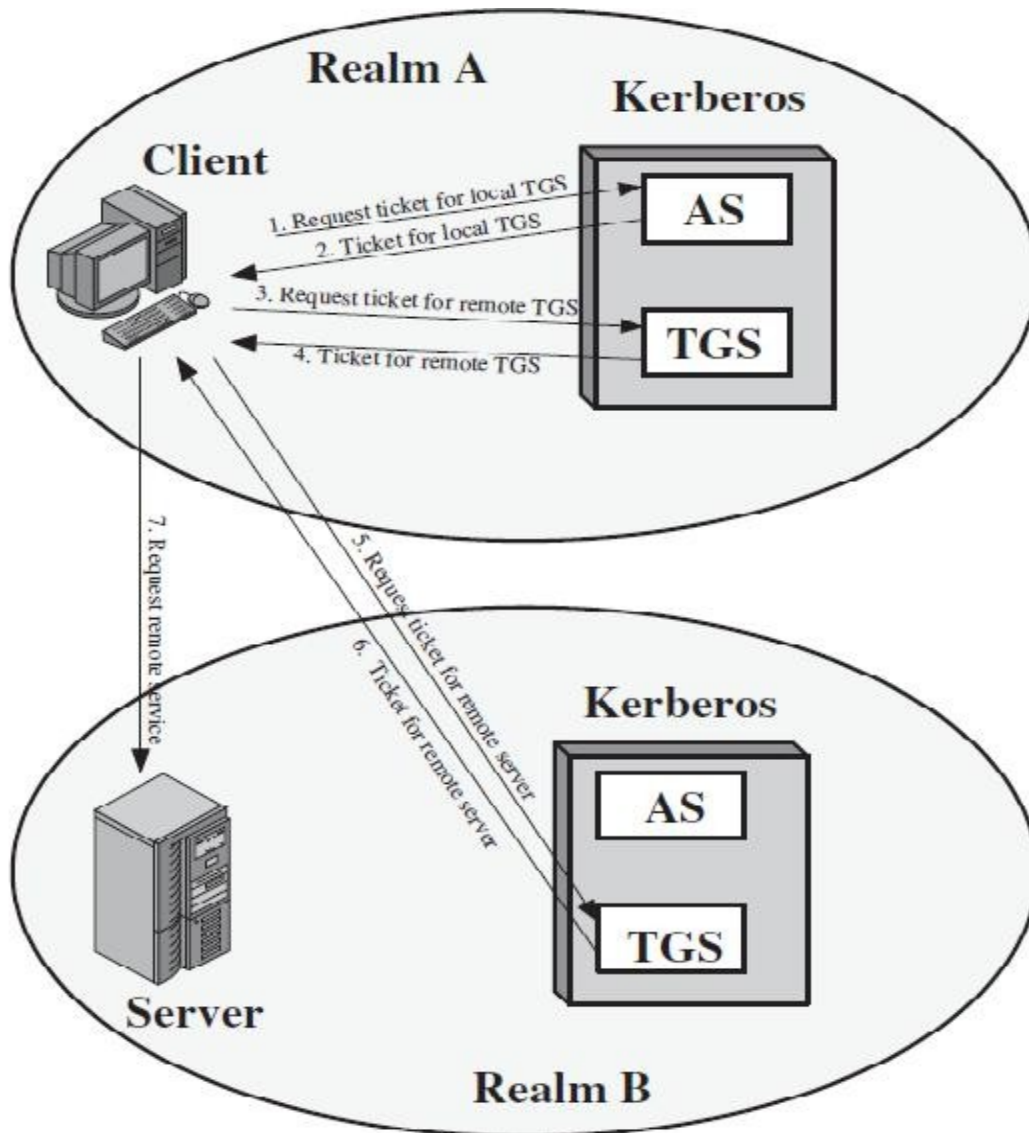
5888 A user wishing service on a server in another realm needs a ticket for that server.

- (1) $C \rightarrow AS: ID_C \parallel ID_{Tgs} \parallel TS_1$
- (2) $AS \rightarrow C: E(K_C, [K_{C,tgs} \parallel ID_{Tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}])$
- (3) $C \rightarrow TGS: ID_{Tgsrem} \parallel Ticket_{tgs} \parallel Authenticator_C$
- (4) $TGS \rightarrow C: E(K_{C,tgs}, [K_{C,tgsrem} \parallel ID_{Tgsrem} \parallel TS_4 \parallel Ticket_{tgsrem}])$
- (5) $C \rightarrow TGS_{rem}: ID_{Vrem} \parallel Ticket_{tgsrem} \parallel Authenticator_C$
- (6) $TGS_{rem} \rightarrow C: E(K_{C,tgsrem}, [K_{C,Vrem} \parallel ID_{Vrem} \parallel TS_6 \parallel Ticket_{Vrem}])$
- (7) $C \rightarrow V_{rem}: Ticket_{Vrem} \parallel Authenticator_C$

23 One problem presented by the foregoing approach is that it does not scale well to many realms.

- If there are N realms, then there must be $N(N-1)/2$ secure key exchanges.

Request for Service in Another Realm:



Kerberos Version 5

23 Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4.

24 Version 5 is intended to address the limitations of version 4 in two areas: **Environmental shortcomings** and **Technical deficiencies**.

5888 Kerberos version 4 did not fully address the need to be of general purpose. Is developed for use within the project Athena environment.

Difference between Version 4 and 5- Environmental shortcomings.

23 Encryption system dependence (V.4 DES)

24 Internet protocol dependence

25 Message byte ordering

26 Ticket lifetime

27 Authentication forwarding

28 Interrealm authentication

Encryption system dependence (V.4 DES)

5888 Version 4 requires the use of DES.

5889 In version 5, cipher text is tagged with an encryption-type identifier so that any encryption technique may be used.

5890 Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.

Internet protocol dependence:

23 Version 4 requires the use of Internet Protocol (IP) addresses.

24 Other address types, such as the ISO network address, are not accommodated.

25 Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

Message byte ordering:

5888 In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address.

5889 In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

Ticket lifetime:

23 Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes.

23 In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

Authentication forwarding:

5888 Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client.

5889 This capability would enable a client to access a server and have that server access another server on behalf of the client.

5890 Version 5 provides this capability.

Interrealm authentication:

5888 In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier.

5889 Version 5 supports a method that requires fewer relationships.

Technical deficiencies:

Double encryption:

Note in Table 4.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client.

The second encryption is not necessary and is computationally wasteful.

PCBC encryption:

Encryption in version 4 makes use of a nonstandard mode of DES known as **propagating cipher block chaining (PCBC)**.

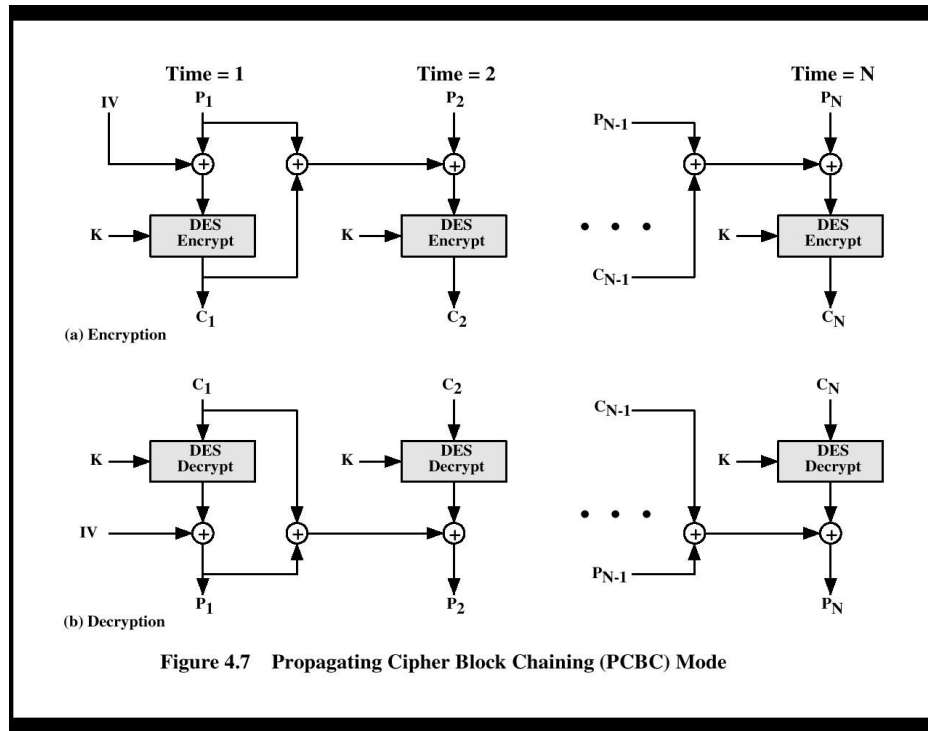
This mode is vulnerable to an attack involving the interchange of ciphertext blocks.

PCBC was intended to provide an integrity Check as part of the encryption operation.

Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption.

In particular, a checksum or hash code is attached to the message prior to encryption using CBC.

PCBC Mode:



Session keys:

Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket.

In addition, the session key subsequently may be used by the client and the server to protect messages passed during that session.

However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server.

In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection.

A new access by the client would result in the use of a new subsession key.

Password attacks:

Both versions are vulnerable to a password attack.

The message from the AS to the client includes material encrypted with a key based on the client's password.

An opponent can capture this message and attempt to decrypt it by trying various passwords.

Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

The version 5 authentication dialogue(Kerberos V 5):

Table 4.3 Summary of Kerberos Version 5 Message Exchanges

(1) C → AS	$Options \parallel ID_c \parallel Realm_c \parallel ID_{tgs} \parallel Times \parallel Nonce_1$
(2) AS → C	$Realm_c \parallel ID_c \parallel Ticket_{tgs} \parallel E(K_c, [K_{c,tgs} \parallel Times \parallel Nonce_1 \parallel Realm_{tgs} \parallel ID_{tgs}])$ $Ticket_{tgs} = E(K_{tgs}, [Flags \parallel K_{c,tgs} \parallel Realm_c \parallel ID_c \parallel AD_c \parallel Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS	$Options \parallel ID_v \parallel Times \parallel Nonce_2 \parallel Ticket_{tgs} \parallel Authenticator_c$
(4) TGS → C	$Realm_c \parallel ID_c \parallel Ticket_v \parallel E(K_{c,tgs}, [K_{c,v} \parallel Times \parallel Nonce_2 \parallel Realm_v \parallel ID_v])$ $Ticket_{tgs} = E(K_{tgs}, [Flags \parallel K_{c,tgs} \parallel Realm_c \parallel ID_c \parallel AD_c \parallel Times])$ $Ticket_v = E(K_v, [Flags \parallel K_{c,v} \parallel Realm_c \parallel ID_c \parallel AD_c \parallel Times])$ $Authenticator_c = E(K_{c,tgs}, [ID_c \parallel Realm_c \parallel TS_1])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V	$Options \parallel Ticket_v \parallel Authenticator_c$
(6) V → C	$E_{K_{c,v}} [TS_2 \parallel Subkey \parallel Seq\#]$ $Ticket_v = E(K_v, [Flags \parallel K_{c,v} \parallel Realm_c \parallel ID_c \parallel AD_c \parallel Times])$ $Authenticator_c = E(K_{c,v}, [ID_c \parallel Realm_c \parallel TS_2 \parallel Subkey \parallel Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

Authentication service exchange:

Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

Realm: Indicates realm of user.

Options: Used to request that certain flags be set in the returned ticket.

Times: Used by the client to request the following time settings in the ticket:

- from: the desired start time for the requested ticket
- till: the requested expiration time for the requested ticket
- rtime: requested renew-till time

- **Nonce: A random value** to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password.

This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5.

Ticket-granting service exchange

Message (3) for both versions includes an authenticator, a ticket, and the name of the requested service.

In addition, version 5 includes requested times and options for the ticket and a nonce-all with functions similar to those of message (1).

Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Client/Server authentication exchange

Several new features appear in version 5.

In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

Subkey: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{C,V}$) is used.

Sequence number: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6).

This message includes the timestamp from the authenticator.

The subkey field, if present, overrides the subkey field, if present, in message (5).

The optional sequence number field specifies the starting sequence number to be used by the client.

Ticket Flags

The flags field included in ticket in version 5 supports expanded functionality compared to that available in version 4.

Table summarizes the flags that may included in a ticket.

Table 4.4 Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

Kerberos - in practice

Currently have two Kerberos versions:

- 4 : restricted to a single realm
- 5 : allows inter-realm authentication, in beta test

Kerberos v5 is an Internet standard
specified in RFC1510, and used by many utilities

To use Kerberos:

- need to have a KDC on your network
- need to have Kerberised applications running on all participating systems

major problem - US export restrictions

Kerberos cannot be directly distributed outside the US in source format (& binary versions must obscure crypto routine entry points and have no encryption)

else crypto libraries must be implemented locally

3.9 X.509 Directory Authentication Service

ITU-T recommendation X.509 is define a directory service.

The directory is, a server or Distributed set of servers that maintains a database about users.

The information includes a mapping from user name to network address, as well as other attributes and information about the users.

The directory may serve as a repository of public-key certificates.

In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

Each certificate contains the public key of a user and is signed with the private key of a CA.

Is used in S/MIME, IP Security, SSL/TLS and SET.

X.509 is based on the use of public-key cryptography and digital signatures.

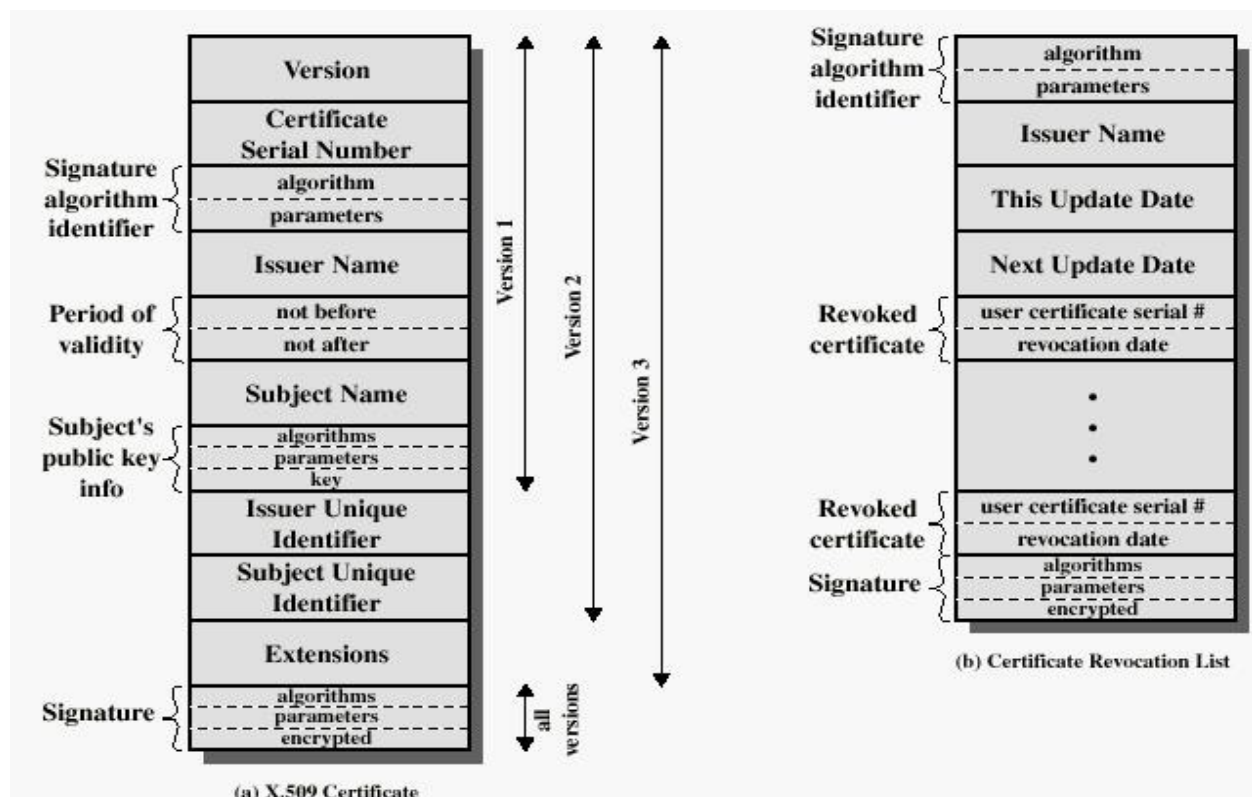
RSA is recommended to use.

Certificates:

The heart of the X.509 scheme is the public-key certificate associated with each user.

These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

X.509 Formats:



Version: Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

Serial number: An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

Signature algorithm identifier: The algorithm used to sign the certificate, together with any associated parameters

Issuer name: X.500 name of the CA that created and signed this certificate.

Period of validity: Consists of two dates: the first and last on which the certificate is valid.

Subject name: The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

Subject's public-key information: of The public key of the subject, plus an identifier the algorithm for which this key isto be used, together with any associated parameters.

Issuer unique identifier: An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

Subject unique identifier: An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

Extensions: A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

Signature: Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

The standard uses the following notation to define a certificate:

$$CA\langle\langle A \rangle\rangle = CA \{V, SN, AI, CA, UCA, A, UA, A_p, T^A\}$$

where

$Y\langle\langle X \rangle\rangle$ = the certificate of user X issued by certification authority Y

$Y\{I\}$ = the signing of I by Y ; consists of I with an encrypted hash code appended

V = version of the certificate

SN = serial number of the certificate

AI = identifier of the algorithm used to sign the certificate

CA = name of certificate authority

UCA = optional unique identifier of the CA

A = name of user A

UA = optional unique identifier of the user A

A_p = public key of user A

T^A = period of validity of the certificate

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

This is the typical digital signature approach.

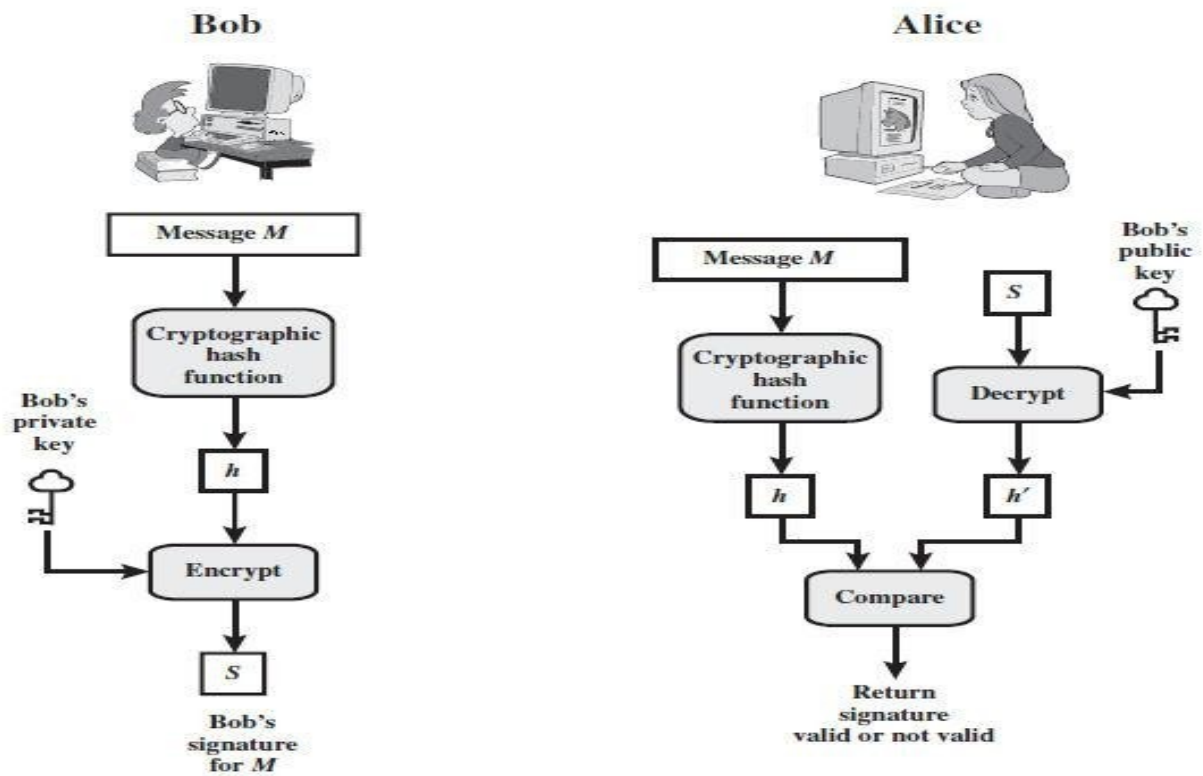


Figure 4.5 Simplified Depiction of Essential Elements of Digital Signature Process

Obtaining a User's Certificate:

User certificates generated by a CA have the following characteristics:

Any user with access to the public key of the CA can verify the user public key that was certified.

No party other than the certification authority can modify the certificate without this being detected.

If all users subscribe to the same CA, then there is a common trust of that CA.

All user certificates can be placed in the directory for access by all users.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA.

If the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key.

- A obtains (from the directory) the certificate of X_2 signed by X_1 . Because A securely knows X_1 's public key, A can obtain X_2 's public key from its certificate and verify it by means of X_1 's signature on the certificate.
- A then goes back to the directory and obtains the certificate of B signed by X_2 . Because A now has a trusted copy of X_2 's public key, A can verify the signature and securely obtain B's public key.
- ▶ A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1 \langle\langle X_2 \rangle\rangle X_2 \langle\langle B \rangle\rangle$$

- ▶ In the same fashion, B can obtain A's public key with the reverse chain:

$$X_2 \langle\langle X_1 \rangle\rangle X_1 \langle\langle A \rangle\rangle$$

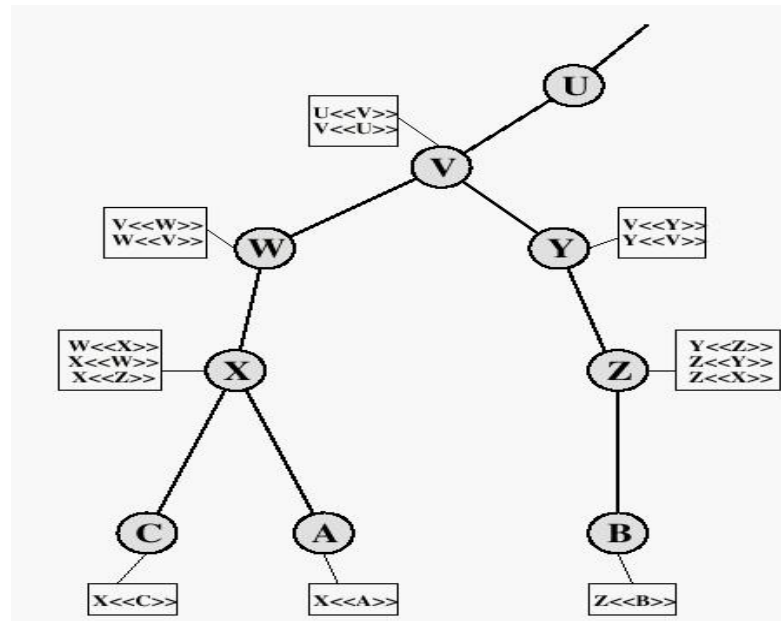
- ▶ This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$$X_1 \langle\langle X_2 \rangle\rangle X_2 \langle\langle X_3 \rangle\rangle \dots X_N \langle\langle B \rangle\rangle$$

All of these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate.

X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Example of a hierarchy:



X.509 CA Hierarchy

The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- ▮ **Forward certificates:** Certificates of X generated by other CAs
- ▮ **Reverse certificates:** Certificates generated by X that are the certificates of other CAs.

In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

$X \ll W \gg W \ll V \gg V \ll Y \gg Y \ll Z \gg Z \ll B \gg$

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key.

Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the certification path:

$Z \ll Y \gg Y \ll V \gg V \ll W \gg W \ll X \gg X \ll A \gg$

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

Revocation of Certificates:

Reasons for revocation:

- ▮ The user's secret key is assumed to be compromised.
- ▮ The user is no longer certified by this CA.

The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both issued to users and to other CAs. These lists should be posted on the directory.

Each CRL posted to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is schedule to be issued, and an entry for each revoked certificate.

Each entry consists of the serial number of a certificate and revocation date for that certificate.

When user receives a certificate in a message, the user must determine whether the certificate has been revoked.

The user could check the directory each time a certificate is received.

To avoid the delays associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certification.

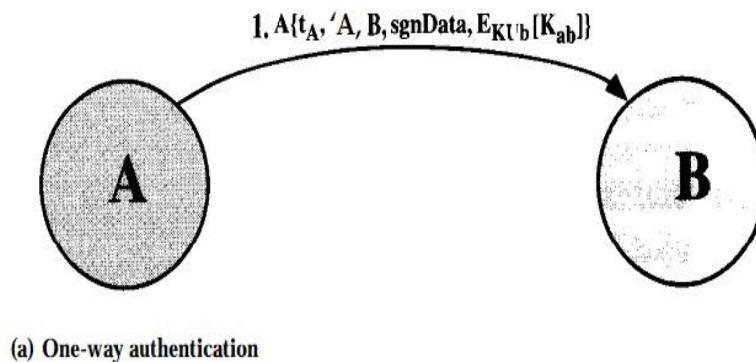
Authentication Procedures:

X.509 supports three authentication procedures.

- One -way authentication
- Two-way authentication
- Three way authentication

All these procedures make use of public-key signatures.

One -way authentication:



One-way authentication involves a single transfer of information from one user (A) to another (B) and establishes the following:

The identity of A and that the message was generated by A .

That the message was intended for B

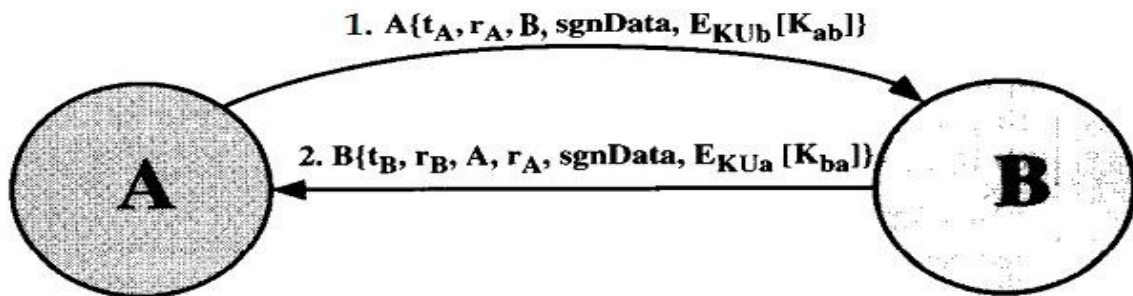
3 . The integrity and originality (it has not been sent multiple times) of the message

Note that only the identity of the initiating entity is verified in this process, not that of the responding entity.

At a minimum, the message includes a timestamp t_A , a nonce r_A , and the identity of B and is signed with A's public key. The timestamp consists of an optional generation time and an expiration time. This prevents delayed delivery of messages. The nonce can be used to detect replay attacks. The nonce value must be unique within the expiration time of the message. Thus, B can store the nonce until it expires and reject any new messages with the same nonce.

For pure authentication, the message is used simply to present credentials to B. The message may also include information to be conveyed. This information, $sgnData$, is included within the scope of the signature, guaranteeing its authenticity and integrity. The message may also be used to convey a session key to B, encrypted with B's public key.

Two-way authentication:



(h) Two-way authentication

In addition to the three elements just listed, two-way authentication establishes the following elements:

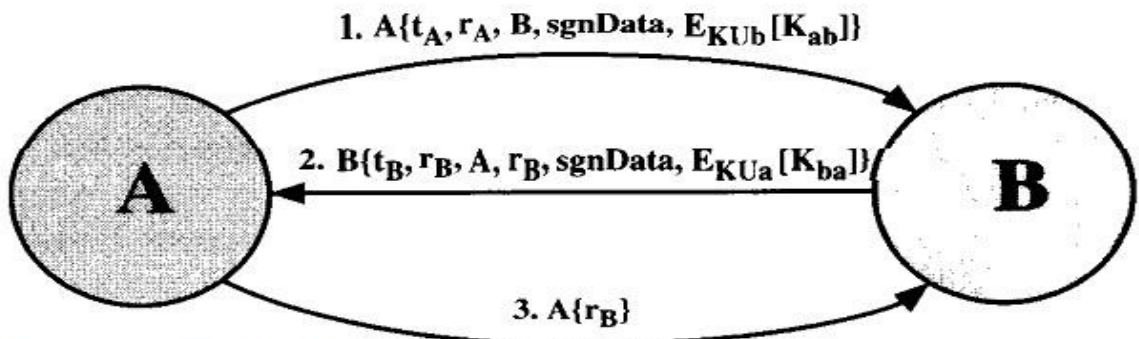
The identity of B and that the reply message was generated by B

That the message was intended for A

The integrity and originality of the reply

Two-way authentication thus permits both parties in a communication to verify the identity of the other. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B. As before, the message may include signed additional information and a session key encrypted with A's public key.

Three way authentication:



(c) Three-way authentication

In three-way authentication, a final message from A to B is included, which contains a signed copy of the nonce r_B . The intent of this design is that timestamps need not be

checked: Because both nonces are echoed back by the other side, each side can check the returned nonce to detect replay attacks. This approach is needed when synchronized clocks are not available.

X.509 Version 3:

lists the following requirements not satisfied by version 2:

- The Subject field is inadequate to convey the identity of a key owner to a public key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.
- The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
- There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.
- There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
- It is important to be able to identify different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Thus, version 3 includes a number of optional extensions that may be added to the version 2 format.

Each extension consists of an extension identifier, a criticality indicator, and an extension value.

The criticality indicator indicates whether an extension can be safely ignored.

If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: **key and policy information**, **subject and issuer attributes**, and **certification path constraints**.

Key and Policy Information:

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.

A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.

This area includes:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.

Subject key identifier: Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).

Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, and CA signature verification on CRLs.

Private-key usage period: Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys,

the usage period for the signing private key is typically shorter than that for the verifying public key.

Certificate policies: Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.

Policy mappings: Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate subject and issuer attributes:

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject to increase a certificate user's confidence that the certificate subject is a particular person or entity.

The extension fields in this area include:

Subject alternative name: Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPsec, which may employ their own name forms.

Issuer alternative name: Contains one or more alternative names, using any of a variety of forms.

Subject directory attributes: Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification path constraints:

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include:

Basic constraints: Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.

Name constraints: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.

Policy constraints: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

UNIT-IV: ELECTRONIC MAIL SECURITY

Email privacy: PGP operations, Radix-64 Conversion, Key Management for PGP, PGP Trust Model, Multipurpose Internet Mail Extension (MIME), Secure MIME (S-MIME).

X-----

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.

4.1 Pretty Good Privacy:

Philip R. Zimmerman is the creator of PGP.

PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications.

Zimmermann has done the following:

- ▢ Selected the best available cryptographic algorithms as building blocks.
- ▢ 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.
- ▢ 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).
- ▢ Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

Why Is PGP Popular?

It is available free on a variety of platforms.

Based on well known algorithms.

Wide range of applicability.

Not developed or controlled by governmental or standards organizations

Notation:

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 format¹

Operational Description:

The actual operation of PGP, Consist of five services:

- Authentication
- Confidentiality
- Compression
- E-mail compatibility
- Segmentation

Authentication:

The sender creates a message.

SHA-1 is used to generate a 160-bit hash code of the message.

The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.

The receiver uses RSA with the sender's public key to decrypt and recover the hash code.

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.

Confidentiality

The sender generates a message and a random 128-bit number to be used as a session key for this message only.

The message is encrypted using CAST-128 (or IDEA or 3DES) with the session key.

The session key is encrypted with RSA using the recipient's public key and is prepended to the message.

The receiver uses RSA with its private key to decrypt and recover the session key.

The session key is used to decrypt the message.

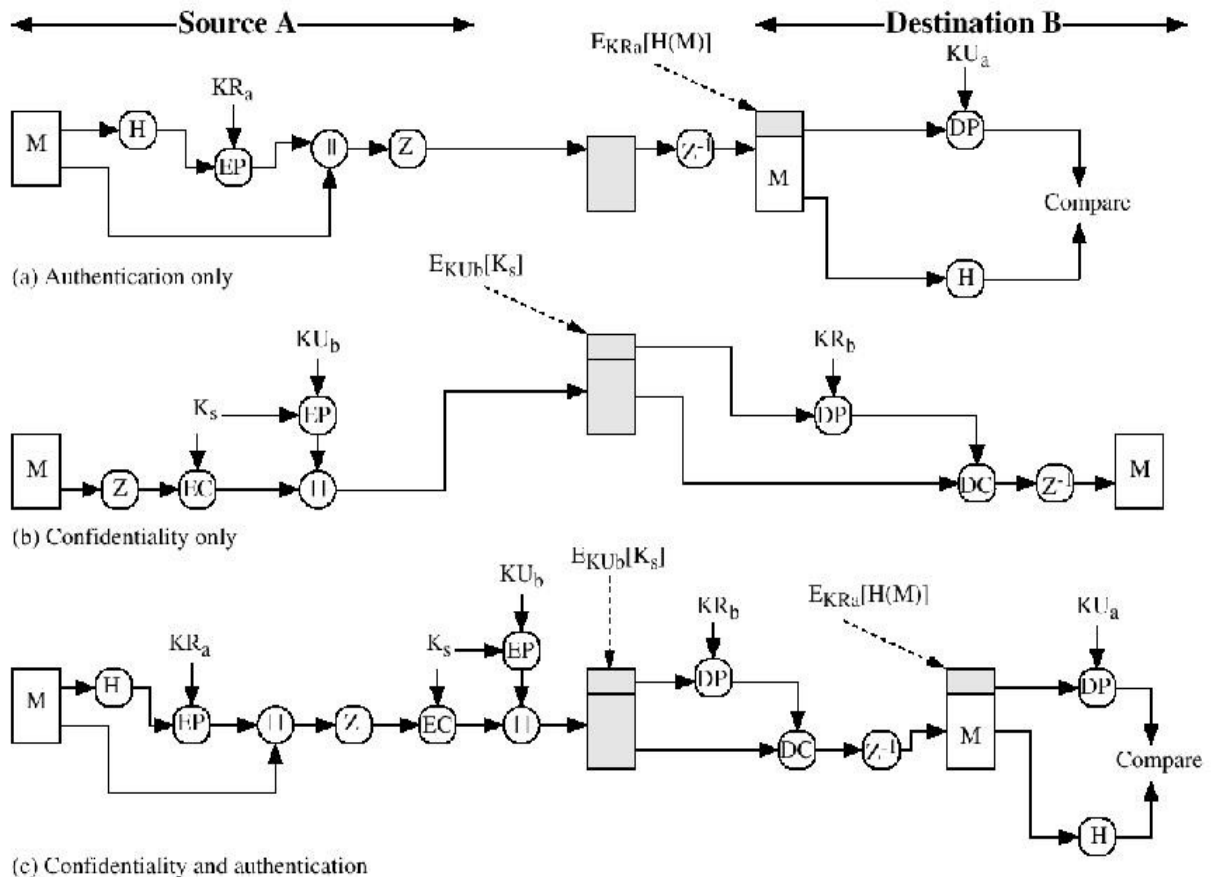


Figure 5.1 PGP Cryptographic Functions

Compression:

PGP compresses the message after applying the signature but before encryption

The placement of the compression algorithm is critical.

The compression algorithm used is ZIP

The signature is generated before compression for two reasons:

- It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification.
- Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext

E-mail Compatibility:

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 is radix-64 conversion

The use of radix-64 expands the message by 33%.

Segmentation and Reassembly:

Often restricted to a maximum message length of 50,000 octets.

Longer messages must be broken up into segments.

PGP automatically subdivides a message that is too large.

The receiver strips off all e-mail headers and reassembles the block.

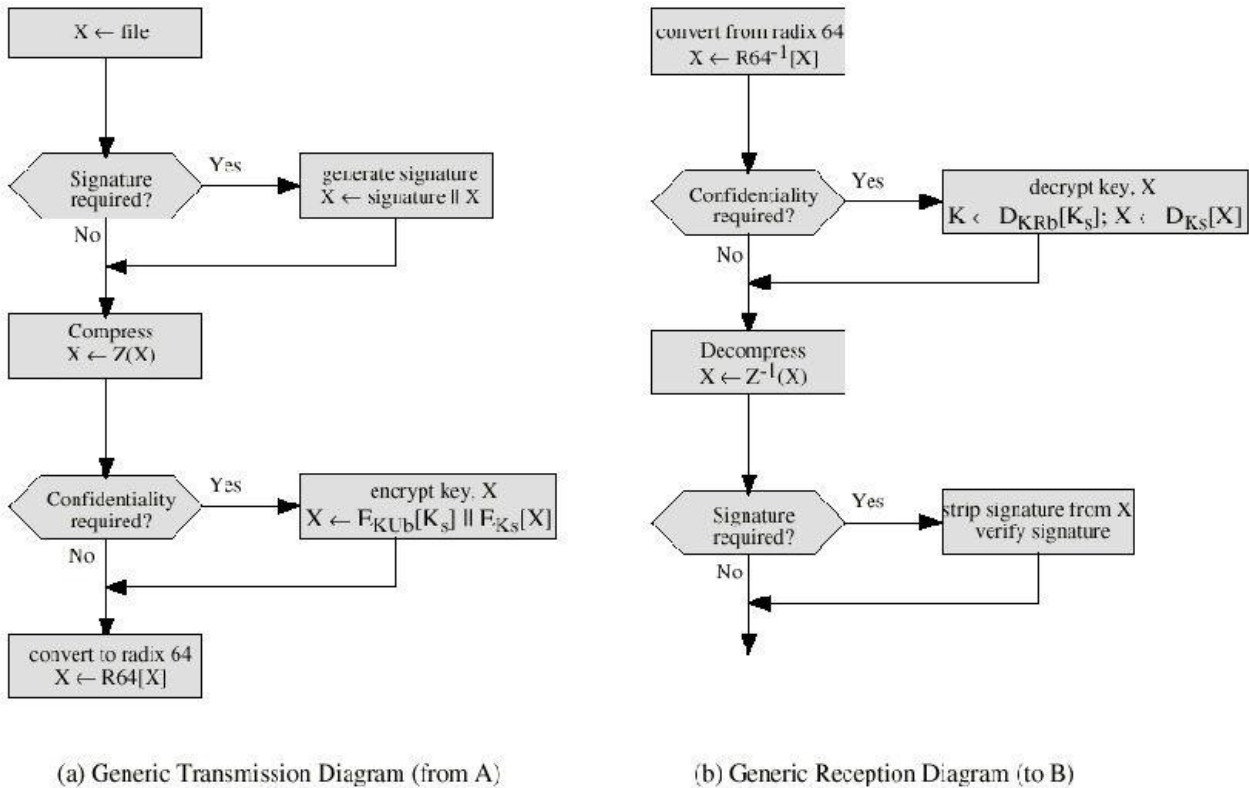


Figure 5.2 Transmission and Reception of PGP Messages

Table 7.1 Summary of PGP Services

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed for storage or transmission using ZIP.
E-mail compatibility	Radix-64 conversion	To provide transparency for e-mail applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.

4.2 Radix-64 Conversion:

Both PGP and S/MIME make use of an encoding technique.

This technique maps arbitrary binary input into printable character output.

The form of encoding has the following relevant characteristics:

The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set.

The character set consists of 65 printable characters, one of which is used for padding. With $2^6=64$ available characters, each character can be used to represent 6 bits of input.

No control characters are included in the set.

The hyphen character "-" is not used. This character has significance in the RFC 5322 format and should therefore be avoided.

Table 7.9 Radix-64 Encoding

6-bit Value	Character Encoding	6-bit Value	Character Encoding	6-bit Value	Character Encoding	6-bit Value	Character Encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

Table shows the mapping of 6-bit input values to characters.

The character set consists of the alphanumeric characters plus "+" and "/". The "=" character is used as the padding character.

Binary input is processed in blocks of 3 octets (24 bits).

Each set of 6 bits in the 24-bit block is mapped into a character.

In the figure, the characters are shown encoded as 8-bit quantities.

In this typical case, each 24-bit input is expanded to 32 bits of output.

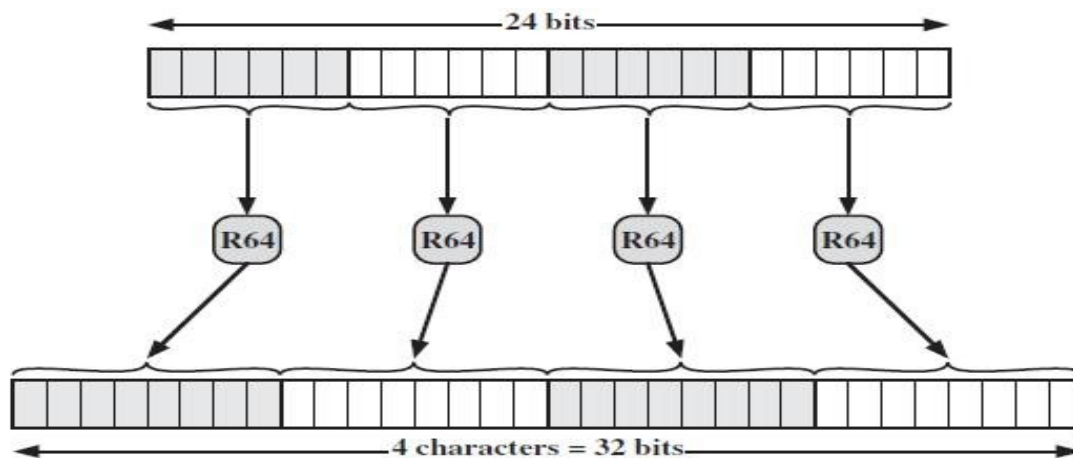


Figure 7.12 Printable Encoding of Binary Data into Radix-64 Format

For example,

- consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:
- 001000 110101 110010 010001
- Looking these up in Table yields the radix-64 encoding as the following characters: I1yR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

01001001 00110001 01111001 01010010

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	I1yR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

4.3 Key Management for PGP:

Cryptographic Keys and Key Rings:

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys (explained subsequently).

Three separate requirements can be identified with respect to these keys.

- User wish to change their pair of keys time to time .
- User may need to correspond with different groups of people.
- Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

Session key generation:

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message.

CAST-128 and IDEA use 128-bit keys; 3DES uses a 168-bit key. For the following discussion, we assume CAST-128.

The “plaintext” input to the random number generator, consisting of two 64-bit blocks, is itself derived from a stream of 128-bit randomized numbers.

These numbers are based on keystroke input from the user.

Both the keystroke timing and the actual keys struck are used to generate the randomized stream.

Thus, if the user hits arbitrary keys at his or her normal pace, a reasonably “random” input will

be generated.

This random input is also combined with previous session key output from CAST-128 to form the key input to the generator.

The result, given the effective scrambling of CAST-128, is to produce a sequence of session keys that is effectively unpredictable.

Key identifiers:

How, then, does the recipient know which of its public keys was used to encrypt the session key?

One simple solution would be to transmit the public key with the message. The recipient could then verify that this is indeed one of its public keys, and proceed.

This scheme would work, but it is unnecessarily wasteful of space.

Another solution would be to associate an identifier with each public key that is unique at least within one user. That is, the combination of user ID and key ID would be sufficient to identify a key uniquely.

Then only the much shorter key ID would need to be transmitted.

This solution, raises a management and overhead problem: Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key.

This seems unnecessarily burdensome.

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID.

The key ID associated with each public key consists of its least significant 64 bits.

That is, the key ID of public key is Pua is $(PUa \bmod 2^{64})$.

This is a sufficient length that the probability of duplicate key IDs is very small.

A key ID is also required for the PGP digital signature.

Detailed look at the format of a transmitted message.

A message consists of three components: the message component, a signature (optional), and a session key component (optional).

The message component includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation.

The signature component includes the following.

Timestamp: The time at which the signature was made.

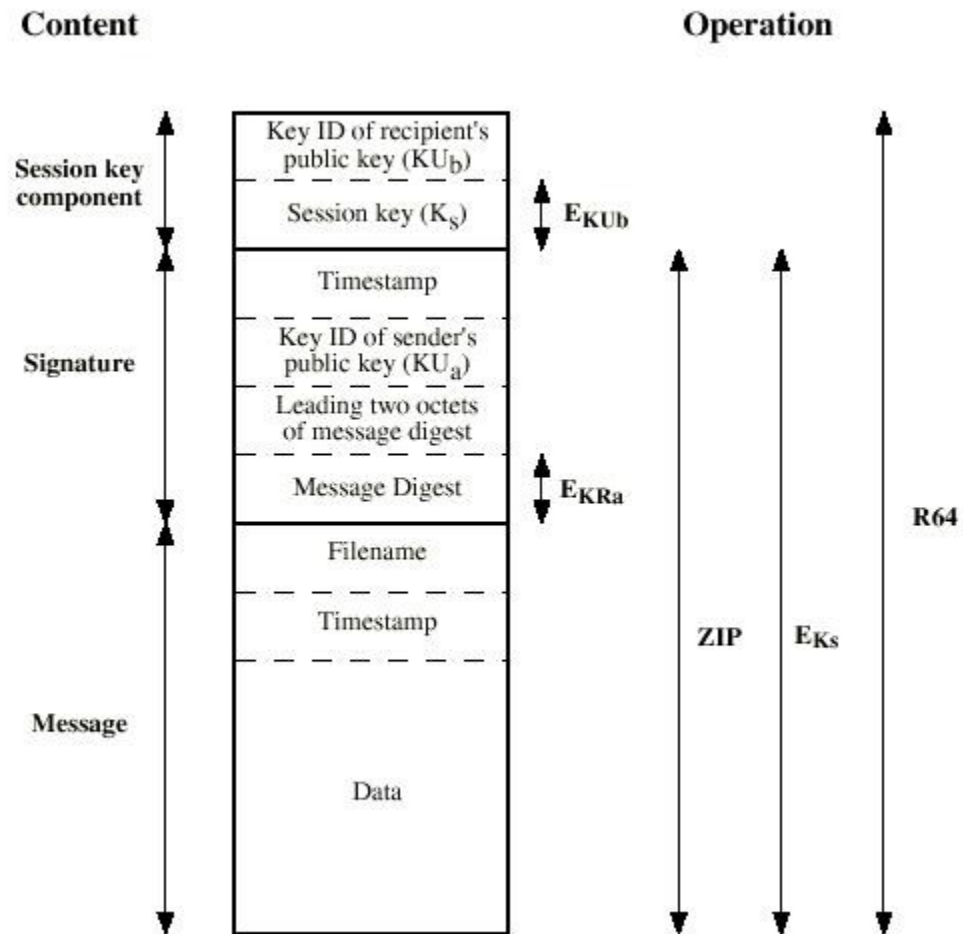
Message digest: The 160-bit SHA-1 digest encrypted with the sender's private signature key.

The digest is calculated over the signature timestamp concatenated with the data portion of the message component.

The inclusion of the signature timestamp in the digest insures against replay types of attacks.

The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message.

Format of PGP Message:



Leading two octets of message digest:

Enables the recipient to determine if the correct public key was used to decrypt the message digest for authentication by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest.

These octets also serve as a 16-bit frame check sequence for the message.

Key ID of sender's public key:

Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key.

The session key component includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.

The entire block is usually encoded with radix-64 encoding.

Key rings:

keys need to be stored and organized in a systematic way for efficient and effective use by all parties.

The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node.

These data structures are referred to, respectively, as the private-key ring and the public-key ring.

General structure of a private-key ring:

We can view the ring as a table in which each row represents one of the public/private key pairs owned by this user. Each row contains the entries:

Timestamp: The date/time when this key pair was generated.

Key ID: The least significant 64 bits of the public key for this entry.

Public key: The public-key portion of the pair.

Private key: The private-key portion of the pair; this field is encrypted.

User ID: Typically, this will be the user's e-mail address (e.g., stallings@acm.org).

However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same User ID more than once.

The private-key ring can be indexed by either User ID or Key ID.

The private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

- The user selects a passphrase to be used for encrypting private keys.
- When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.
- The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Private-Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
⋮	⋮	⋮	⋮	⋮
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
⋮	⋮	⋮	⋮	⋮

General structure of a public-key ring:

This data structure is used to store public keys of other users that are known to this user.

Timestamp: The date/time when this entry was generated.

Key ID: The least significant 64 bits of the public key for this entry.

Public Key: The public key for this entry.

User ID: Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

Public-Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T_i	$PU_i \bmod 2^{64}$	PU_i	trust_flag_i	User i	trust_flag_i		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

* = field used to index table

We are now in a position to show how these key rings are used in message transmission and reception. For simplicity, we ignore compression and radix-64 conversion in the following discussion. First consider message transmission (Figure below) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps.

Signing the message:

- a. PGP retrieves the sender's private key from the private-key ring using *your_userid* as an index. If *your_userid* was not provided in the command, the first private key on the ring is retrieved.

- b. PGP prompts the user for the passphrase to recover the unencrypted private key.
 - c. The signature component of the message is constructed.
- Encrypting the message:
- a. PGP generates a session key and encrypts the message.
 - b. PGP retrieves the recipient's public key from the public-key ring using *her_userid* as an index.
 - c. The session key component of the message is constructed.

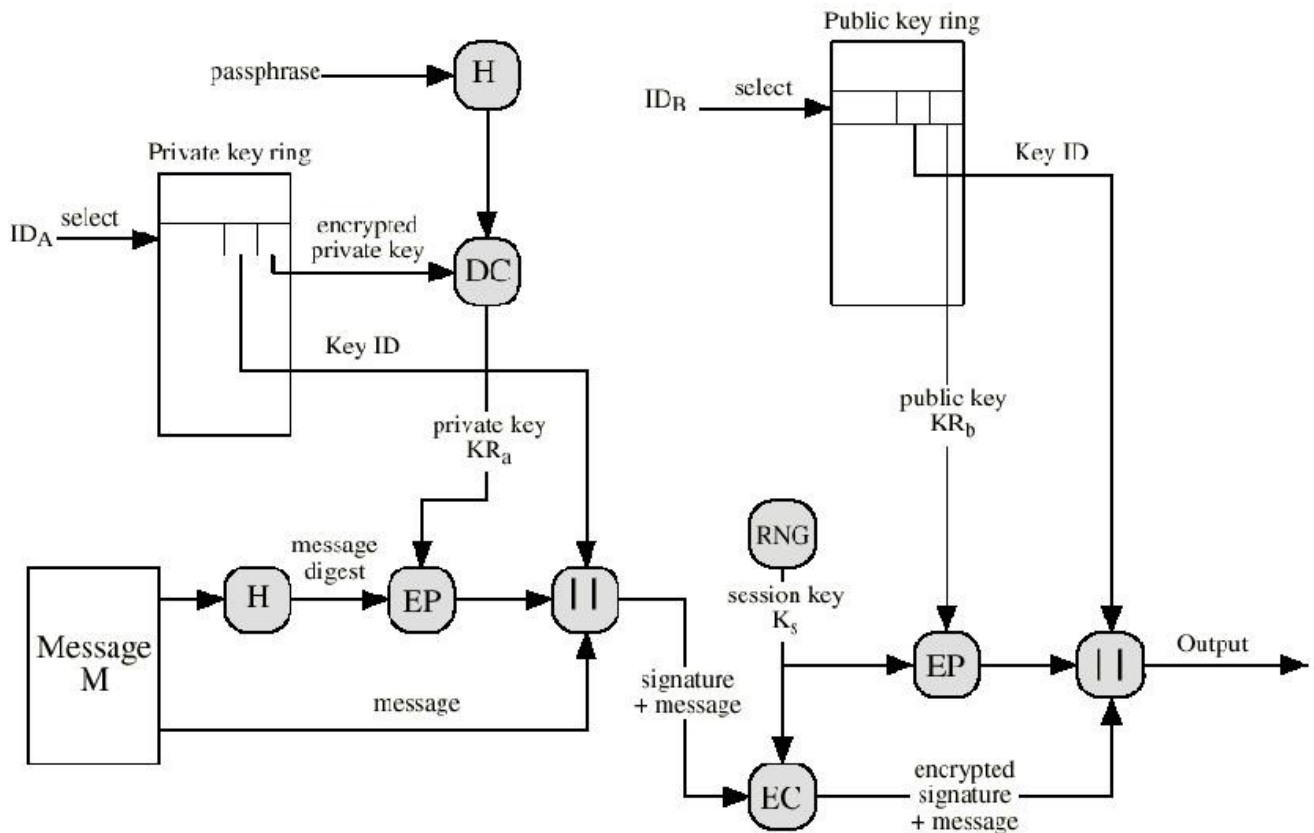


Figure 5.5 PGP Message Generation (from User A to User B; no compression or radix 64 conversion)

The receiving PGP entity performs the following steps (Figure below).:

1. Decrypting the message:
 - a. PGP retrieves the receiver's private key from the private-key ring using the Key ID field in the session key component of the message as an index.
 - b. PGP prompts the user for the passphrase to recover the unencrypted private key.
 - c. PGP then recovers the session key and decrypts the message.
2. Authenticating the message:

- PGP retrieves the sender's public key from the public-key ring using the Key ID field in the signature key component of the message as an index.
- PGP recovers the transmitted message digest.
- PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

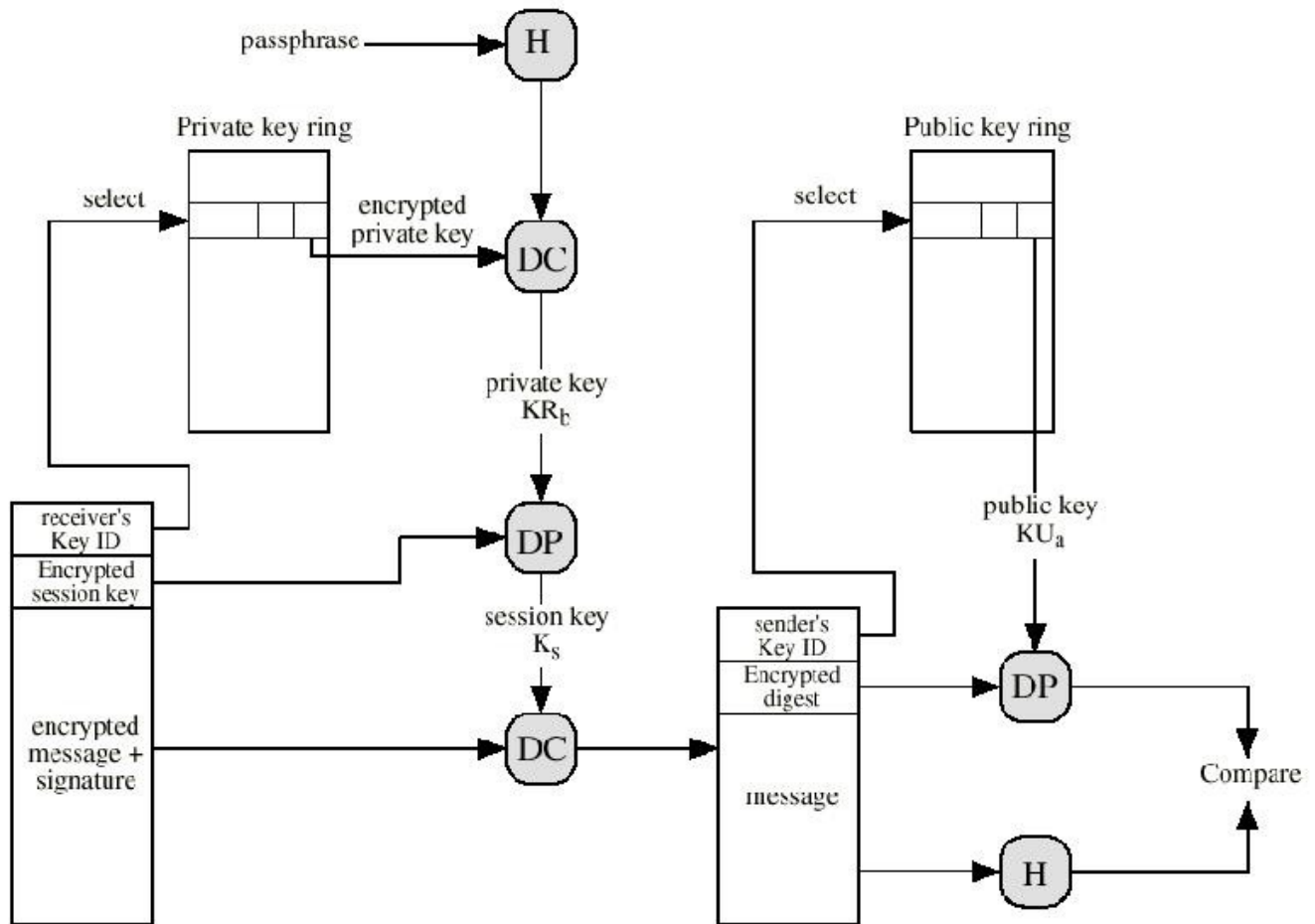


Figure 5.6 PGP Message Reception (from User A to User B; no compression or radix 64 conversion)

Public-Key Management:

PGP contains a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service.

To complete the system, one final area needs to be addressed, that of public-key management.

for use in a variety of formal and informal environments, no rigid public-key management scheme is set up.

Approaches to public-key management:

The essence of the problem is this: User A must build up a public-key ring containing the public keys of other users to interoperate with them using PGP.

Suppose that A's key ring contains a public key attributed to B, but in fact the key is owned by C.

This could happen, for example, if A got the key from a bulletin board system (BBS) that was used by B to post the public key but that has been compromised by C.

The result is that two threats now exist.

First, C can send messages to A and forge B's signature so that A will accept the message as coming from B. Second, any encrypted message from A to B can be read by C.

A number of approaches are possible for minimizing the risk that a user's public-key ring contains false public keys.

The following are some approaches that could be used.

- Physically get the key from B. B could store her public key on a floppy disk and hand it to A. A could then load the key into his system from the floppy disk. This is a very secure method but has obvious practical limitations.

Verify a key by telephone. If A can recognize B on the phone, A could call B and ask her to dictate the key, in radix-64 format, over the phone. As a more practical alternative, B could transmit her key in an e-mail message to A. A could have PGP generate a 160-bit SHA-1 digest of the key and display it in hexadecimal format; this is referred to as the "fingerprint" of the key. A could then call B and ask her to dictate the fingerprint over the phone. If the two fingerprints match, the key is verified.

Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key. D generates an SHA-1 digest of this certificate, encrypts it with her private key, and attaches the signature to the certificate. Because only D could have created the signature, no one else can create a false public key and pretend that it is signed by D. The signed certificate could be sent directly to A by B or D, or it could be posted on a bulletin board.

Obtain B's public key from a trusted certifying authority. Again, a public-key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.

4.4 Trust Model

Although PGP does not include any specification for establishing certifying authorities or for establishing trust, it does provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

The basic structure is as follows.

Key legitimacy field

- that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID to this key. This field is computed by PGP. Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate.

Signature trust field

- indicates the degree to which this PGP user trusts the signer to certify public keys. The key legitimacy field is derived from the collection of signature trust fields in the entry.

Owner trust field

- indicates the degree to which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user.

We can describe the operation of the trust processing as follows.

When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of ultimate trust is automatically assigned to the trust field.

Table 7.2 Contents of Trust Flag Byte

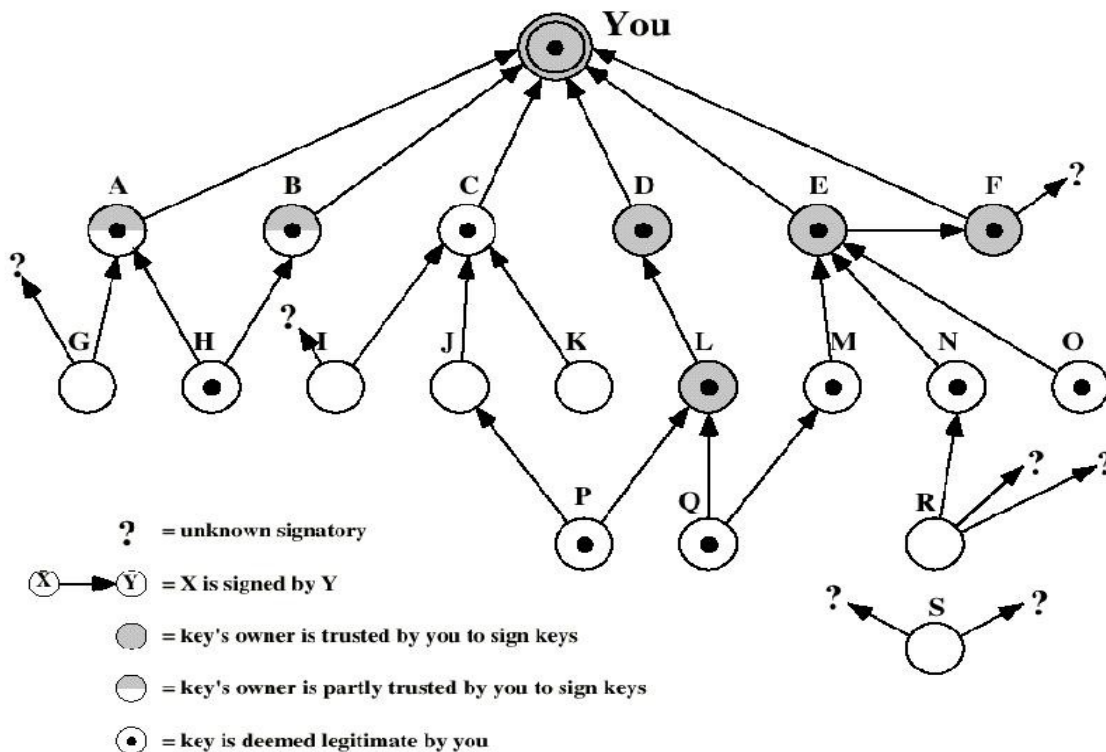
(a) Trust Assigned to Public-Key Owner (appears after key packet; user defined)	(b) Trust Assigned to Public Key/User ID Pair (appears after User ID packet; computed by PGP)	(c) Trust Assigned to Signature (appears after signature packet; cached copy of OWNERTRUST for this signator)
OWNERTRUST Field —undefined trust —unknown user —usually not trusted to sign other keys —usually trusted to sign other keys —always trusted to sign other keys —this key is present in secret key ring (ultimate trust)	KEYLEGIT Field —unknown or undefined trust —key ownership not trusted —marginal trust in key ownership —complete trust in key ownership	SIGTRUST Field —undefined trust —unknown user —usually not trusted to sign other keys —usually trusted to sign other keys —always trusted to sign other keys —this key is present in secret key ring (ultimate trust)
BUCKSTOP bit —set if this key appears in secret key ring	WARNONLY bit —set if user wants only to be warned when key that is not fully validated is used for encryption	CONTIG bit —set if signature leads up a contiguous trusted certification path back to the ultimately trusted key ring owner

Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify

that this owner is unknown, untrusted, marginally trusted, or completely trusted.

When the new public key is entered, one or more signatures may be attached to it. More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an *unknown user value* is assigned.

The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of *ultimate*, then the key legitimacy value is set to *complete*. Otherwise, PGP computes a weighted sum of the trust values. A weight of $1/X$ is given to signatures that are always trusted $1/Y$ and to signatures that are usually trusted, where X and Y are user-configurable parameters. When the total of weights of the introducers of a Key/UserID combination reaches 1, the binding is considered to be trustworthy, and the key legitimacy value is set to complete. Thus, in the absence of ultimate trust, at least X signatures that are always trusted, Y signatures that are usually trusted, or some combination is needed.



PGP Trust Model Example

Several points are illustrated

- Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L. Such a user signature is not always necessary, as the presence of node L indicates, but in practice, most users are likely to sign the keys for most owners that they trust. So, for example, even though E's key is already signed by trusted introducer F, the user chose to sign E's key directly.
- We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.
- A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate.
- This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.
- Figure above also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

Revoking Public Keys:

The owner issues a key revocation certificate.

Normal signature certificate with a revoke indicator.

Corresponding private key is used to sign the certificate.

4.5 S/MIME.

Simple Mail Transfer Protocol (SMTP, RFC 822):

Defines a format for text messages to be sent using e-mail.

Internet standard.

Structure of RFC 822 compliant messages.

- ▢ **header lines (e.g., from: ..., to: ..., cc: ...)**
- ▢ blank line
- ▢ body (the text to be sent)

Example

```
Date: Tue, 16 Jan 1998 10:37:17 (EST)
From: "Levente Buttyan" <buttyan@hit.bme.hu>
Subject: Test
To: afriend@otherhost.bme.hu
Blablabla
```

◦

SMTP Limitations - Can not transmit, or has a problem with:

- ▢ executable files, or other binary files (jpeg image)
- ▢ "national language" characters (non-ASCII)
- ▢ messages over a certain size
- ▢ ASCII to EBCDIC translation problems
- ▢ lines longer than a certain length (72 to 254 characters)

some servers

- ▢ reject messages over a certain size delete, add, or reorder CR and LF characters truncate or wrap lines longer than 76 characters remove trailing white space (tabs and spaces) pad lines in a message to the same length convert tab characters into multiple spaces

Multipurpose Internet Mail Extension (MIME):

Defines new message header fields.

Defines a number of content formats (standardizing representation of multimedia contents).

Defines transfer encodings that protects the content from alteration by the mail system.

Header fields in MIME:

MIME-Version: Must be "1.0" -> RFC 2045, RFC 2046

Content-Type: More types being added by developers (application/word)

- describes the data contained in the body
- receiving agent can pick an appropriate method to represent the content

Content-Transfer-Encoding: How message has been encoded (radix-64)

Content-ID: Unique identifying character string.

Content Description:

- description of the object in the body of the message
- useful when content is not readable (e.g., audio data)

Table 7.3 MIME Content Types

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript format.
	octet-stream	General binary data consisting of 8-bit bytes.

Table 7.4 MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present, but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

MIME – Example:

MIME-Version: 1.0

From: Nathaniel Borenstein <nsb@nsb.fv.com>

To: Ned Freed <ned@innosoft.com>

Date: Fri, 07 Oct 1994 16:15:05 -0700 (PDT)

Subject: A multipart example

Content-Type: multipart/mixed; boundary=unique-boundary-1

This is the preamble area of a multipart message. Mail readers that understand multipart format

should ignore this preamble. If you are reading this text, you might want to consider changing to a mail reader

that understands how to properly display multipart messages.

--unique-boundary-1

Content-type: text/plain; charset=US-ASCII

... Some text ...

--unique-boundary-1

Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2

Content-Type: audio/basic

Content-Transfer-Encoding: base64

... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here ...

```
--unique-boundary-2
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
... base64-encoded image data goes here ...
--unique-boundary-2--
--unique-boundary-1
Content-type: text/enriched
This is <bold><italic>enriched.</italic></bold><smaller>as defined in
RFC 1896</smaller>
Isn't it <bigger><bigger>cool?</bigger></bigger>
--unique-boundary-1
Content-Type: message/rfc822
From: (mailbox in US-ASCII) To: (address in
US-ASCII) Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-
printable ... Additional text in ISO-8859-1 goes
here ... --unique-boundary-1—
```

S/MIME:

Secure/Multipurpose Internet Mail Extension
S/MIME will probably emerge as the industry standard.
PGP for personal e-mail security.
A security enhancement to MIME.
Provides similar services to PGP.
Based on technology from RSA Security.
Industry standard for commercial and organizational use.
RFC 2630, 2632, 2633.

S/MIME Functions:

Enveloped Data: Encrypted content and encrypted session keys for recipients.
Signed Data: Message Digest encrypted with private key of "signer."
Clear-Signed Data: Signed but not encrypted.
Signed and Enveloped Data: Various orderings for encrypting and signing.

Algorithms Used:

Message Digesting: SHA-1 and MDS
Digital Signatures: DSS

Secret-Key Encryption: Triple-DES, RC2/40 (exportable)

Public-Private Key Encryption: RSA with key sizes of 512 and 1024 bits, and Diffie-Hellman (for session keys).

S/MIME Messages:

S/MIME secures a MIME entity with a signature, encryption, or both.

A MIME entity may be an entire message or one or more of the subparts of the message.

The MIME entity plus some security related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce a PKCS, which refers to a set of public-key cryptography specifications issued by RSA Laboratories.

A PKCS object is then treated as message content and wrapped in MIME.

Have a range of S/MIME content-types, as shown.

- enveloped data
- signed data
- clear-signed data
- registration request
- certificate only message

Table 7.7 S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity.
	pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

enveloped data (application/pkcs7-mime; smime-type = enveloped-data)

- standard digital envelop

signed data (application/pkcs7-mime; smime-type = signed-data)

- standard digital signature ("hash and sign")

- content + signature is encoded using base64 encoding
- clear-signed data (multipart/signed)
- standard digital signature
 - only the signature is encoded using base64
 - recipient without S/MIME capability can read the message but cannot verify the signature
- signed and enveloped data
- signed and encrypted entities may be nested in any order

Registration Request

- Typically, an application or user will apply to a certification authority for a public-key certificate.
- The application/pkcs10 S/MIME entity is used to transfer a certification request.

Certificates-only Message

- A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request.
- The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate.

Registration Request

- Typically, an application or user will apply to a certification authority for a public-key certificate.
- The application/pkcs10 S/MIME entity is used to transfer a certification request.

Certificates-only Message

- A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request.
- The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate.

S/MIME Certificate Processing:

S/MIME uses public-key certificates that conform to version 3 of X.509.

The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust.

User Agent Role:

S/MIME user several key management functions to performs:

- **Key Generation** - Diffie-Hellman, DSS, and RSA key-pairs.
- **Registration** - Public keys must be registered with X.509 CA.
- **Certificate Storage** - Local (as in browser application) for different services.
- **Signed and Enveloped Data** - Various orderings for encrypting and signing.

Example: Verisign (www.verisign.com)

Class-1: Buyer's email address confirmed by emailing vital info.

Class-2: Postal address is confirmed as well, and data checked against directories.

Class-3: Buyer must appear in person, or send notarized documents.

UNIT-V: IP SECURITY ARCHITECTURE AND SERVICES

IP Security Overview, IP Security Architecture, Security Association, Authentication Header, Encapsulating Security Payload, Combining Security Associations and Key Management: OAKLEY key determination protocol, ISAKMP.

Introduction:

There are application-specific security mechanisms for a number of application areas, including electronic mail (S/MIME, PGP), client/server (Kerberos), Web access (Secure Sockets Layer), and others. However, users have security concerns that cut across protocol layers. For example, an enterprise can run a secure, private IP network by disallowing links to untrusted sites, encrypting packets that leave the premises, and authenticating packets that enter the premises. By implementing

security at the IP level, an organization can ensure secure networking not only for applications that have security mechanisms but also for the many security-ignorant applications.

IP-level security encompasses three functional areas: authentication, confidentiality, and key management.

5.0 IP SECURITY OVERVIEW: covers topics

- Applications of IPsec
- Benefits of IPsec
- Routing Applications
- IPsec Documents
- IPsec Services
- Transport and Tunnel Modes.

Applications of IPsec

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.

Figure 8.1(below) 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 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.

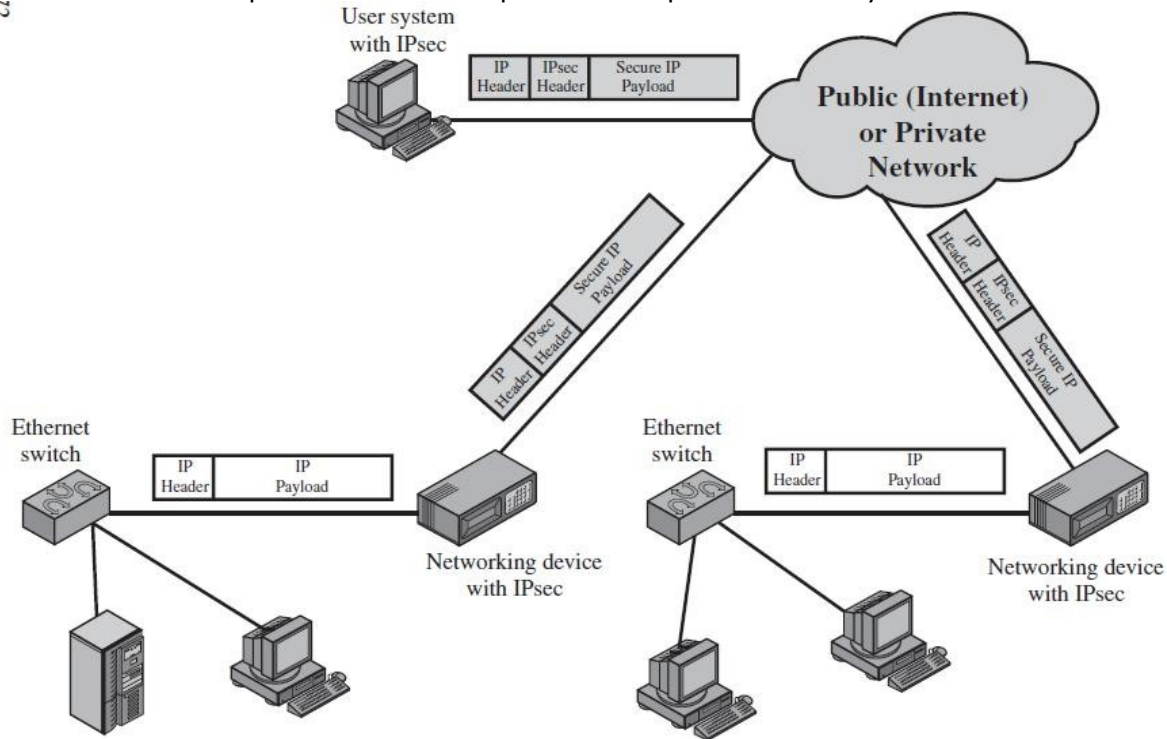


Figure 8.1 An IP Security Scenario

Benefits of IPsec:

- When IPsec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter.
- 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 subnet work within an organization for sensitive applications.

Routing Applications

In addition to supporting end users and protecting premises systems and networks, IPsec can play a vital role in the routing architecture required for internetworking. [HUIT98] lists the following examples of the use of IPsec. IPsec can assure that

- A router advertisement (a new router advertises its presence) comes from an authorized router.

A neighbor advertisement (a router seeks to establish or maintain a neighbor relationship with a router in another routing domain) comes from an authorized router.

A redirect message comes from the router to which the initial IP packet was sent.

A routing update is not forged.

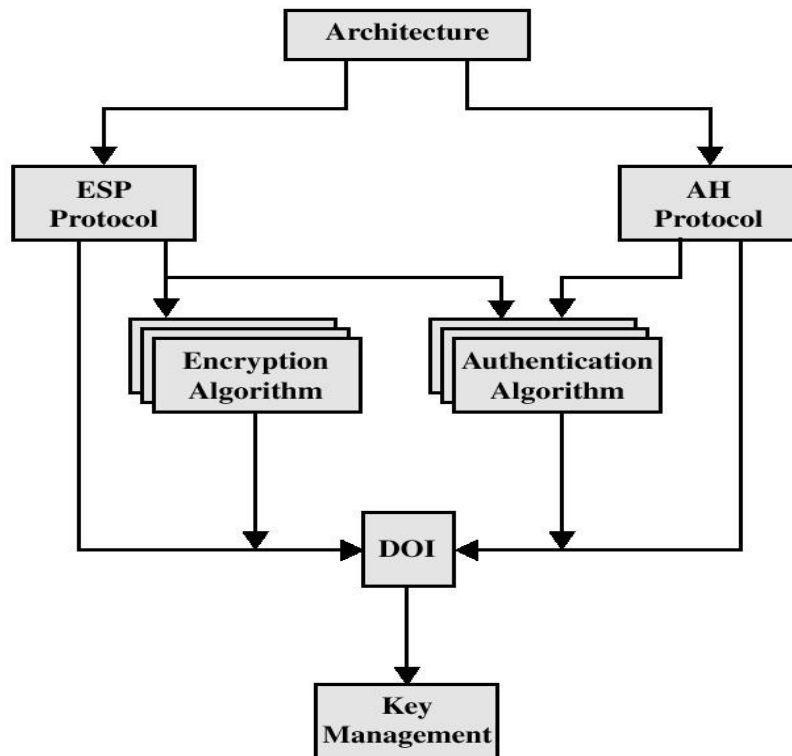
5.1 IP Security Architecture:

IPsec encompasses three functional areas: authentication, confidentiality, and key management. The totality of the IPsec specification is scattered across dozens of RFCs and draft IETF documents, making this the most complex and difficult to grasp of all IETF specifications. The best way to grasp the scope of IPsec is to consult the latest version of the IPsec document roadmap, which as of this writing is [FRAN09]. The documents can be categorized into the following groups

IPSec documents:

- RFC 2401: An overview of security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

IPSec Document Overview:



Architecture: covers the general concepts, security requirements, definitions, and mechanisms defining IPsec technology.

Key management: Documents that describe key management schemes.

Domain of Interpretation (DOI): contains values needed for the other documents to relate to each other.

These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key life time.

IPSec Services

IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services.

Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined

encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP).

Transport and Tunnel Modes:

Both AH and ESP support two modes of use: Transport and tunnel mode.

Transport mode provides protection primarily for upper-layer protocols.

That is, transport mode protection extends to the payload of an IP packet.

Examples include a TCP or UDP segment or an ICMP packet.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header.

AH in transport mode authenticates the IP payload and selected portions of the IP header.

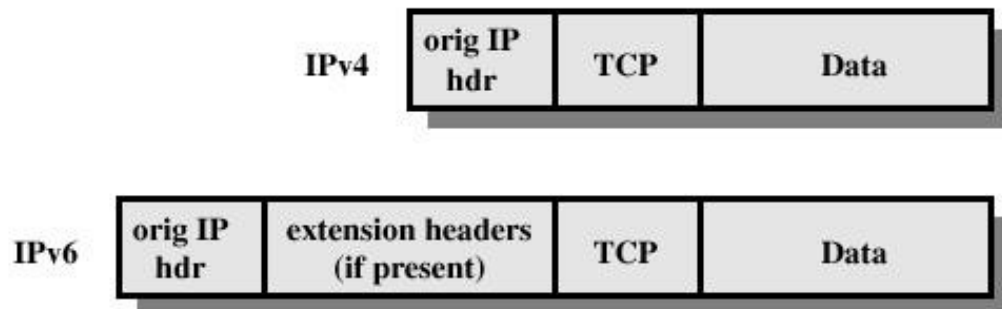
Tunnel mode provides protection to the entire IP packet.

To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new outer IP packet with a new outer IP header.

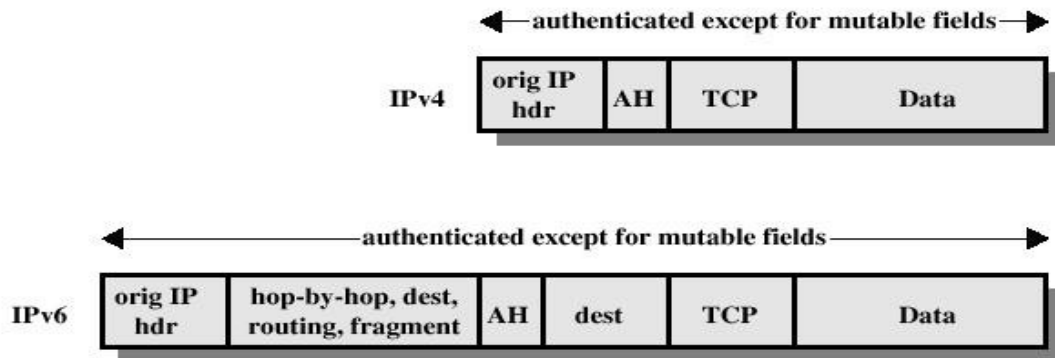
The entire original, inner, packet travels through a tunnel from one point of an IP network to another; no routers along the way are able to examine the inner IP header.

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers	Authenticates entire inner IP packet plus selected portions of outer IP header
ESP	Encrypts IP payload and any IPv6 extension header	Encrypts inner IP packet
ESP with authentication	Encrypts IP payload and any IPv6 extension header. Authenticates IP payload but no IP header	Encrypts inner IP packet. Authenticates inner IP packet.

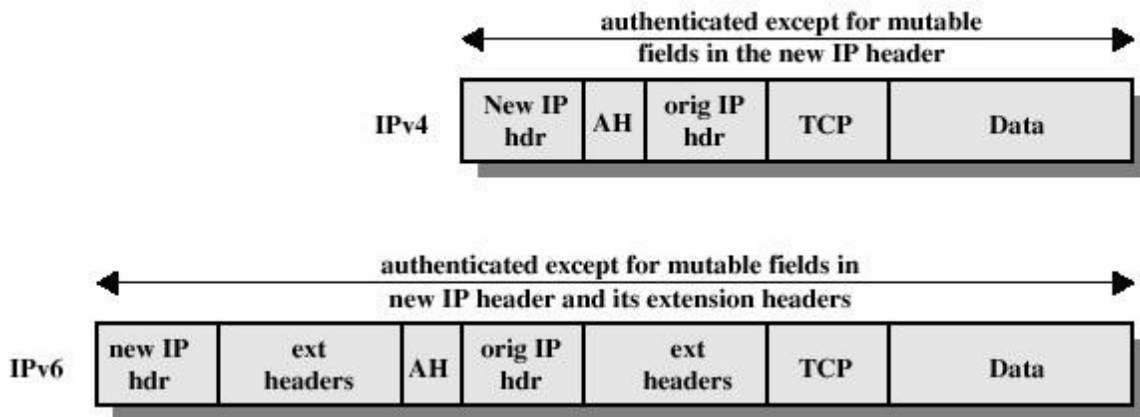
Before applying AH:



Transport Mode (AH Authentication):



Tunnel Mode (AH Authentication):



IP SECURITY POLICY

- Fundamental to the operation of IPsec is the concept of a security policy applied to each IP packet that transits from a source to a destination.
- IPsec policy is determined primarily by the interaction of two databases, the **security association database (SAD)** and the **security policy database (SPD)**.
- This section provides an overview of these two databases and then summarizes their use during IPsec operation.

2. Security Associations:

- A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA).
- An association is a one-way logical connection between a sender and a receiver that affords security services to the traffic carried on it.
- If a peer relationship is needed for two-way secure exchange, then two security associations are required.
- Security services are afforded to an SA for the use of AH or ESP, but not both.
- A security association is uniquely identified by three parameters.
- **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- **IP Destination Address:** This is the address of the destination endpoint of the SA, which may be an end-user system or a network system such as a firewall or router.

Security Protocol Identifier: This field from the outer IP header indicates whether the association is an AH or ESP security association.

Security Association Database:

In each IPsec implementation, there is a nominal Security Association Database that defines the parameters associated with each SA.

A security association is normally defined by the following parameters in an SAD entry.

Security Parameter Index: A 32-bit value selected by the receiving end of an SA

to uniquely identify the SA. In an SAD entry for an outbound SA, the SPI is used to construct the packet's AH or ESP header. In an SAD entry for an inbound SA, the SPI is used to map traffic to the appropriate SA.

Sequence Number Counter: A 32-bit value used to generate the Sequence Number field in AH or ESP headers•

Sequence Counter Overflow: A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).

Anti-Replay Window: Used to determine whether an inbound AH or ESP packet is a replay,

AH Information: Authentication algorithm, keys, key lifetimes, and related parameters being used with AH (required for AH implementations).

ESP Information: Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations).

Lifetime of this Security Association: A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations).

IPsec Protocol Mode: Tunnel, transport, or wildcard.

Path MTU: Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations).

Security Policy Database

SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic.

Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*.

these selectors are used to filter outgoing traffic in order to map it into a particular SA. Outbound processing obeys the following general sequence for each IP packet.

- Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.
- Determine the SA if any for this packet and its associated SPI.
- Do the required IPsec processing (i.e., AH or ESP processing).

The following selectors determine an SPD entry:

Remote IP Address: This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address.

Local IP Address: This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address.

Next Layer Protocol: The IP protocol header includes a field that designates the protocol operating over IP. If AH or ESP is used, then this IP protocol header immediately precedes the AH or ESP header in the packet.

Name: A user identifier from the operating system. This is not a field in the IP or upper-layer headers but is available if IPsec is running on the same operating system as the user.

Local and Remote Ports: These may be individual TCP or UDP port values, an enumerated list of ports, or a wildcard port.

5.3 ENCAPSULATING SECURITY PAYLOAD:

ESP can be used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and (limited) traffic flow confidentiality. The set of services provided depends on options selected at the time of Security Association (SA) establishment and on the location of the implementation in a network topology.

ESP can work with a variety of encryption and authentication algorithms, including authenticated encryption algorithms such as GCM.

ESP provides confidentiality services

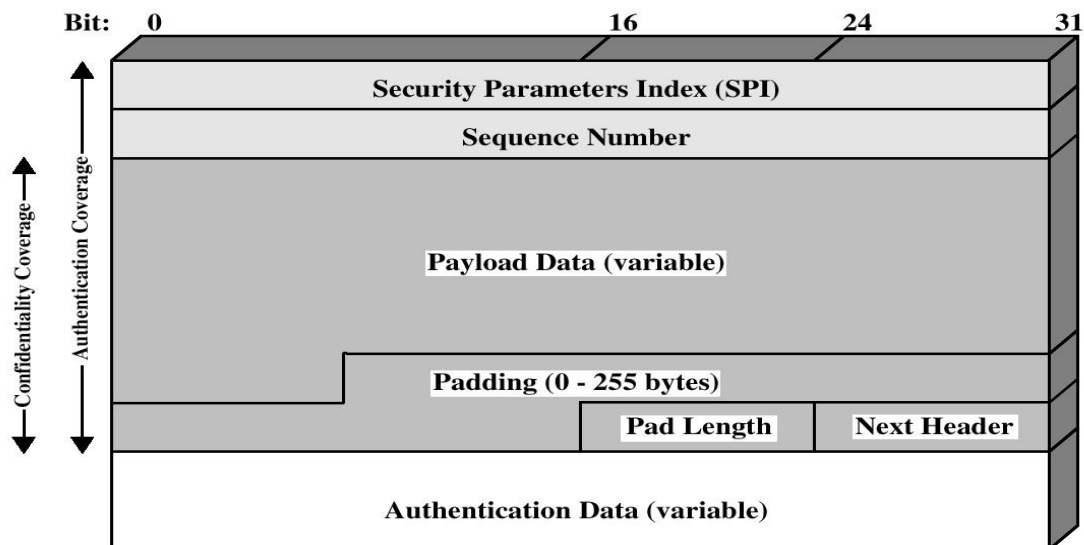


Figure 6.7 IPsec ESP Format

Security Parameters Index (32 bits): Identifies a security association.

Sequence Number (32 bits): A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.

Payload Data (variable): This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.

Padding (0-255 bytes): The purpose of this field is discussed later.

Pad Length (8 bits): Indicates the number of pad bytes immediately preceding this field.

Next Header (8 bits): Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).

The Padding field serves several purposes:

- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes (e.g., the multiple of a single block for a block cipher), the Padding field is used to expand the plaintext to the required length.

The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32-bit word. Equivalently, the ciphertext must be an integer multiple of 32 bits. The Padding field is used to assure this alignment.

Additional padding may be added to provide partial traffic-flow confidentiality by concealing the actual length of the payload.

5.4 Authentication Header:

Provides support for data integrity and authentication (MAC code) of IP packets. Guards against replay attacks.

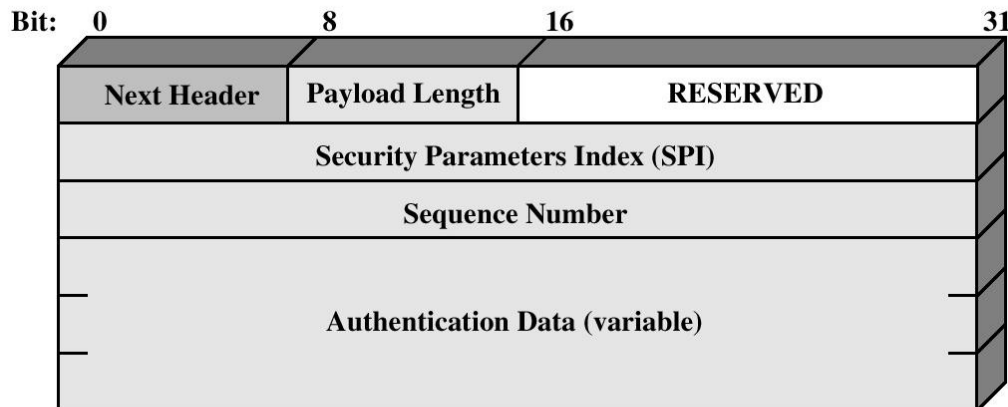


Figure 6.3 IPsec Authentication Header

Next Header (8 bits): Identifies the type of header immediately following this header.

Payload Length (8 bits): Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.

Reserved (16 bits): For future use.

Security Parameters Index (32 bits): Identifies a security association.

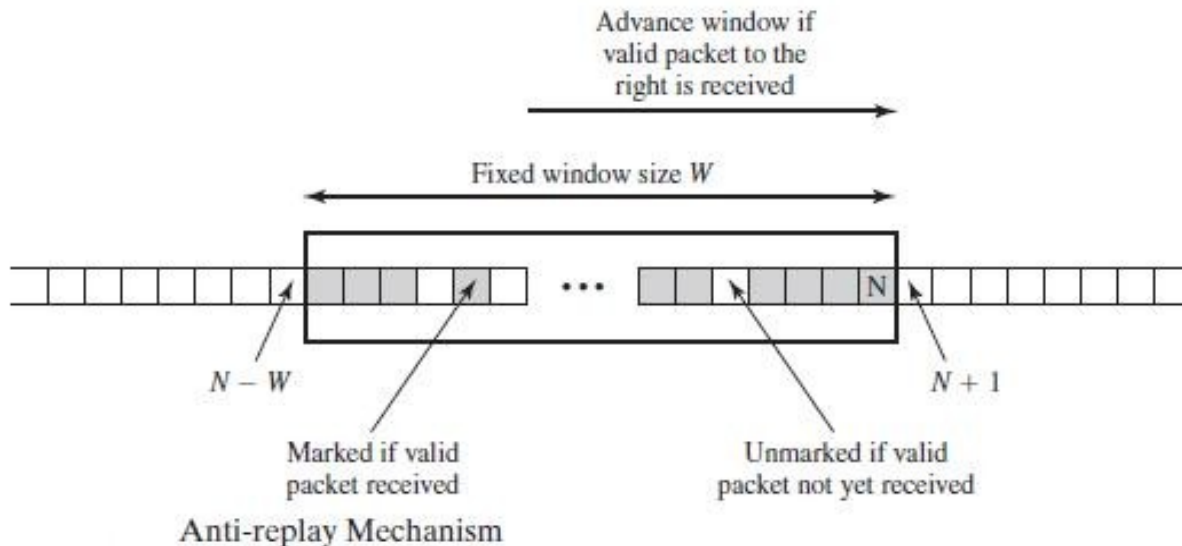
Sequence Number (32 bits): A monotonically increasing counter value, discussed later.

Authentication Data (variable): A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet, discussed later.

Anti-Replay Service:

A replay attack is one in which an attacker obtains a copy of an authenticated packet and later transmits it to the intended destination.

The receipt of duplicate, authenticated IP packets may disrupt service in some way or may have some other undesired consequence. The Sequence Number field is designed to thwart such attacks.



When a new SA is established, the sender initializes a sequence number counter to 1. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. If

anti-replay is enabled (the default), the sender must not allow the sequence number to cycle past $2^{32} - 1$ back to zero. Otherwise, there would be multiple valid packets with the same sequence number. If the limit of $2^{32} - 1$ is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPsec authentication document dictates that the receiver should implement a window of size W, with a default of $w=64$. The right edge of the window represents the highest sequence number, N, so far received for a valid packet. For any packet with a sequence number in the range from $N - W + 1$ to N that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked (Figure 8.6). Inbound processing proceeds as follows when a packet is received:

If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.

If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.

If the received packet is to the left of the window or if authentication fails, the packet is discarded; this is an auditable event.

Integrity Check Value:

- The Authentication Data field holds a value referred to as the Integrity Check Value.
- The ICV is a message authentication code or a truncated version of a code produced by a MAC algorithm.
 - 0 HMAC-MD5-96
 - 1 HMAC-SHA-1-96
- The MAC is calculated over the following:
 - 0 IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival are unpredictable are set to zero for purposes of calculation at both source and destination.

The AH header other than the Authentication Data field. The Authentication Data field is set to zero for purposes of calculation at both source and destination. The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).

Transport and Tunnel Modes:

TRANSPORT MODE ESP Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment), as shown in Figure 8.8b.

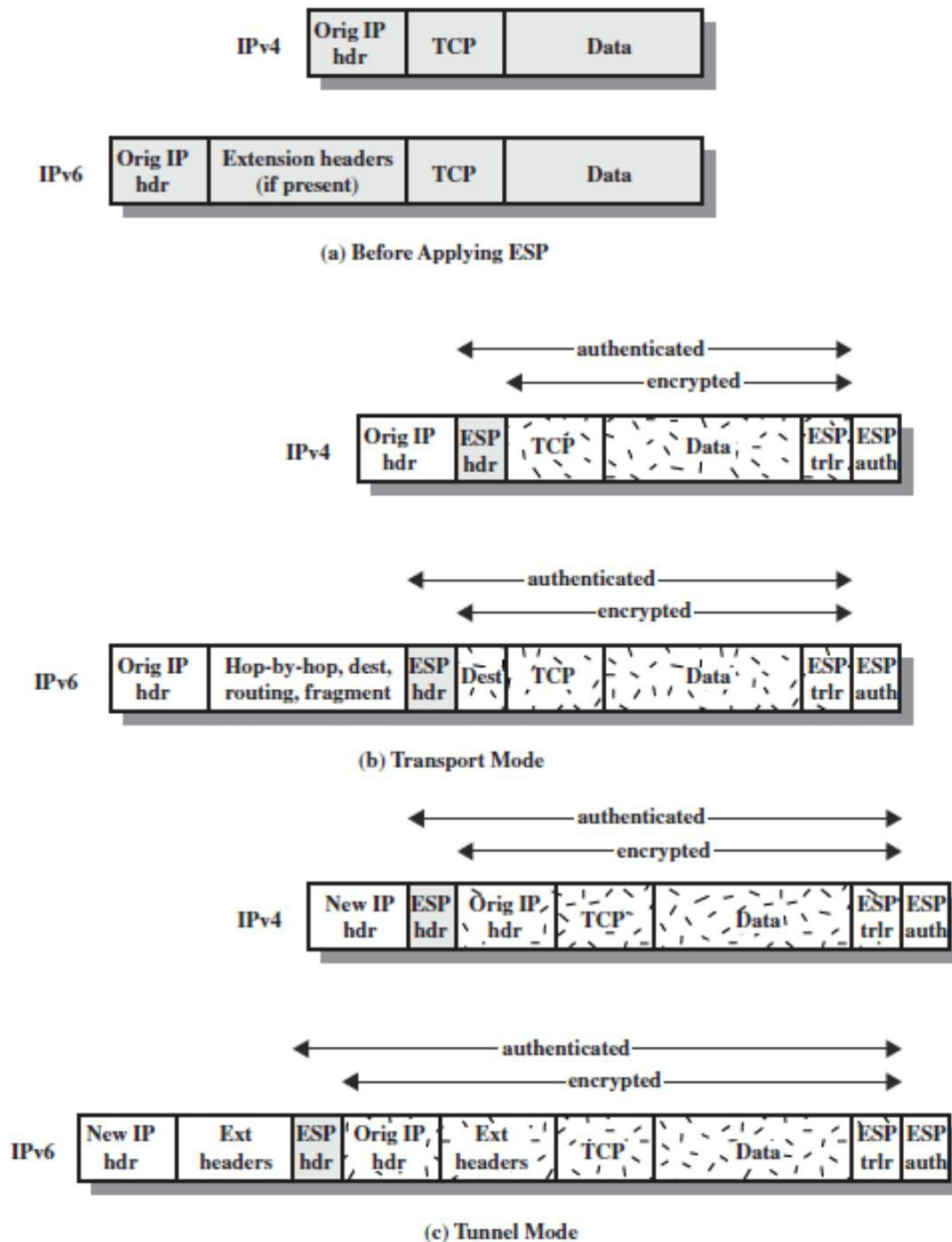


Figure 8.8 Scope of ESP Encryption and Authentication

Transport mode operation may be summarized as follows.

At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.

The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

TUNNEL MODE ESP Tunnel mode ESP is used to encrypt an entire IP packet (Figure 8.8c). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

The following steps occur for transfer of a transport-layer segment from the external host to the internal host.

- The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms the outer IP packet.

- The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus any outer IP extension headers but does not need to examine the ciphertext.

- The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet. This packet is then transmitted in the internal network.

- The inner packet is routed through zero or more routers in the internal network to the destination host.

Figure 8.9 shows the protocol architecture for the two modes.

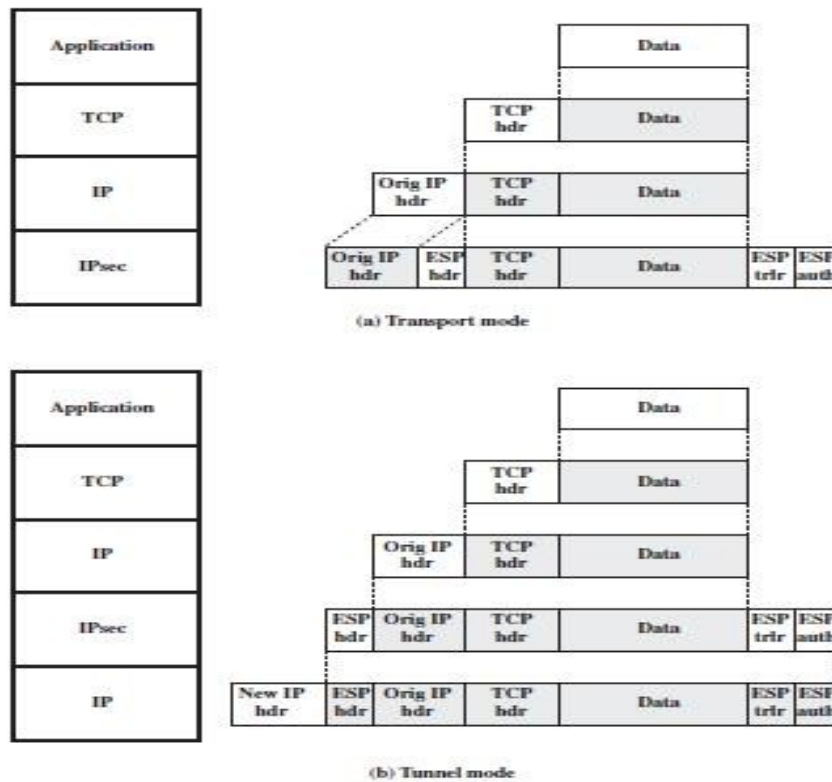


Figure 8.9 Protocol Operation for ESP

Encryption and Authentication Algorithms:

Encryption:

- Three-key triple DES
- RC5
- IDEA
- Three-key triple IDEA
- CAST
- Blowfish

Authentication:

- HMAC-MD5-96
- HMAC-SHA-1-96

5.5 COMBINING SECURITY ASSOCIATIONS:

An individual SA can implement either the AH or ESP protocol but not both.

Sometimes a particular traffic flow will call for the services provided by both AH and ESP.

a particular traffic flow may require IPsec services between hosts and, for that same flow, separate services between security gateways, such as firewalls.

In all of these cases, multiple SAs must be employed for the same traffic flow to achieve the desired IPsec services.

The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPsec services.

The SAs in a bundle may terminate at different endpoints or at the same endpoints.

Security associations may be combined into bundles in two ways:

Transport adjacency: Refers to applying more than one security protocol to the same IP packet without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination.

Iterated tunneling: Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. We look at several approaches.

ESP WITH AUTHENTICATION OPTION This approach is illustrated in Figure 8.8. In this approach, the user first applies ESP to the data to be protected and then appends the authentication data field. There are actually two subcases:

- **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.

- **Tunnel mode ESP:** Authentication applies to the entire IP packet delivered to the outer IP destination address (e.g., a firewall), and authentication is performed at that destination. The entire inner IP packet is protected by the privacy mechanism for delivery to the inner IP destination. For both cases, authentication applies to the ciphertext rather than the plaintext.

TRANSPORT ADJACENCY Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case, ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

TRANSPORT-TUNNEL BUNDLE The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for later reference. It is more convenient to do this if the authentication information applies to the unencrypted message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

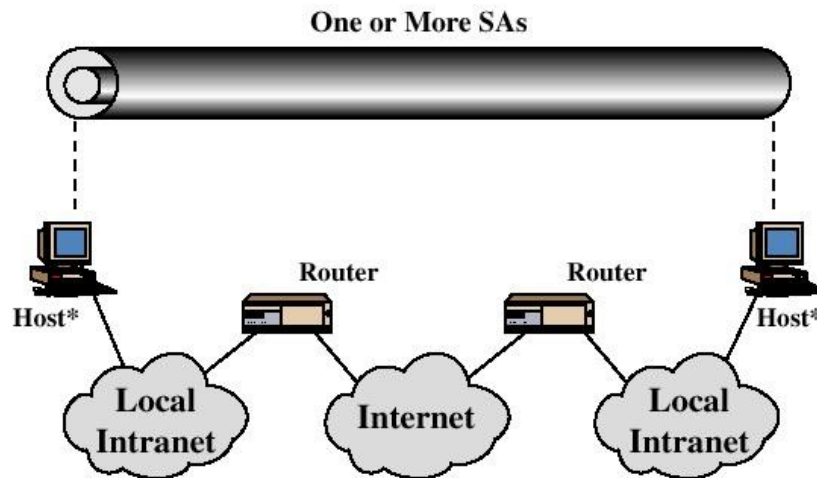
5.5.1 Basic Combinations of Security Associations

The IPsec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPsec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router).

Case 1. All security is provided between end systems that implement IPsec.

For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations are

- **a.** AH in transport mode
- **b.** ESP in transport mode
- **c.** ESP followed by AH in transport mode (an ESP SA inside an AH SA)
- **d.** Any one of a, b, or c inside an AH or ESP in tunnel mode

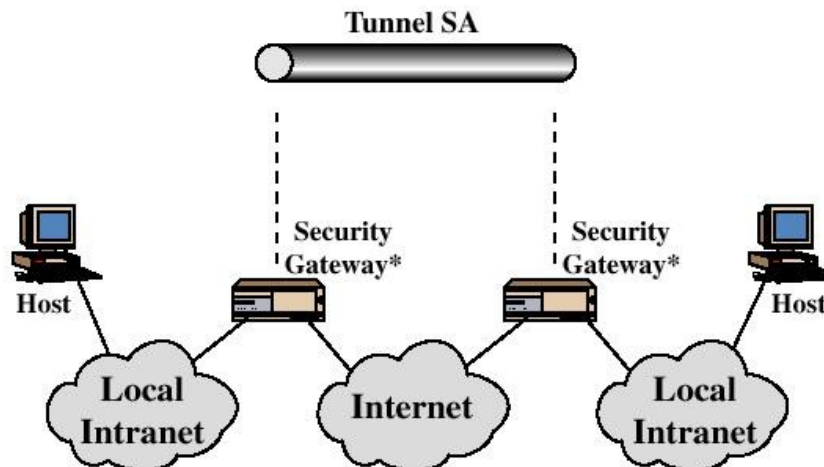


(a) Case 1

Case 2. Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPsec.

This case illustrates simple virtual private network support.

The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required, because the IPsec services apply to the entire inner packet.



(b) Case 2

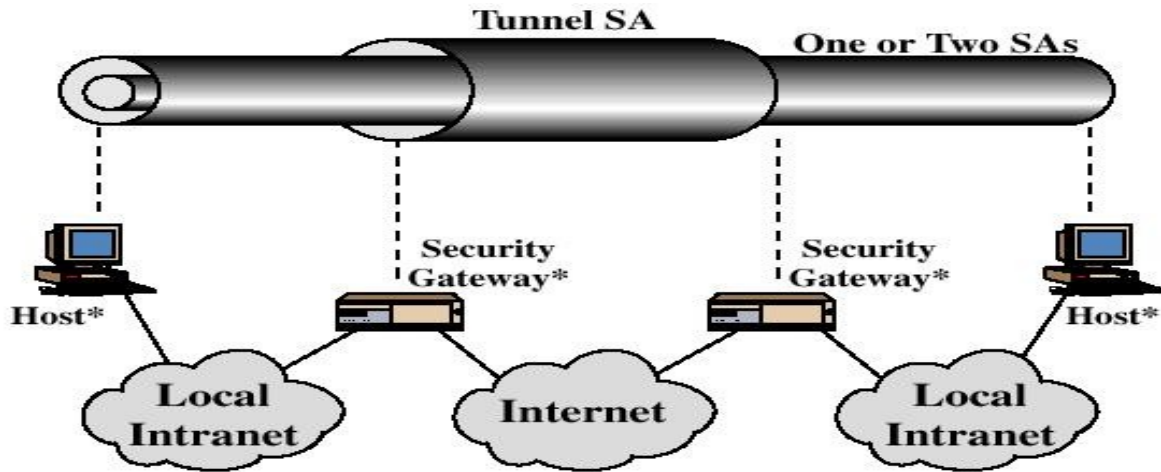
Case 3. This builds on case 2 by adding end-to-end security.

The same combinations discussed for cases 1 and 2 are allowed here.

The gateway-to-gateway tunnel provides either authentication, confidentiality, or both for all traffic between end systems.

When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality.

Individual hosts can implement any additional IPsec services required for given applications or given users by means of end-to-end SAs.

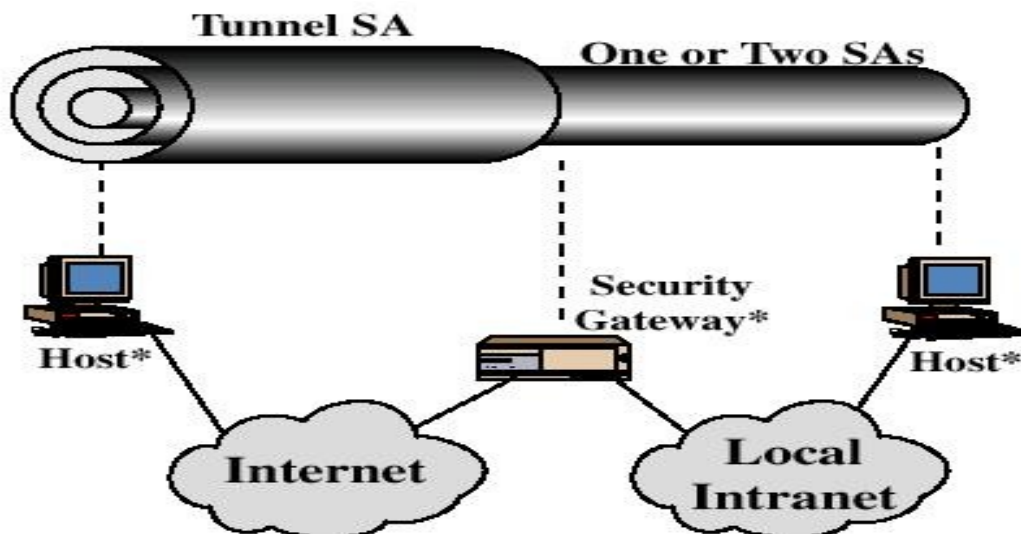


(c) Case 3

Case 4. This provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall.

Only tunnel mode is required between the remote host and the firewall.

As in case 1, one or two SAs may be used between the remote host and the local host.



(d) Case 4

5.6 Key Management:

The key management portion of IPsec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both integrity and confidentiality. The IPsec Architecture document mandates support for two types of key management:

Manual: A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.

Automated: An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

Oakley Key Determination Protocol

Internet Security Association and Key Management Protocol
(ISAKMP)

Oakley Key Determination Protocol: Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats. **Internet Security Association and Key Management Protocol (ISAKMP):**

ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

Key Determination Protocol

IKE key determination is a refinement of the Diffie-Hellman key exchange algorithm. Recall that Diffie-Hellman involves the following interaction between users A and B. There is prior agreement on two global parameters: q , a large prime number; and α , a primitive root of q . A selects a random integer X_A as its private key and transmits to B its public key $Y_A = \alpha^{X_A} \bmod q$. Similarly, B selects a random integer X_B as its private key and transmits to A its public key $Y_B = \alpha^{X_B} \bmod q$. Each side can now compute the secret session key:

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q = \alpha^{X_A X_B} \bmod q$$

The Diffie-Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.
- The exchange requires no pre-existing infrastructure other than an agreement on the global parameters.

However, there are a number of weaknesses to Diffie-Hellman, as pointed out in [HUIT98].

- It does not provide any information about the identities of the parties.
- It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with B. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic. The man-in-the-middle attack proceeds as
 1. B sends his public key Y_B in a message addressed to A (see Figure 3.13).
 2. The enemy (E) intercepts this message. E saves B's public key and sends a message to A that has B's User ID but E's public key Y_E . This message is

sent in such a way that it appears as though it was sent from B's host system. A receives E's message and stores E's public key with B's User ID. Similarly, E sends a message to B with E's public key, purporting to come from A.

3. B computes a secret key K_1 based on B's private key and Y_E . A computes a secret key K_2 based on A's private key and Y_E . E computes K_1 using E's secret key X_E and Y_B and computes K_2 using X_E and Y_A .
4. From now on, E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.
- It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys. The victim spends considerable computing resources doing useless modular exponentiation rather than real work.

IKE key determination is designed to retain the advantages of Diffie-Hellman, while countering its weaknesses.

FEATURES OF IKE KEY DETERMINATION The IKE key determination algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

clogging attacks.

In this attack, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim.

The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can *clog the victim's system* with useless work.

The cookie exchange requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges.

This acknowledgment must be repeated in the first message of the Diffie-Hellman key exchange. If the source address was forged, the opponent gets no answer.

Thus, an opponent can only force a user to generate acknowledgments and not to perform the Diffie-Hellman calculation.

IKE mandates that cookie generation satisfy three basic requirements:

- The cookie must depend on the specific parties.
- It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity.
- The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources.

The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value.

IKE key determination supports the use of different groups for the Diffie- Hellman key exchange.

Each group includes the definition of the two global parameters and the identity of the algorithm.

The current specification includes the following groups.

- Modular exponentiation with a 768-bit modulus

$$q = 2^{768} - 2^{704} - 1 + 2^{64} \times (\lfloor 2^{638} \times \pi \rfloor + 149686)$$

$$\alpha = 2$$

- Modular exponentiation with a 1024-bit modulus

$$q = 2^{1024} - 2^{960} - 1 + 2^{64} \times (\lfloor 2^{894} \times \pi \rfloor + 129093)$$

$$\alpha = 2$$

- Modular exponentiation with a 1536-bit modulus

- Parameters to be determined

- Elliptic curve group over 2^{155}

- Generator (hexadecimal): $X = 7B, Y = 1C8$

- Elliptic curve parameters (hexadecimal): $A = 0, Y = 7338F$

- Elliptic curve group over 2^{185}

- Generator (hexadecimal): $X = 18, Y = D$

- Elliptic curve parameters (hexadecimal): $A = 0, Y = 1EE9$

Three different authentication methods can be used with IKE key determination:

Digital signatures: The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.

Public-key encryption: The exchange is authenticated by encrypting parameters such as IDs and nonces with the sender's private key.

Symmetric-key encryption: A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

Oakley Exchange Example

The Oakley specification includes a number of examples of exchanges that are allowable under the protocol. To give a flavor of Oakley, we present one example, called aggressive key exchange in the specification, so called because only three messages are exchanged.

Figure below shows the aggressive key exchange protocol. In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms.

When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's

nonce for this exchange. Finally, R appends a signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

$I \rightarrow R$: CKY _I , OK_KEYX, GRP, g^x , EHAO, NIDP, ID _I , ID _R , N _I , SKI[ID _I ID _R N _I GRP g^x EHAO]
$R \rightarrow I$: CKY _R , CKY _I , OK_KEYX, GRP, g^y , EHAS, NIDP, ID _R , IQ, N _R , N _I , SKR[ID _R ID _I N _R N _I GRP g^y g^x EHAS]
$I \rightarrow R$: CKY _I , CKY _R , OK_KEYX, GRP, g^x , EHAS, NIDP, ID _I , ID _R , N _I , N _R , SKI[ID _I ID _R N _I N _R GRP g^x g^y EHAS]

Notation:

I	=	Initiator
R	=	Responder
CKY _I , CKY _R	=	Initiator, responder cookies
OK_KEYX	=	Key exchange message type
GRP	=	Name of Diffie-Hellman group for this exchange
g^x, g^y	=	Public key of initiator, responder; g^{xy} = session key from this exchange
EHAO, EHAS	=	Encryption, hash, authentication functions, offered and selected
NIDP	=	Indicates encryption is not used for remainder of this message
ID _I , ID _R	=	Identifier for initiator, responder
N _I , N _R	=	Random nonce supplied by initiator, responder for this exchange
SKI[X], SKR[X]	=	Indicates the signature over X using the private key (signing key) of initiator, responder

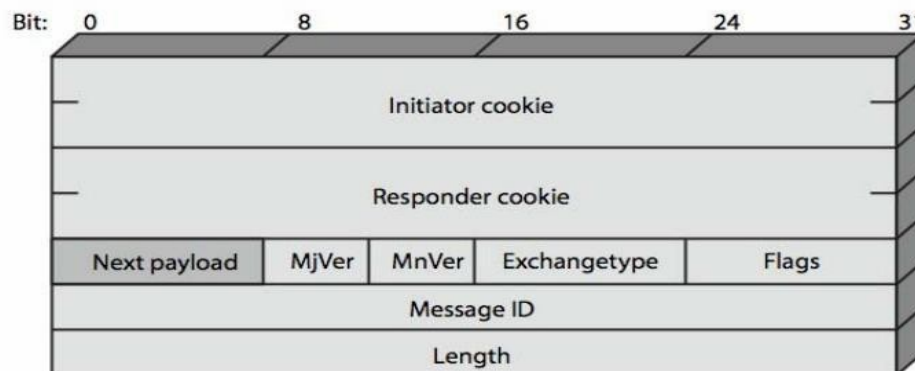
Figure 13.11 Example of Aggressive Oakley Key Exchange.

ISAKMP (Internet Security Association and Key Management Protocol):

Provides framework for key management.

Defines procedures and packet formats to establish, negotiate, modify, & delete SAs.

Independent of key exchange protocol, encryption alg, & authentication method.



(a) ISAKMP Header



(b) Generic Payload Header

Initiator SPI (64 bits): A value chosen by the initiator to identify a unique IKE security association (SA).

Responder SPI (64 bits): A value chosen by the responder to identify a unique IKE SA.

Next Payload (8 bits): Indicates the type of the first payload in the message; payloads are discussed in the next subsection.

Major Version (4 bits): Indicates major version of IKE in use.

Minor Version (4 bits): Indicates minor version in use.

Exchange Type (8 bits): Indicates the type of exchange; these are discussed later in this section.

Flags (8 bits): Indicates specific options set for this IKE exchange. Three bits are defined so far. The initiator bit indicates whether this packet is sent by the SA initiator. The version bit indicates whether the transmitter is capable of using a higher major version number than the one currently indicated. The response bit indicates whether this is a response to a message containing the same message ID.

Message ID (32 bits): Used to control retransmission of lost packets and matching of requests and responses.

Length (32 bits): Length of total message (header plus all payloads) in octets.

ISAKMP Payloads & Exchanges:

have a number of ISAKMP **payload** types:

- ▯ Security, Proposal, Transform, Key, Identification, Certificate, Certificate, Hash, Signature, Nonce, Notification, Delete

ISAKMP has framework for 5 types of **message** exchanges:

- ▯ Base, identity protection, authentication only, aggressive, informational.

payload types:

Security Association (SA): used to begin the setup of a new SA; carries various attributes

Proposal (P): used during SA setup; indicates protocol to be used (AH or ESP) and number of transforms

Transform (T): used during SA setup; indicates transform (e.g., DES, 3DES) and its attributes

Key exchange (KE): used to carry key exchange data (e.g., Oakley)

Identification (ID): used to exchange identification information (e.g., IP address)

Certificate (CR): carries a **public key certificate (PGP, X.509, SPKI, ...)**

Hash (HASH): The **Authentication** payload contains data used for message authentication purposes. The authentication method types so far defined are RSA digital signature, shared-key message integrity code, and DSS digital signature.

Signature (SIG): The **Notify** payload contains either error or status information associated with this SA or this SA negotiation.

Nonce (NONCE): The **Nonce** payload contains random data used to guarantee liveness during an exchange and to protect against replay attacks.

Notification (N): contains error or status information

Delete (D): indicates one or more SAs that the sender has deleted from its database (no longer valid).

Type	Parameters	Description
Security Association (SA)	Domain of Interpretation, Situation	Used to negotiate security attributes and indicate the DOI and Situation under which negotiation is taking place.
Proposal (P)	Proposal #, Protocol-ID, SPI Size, # of Transforms, SPI	Used during SA negotiation; indicates protocol to be used and number of transforms.
Transform (T)	Transform #, Transform-ID, SA Attributes	Used during SA negotiation; indicates transform and related SA attributes.
Key Exchange (KE)	Key Exchange Data	Supports a variety of key exchange techniques.
Identification (ID)	ID Type, ID Data	Used to exchange identification information.
Certificate (CERT)	Cert Encoding, Certificate Data	Used to transport certificates and other certificate-related information.
Certificate Request (CR)	# Cert Types, Certificate Types, # Cert Auths, Certificate Authorities	Used to request certificates; indicates the types of certificates requested and the acceptable certificate authorities.
Hash (HASH)	Hash Data	Contains data generated by a hash function.
Signature (SIG)	Signature Data	Contains data generated by a digital signature function.
Nonce (NONCE)	Nonce Data	Contains a nonce.
Notification (N)	DOI, Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data	Used to transmit notification data, such as an error condition.
Delete (D)	DOI, Protocol-ID, SPI Size, # of SPIs, SPI (one or more)	Indicates an SA that is no longer valid.

ISAKMP Exchanges

ISAKMP provides a framework for message exchange, with the payload types serving as the building blocks. The specification identifies five default exchange types that should be supported;

The **Base Exchange** allows key exchange and authentication material to be transmitted together. This minimizes the number of exchanges at the expense of not providing identity protection. The first two messages provide cookies and establish an SA with agreed protocol and transforms; both sides use a nonce to ensure against replay attacks. The last two messages exchange the key material and user IDs, with the AUTH payload used to authenticate keys, identities. and the nonces from the first two messages.

The **Identity Protection Exchange** expands the Base Exchange to protect the users' identities. The first two messages establish the SA. The next two messages perform key exchange, with nonces for replay protection. Once the session key has been computed, the two parties exchange encrypted messages that contain authentication information, such as digital signatures and optionally certificates validating the public keys.

The **Authentication Only Exchange** is used to perform mutual authentication, without a key exchange. The first two messages establish the SA. In addition, the responder uses the second message to convey its ID and uses authentication to protect the message. The initiator sends the third message to transmit its authenticated ID.

The **Aggressive Exchange** minimizes the number of exchanges at the expense of not providing identity protection. In the first message, the initiator proposes an SA with

associated offered protocol and transform options. The initiator also begins the key exchange and provides its ID. In the second message, the responder indicates its acceptance of the SA with a particular protocol and transform, completes the key exchange, and authenticates the transmitted information. In the third message, the initiator transmits an authentication result that covers the previous information, encrypted using the shared secret session key.

The **Informational Exchange** is used for one-way transmittal of information for SA management.

Exchange	Note
(a) Base Exchange	
(1) I → R: SA; NONCE	Begin ISAKMP-SA negotiation
(2) R → I: SA; NONCE	Basic SA agreed upon
(3) I → R: KE; ID _I ; AUTH	Key generated; Initiator identity verified by responder
(4) R → I: KE; ID _R ; AUTH	Responder identity verified by initiator; Key generated; SA established
(b) Identity Protection Exchange	
(1) I → R: SA	Begin ISAKMP-SA negotiation
(2) R → I: SA	Basic SA agreed upon
(3) I → R: KE; NONCE	Key generated
(4) R → I: KE; NONCE	Key generated
(5)* I → R: ID _I ; AUTH	Initiator identity verified by responder
(6)* R → I: ID _R ; AUTH	Responder identity verified by initiator; SA established
(c) Authentication Only Exchange	
(1) I → R: SA; NONCE	Begin ISAKMP-SA negotiation
(2) R → I: SA; NONCE; ID _R ; AUTH	Basic SA agreed upon; Responder identity verified by initiator
(3) I → R: ID _I ; AUTH	Initiator identity verified by responder; SA established
(d) Aggressive Exchange	
(1) I → R: SA; KE; NONCE; ID _I	Begin ISAKMP-SA negotiation and key exchange
(2) R → I: SA; KE; NONCE; ID _R ; AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)* I → R: AUTH	Responder identity verified by initiator; SA established
(e) Informational Exchange	
(1)* I → R: N/D	Error or status notification, or deletion

Notation:
I = initiator
R = responder
***** = signifies payload encryption after the ISAKMP header

UNIT-VI: WEB SECURITY

Web Security Considerations, Secure Socket Layer (SSL) and Transport Layer Security (TLS), Secure Electronic Transaction (SET).

Introduction:

Virtually all businesses, most government agencies, and many individuals now have Web sites. The number of individuals and companies with Internet access is expanding rapidly and all of these have graphical Web browsers. As a result, businesses are enthusiastic about setting up facilities on the Web for electronic commerce. But the reality is that the Internet and the Web are extremely vulnerable to compromises of various sorts. As businesses wake up to this reality, the demand for secure Web services grows.

6.1 Web Security Considerations:

The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets. the Web presents new challenges not generally appreciated in the context of computer and network security.

The Internet is two-way. Unlike traditional publishing environments—even electronic publishing systems involving teletext, voice response, or fax-back— the Web is vulnerable to attacks on the Web servers over the Internet.

The Web is increasingly serving as a highly visible outlet for corporate and product information and as the platform for business transactions. Reputations can be damaged and money can be lost if the Web servers are subverted.

Although Web browsers are very easy to use, Web servers are relatively easy to configure and manage, and Web content is increasingly easy to develop, the underlying software is extraordinarily complex. This complex software may hide many potential security flaws. The short history of the Web is filled with examples of new and upgraded systems, properly installed, that are vulnerable to a variety of security attacks.

A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex. Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.

Casual and untrained (in security matters) users are common clients for Web-based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

6.1.1 Web Security Threats

Web now widely used by business, government, individuals
but Internet & Web are vulnerable
have a variety of threats

- Integrity

- The customer and the vendor need to be sure that the contents of the message are not modified during transmission.

- Confidentiality

- The customer and the vendor need to be sure that an imposter does not intercept sensitive information such as a credit card number.

- denial of service

- Authentication

- The customer need to be sure that the server belongs to the actual vendor, not an imposter. The customer does not want to give an imposter her credit card number.

need added security mechanisms

Table below provides a summary of the types of security threats faced when using the Web.

Table 5.1 A Comparison of Threats on the Web

	Threats	Consequences	Countermeasures
Integrity	<ul style="list-style-type: none"> • Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit 	<ul style="list-style-type: none"> • Loss of information • Compromise of machine • Vulnerability to all other threats 	Cryptographic checksums
Confidentiality	<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • Loss of information • Loss of privacy 	Encryption, Web proxies
Denial of Service	<ul style="list-style-type: none"> • Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks 	<ul style="list-style-type: none"> • Disruptive • Annoying • Prevent user from getting work done 	Difficult to prevent
Authentication	<ul style="list-style-type: none"> • Impersonation of legitimate users • Data forgery 	<ul style="list-style-type: none"> • Misrepresentation of user • Belief that false information is valid 	Cryptographic techniques

6.1.2: Web Traffic Security Approaches

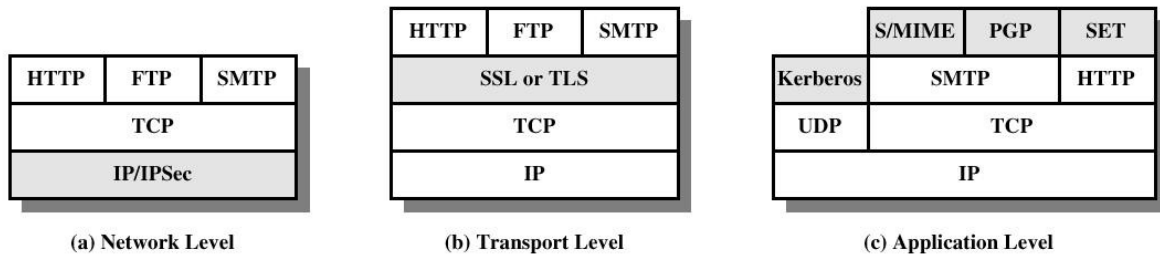
A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

Figure 5.1 illustrates this difference. One way to provide Web security is to use IP security (IPsec) (Figure 5.1a). 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 5.1b). 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 5.1c 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.



6.2 SECURE SOCKET LAYER (SSL) AND TRANSPORT LAYER SECURITY (TLS):

SSL – Secure Socket Layer

TLS – Transport Layer Security

both provide a secure transport connection between applications (e.g., a web server and a browser)

SSL was developed by Netscape

SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and MS Internet Explorer) and web servers and widely used on the Internet

SSL v3.0 was specified in an Internet Draft (1996)

it evolved into TLS specified in RFC 2246

TLS can be viewed as SSL v3.1

6.2.1 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.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows.

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.

Connection: A connection is a transport 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.



A **session state** is defined by the following parameters.

Session identifier: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.

Peer certificate: An X509.v3 certificate of the peer. This element of the state may be null.

Compression method: The algorithm used to compress data prior to encryption.

Cipher spec: Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.

Master secret: 48-byte secret shared between the client and server.

Is resumable: A flag indicating whether the session can be used to initiate new connections.



A **connection state** is defined by the following parameters.

Server and client random: Byte sequences that are chosen by the server and client for each connection.

Server write MAC secret: The secret key used in MAC operations on data sent by the server.

Client write MAC secret: The secret key used in MAC operations on data sent by the client.

Server write key: The secret encryption key for data encrypted by the server and decrypted by the client.

Client write key: The symmetric encryption key for data encrypted by the client and decrypted by the server.

Initialization vectors: When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter, the final ciphertext block from each record is preserved for use as the IV with the following record.

Sequence numbers: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64} - 1$.

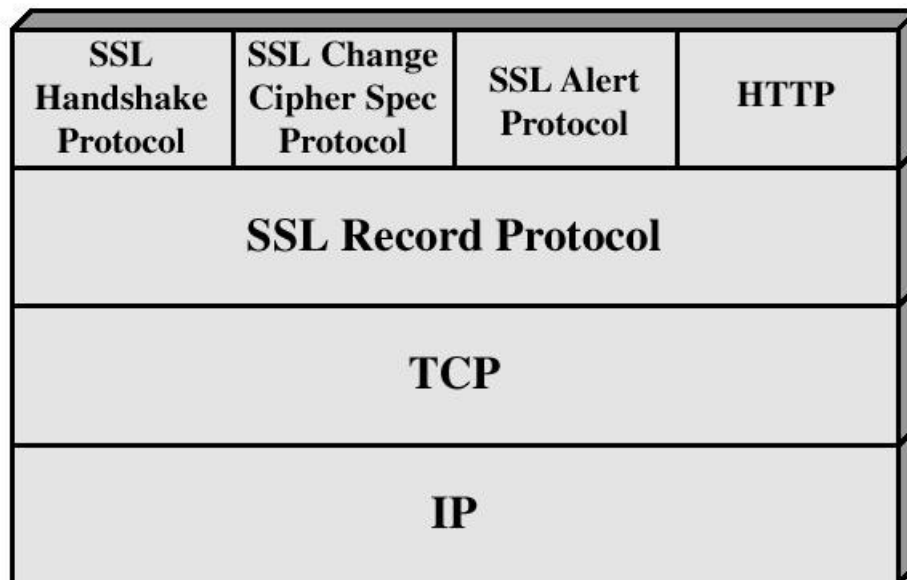


Figure 7.2 SSL Protocol Stack

SSL components:

SSL Handshake Protocol

- negotiation of security algorithms and parameters
- key exchange
- server authentication and optionally client authentication

SSL Record Protocol

- ▢ fragmentation
- ▢ compression
- ▢ message authentication and integrity protection
- ▢ encryption

SSL Alert Protocol

- ▢ error messages (fatal alerts and warnings)

SSL Change Cipher Spec Protocol

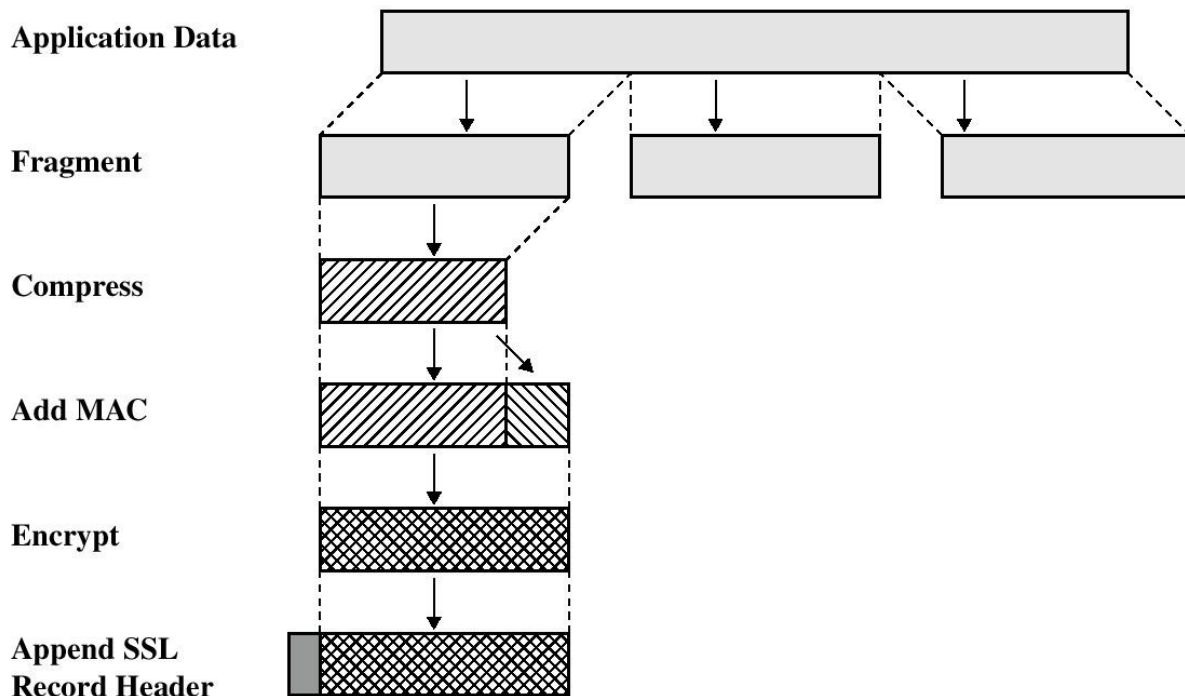
- ▢ a single message that indicates the end of the SSL handshake

SSL Record Protocol:

The SSL Record Protocol provides two services for SSL connections:

- ▢ Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- ▢ Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code(MAC).

SSL Record Protocol Operation:



The first step is **fragmentation**.

Each upper-layer message is fragmented into blocks of 2^{14} bytes (16384 bytes) or less. Next, compression is optionally applied.

Compression must be lossless and may not increase the content length by more than 1024 bytes.

In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.

The next step in processing is to compute a **message authentication code over** the compressed data. For this purpose, a shared secret key is used.

The calculation is defined as.


```
hash(MAC_write_secret || pad_2 ||
      hash(MAC_write_secret || pad_1 || seq_num ||
            SSLCompressed.type || SSLCompressed.length ||
            SSLCompressed.fragment))
```

where

	= concatenation
MAC_write_secret	= shared secret key
hash	= cryptographic hash algorithm; either MD5 or SHA-1
pad_1	= the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
pad_2	= the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1
seq_num	= the sequence number for this message
SSLCompressed.type	= the higher-level protocol used to process this fragment
SSLCompressed.length	= the length of the compressed fragment
SSLCompressed.fragment	= the compressed fragment (if compression is not used, this is the plaintext fragment)

►

Next, the **compressed message plus the MAC** are encrypted using symmetric encryption.

Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed $2^{14} + 2048$.

The following encryption algorithms are permitted:

Block Cipher		Stream Cipher	
Algorithm	Key Size	Algorithm	Key Size
AES	128, 256	RC4-40	40
IDEA	128	RC4-128	128
RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Fortezza can be used in a smart card encryption scheme.

For stream encryption, the compressed message plus the MAC are encrypted.

Note that the MAC is computed before encryption takes place and that the MAC is then encrypted along with the plaintext or compressed plaintext.

For block encryption, padding may be added after the MAC prior to encryption.

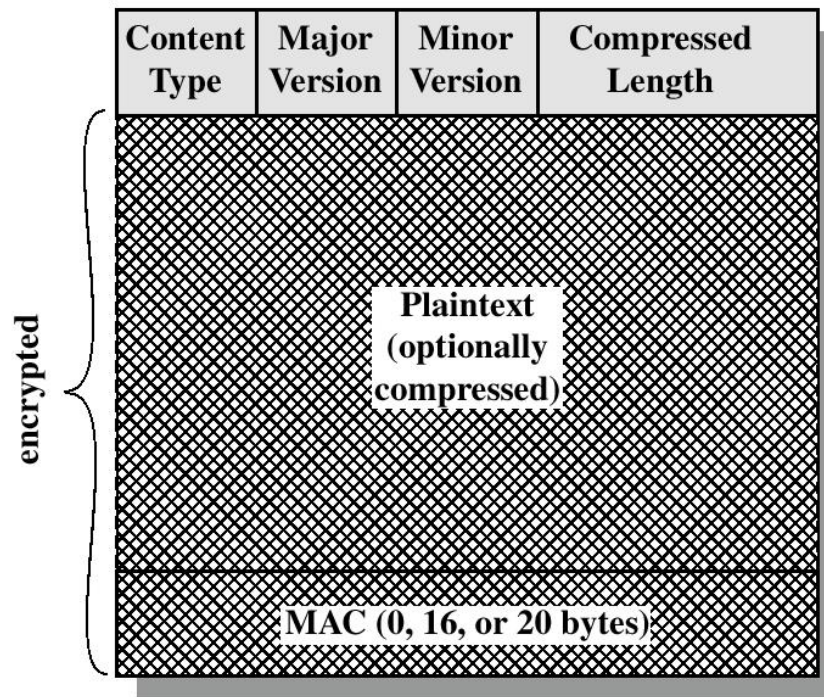
The final step of SSL Record Protocol processing is to prepare a header consisting of the following fields:

Content Type (8 bits): The higher-layer protocol used to process the enclosed fragment.

Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the value is 3.

Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.

Compressed Length (16 bits): The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is $2^{14} + 2048$. **SSL Record Format**



Change Cipher Spec Protocol:

This protocol consists of a single message, which consists of a single byte with the value 1.

The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

Alert Protocol:

is used to convey SSL-related alerts to the peer entity.

Each message in this protocol consists of two bytes .

The first byte takes the value warning (1) or fatal (2) to convey the severity of the message.

If the level is fatal, SSL immediately terminates the connection.

Other connections on the same session may continue, but no new connections on this session may be established.

The second byte contains a code that indicates the specific alert.

The following alerts that are always fatal.

- **unexpected_message:** An inappropriate message was received.
- **bad_record_mac:** An incorrect MAC was received.
- **decompression_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter:** A field in a handshake message was out of range or inconsistent with other fields.

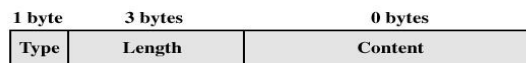
Remaining alerts:

- **close_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a **close_notify** alert before closing the write side of a connection.
- **no_certificate:** May be sent in response to a certificate request if no appropriate certificate is available.
- **bad_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate:** The type of the received certificate is not supported.
- **certificate_revoked:** A certificate has been revoked by its signer.
- **certificate_expired:** A certificate has expired.
- **certificate_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

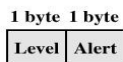
Formats of various protocols:



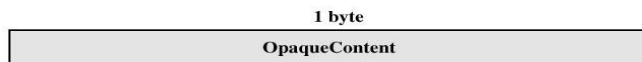
(a) Change Cipher Spec Protocol



(c) Handshake Protocol



(b) Alert Protocol



(d) Other Upper-Layer Protocol (e.g., HTTP)

Handshake Protocol:

The most complex part of SSL.

Allows the server and client to authenticate each other.

Negotiate encryption, MAC algorithm and cryptographic keys.

Used before any application data are transmitted.

The Handshake Protocol consists of a series of messages exchanged by client and server.

Each message has three fields:

Type (1 byte): Indicates one of 10 messages.

Length (3 bytes): The length of the message in bytes.

Content (≥bytes): The parameters associated with this message;

Table 5.2 SSL Handshake Protocol Message Types

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Figure below shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases. PHASE 1.

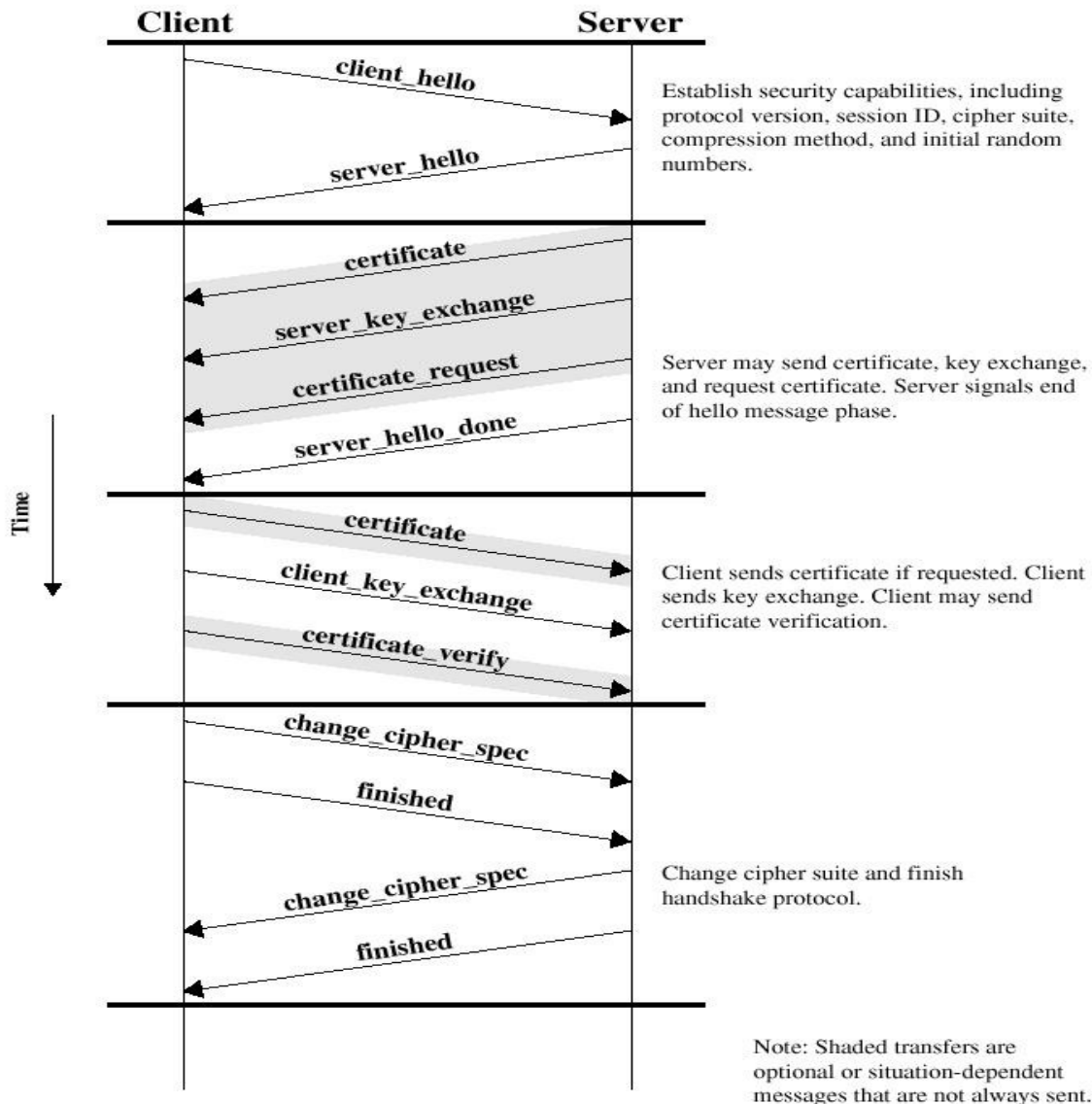
ESTABLISH SECURITY CAPABILITIES

PHASE 2. SERVER AUTHENTICATION AND KEY EXCHANGE

PHASE 3. CLIENT AUTHENTICATION AND KEY EXCHANGE

PHASE 4. FINISH

Handshake Protocol Action:



PHASE 1. ESTABLISH SECURITY CAPABILITIES

- This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it.
- The exchange is initiated by the client, which sends a `client_hello` message with the following parameters:
- Version: The highest SSL version understood by the client.
- Random: A client-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator.
- These values serve as nonces and are used during key exchange to prevent replay attacks.

Session ID:

- A variable-length session identifier.
- A nonzero value indicates that the client wishes to update the parameters of an existing connection or to create a new connection on this session.

- A zero value indicates that the client wishes to establish a new connection on a new session.

CipherSuite:

- This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference.
- Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec;
- Compression Method: This is a list of the compression methods the client supports.

After sending the *client_hello* message, the client waits for the *server_hello* message, which contains the same parameters as the *client_hello* message.

For the *server_hello* message, the following conventions apply.

The Version field contains the lower of the versions suggested by the client and the highest supported by the server.

The Random field is generated by the server and is independent of the client's Random field.

If the *SessionID* field of the client was nonzero, the same value is used by the server; otherwise the server's *SessionID* field contains the value for a new session.

The *CipherSuite* field contains the single cipher suite selected by the server from those proposed by the client.

The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the *CipherSuite* parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged). The following key exchange methods are supported.

- RSA: The secret key is encrypted with the receiver's RSA public key.
- A publickey certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman**: This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA).
- The client provides its Diffie-Hellman public-key parameters either in a certificate, if client authentication is required, or in a key exchange message.
- This method results in a fixed secret key between two peers based on the Diffie-Hellman calculation using the fixed public keys.
- **Ephemeral Diffie-Hellman**: This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key.
- The receiver can use the corresponding public key to verify the signature.
- Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options, because it results in a temporary, authenticated key.
- ← **Anonymous Diffie-Hellman**: The base Diffie-Hellman algorithm is used with no authentication.
- That is, each side sends its public Diffie-Hellman parameters to the other with no authentication.
- This approach is vulnerable to man-in-the middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- **Fortezza**: the technique defined by US National security Agency.
- Developed for defense department.

Following the definition of a key exchange method is the CipherSpec, which includes the following fields.

Cipher Algorithm: Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, or Fortezza.

MACAlgorithm: MD5 or SHA-1

CipherType: Stream or Block

IsExportable: True or False

HashSize: 0, 16 (for MD5), or 20 (for SHA-1) bytes

Key Material: A sequence of bytes that contain data used in generating the write keys

IV Size: The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

SSL Handshake Protocol / Phase 2

Server certificate and key exchange messages:

certificate

- required for every key exchange method except for anonymous DH
- contains one or a chain of X.509 certificates (up to a known root CA)
- may contain
 - 0** public RSA key suitable for encryption, or
 - 1** public RSA or DSS key suitable for signing only, or
 - 2** fix DH parameters

server_key_exchange

- sent only if the certificate does not contain enough information to complete the key exchange (e.g., the certificate contains an RSA signing key only)
- may contain
 - 0** public RSA key (exponent and modulus), or
 - 1** DH parameters (p, g, public DH value), or
 - 2** Fortezza parameters
- digitally signed
 - 0** if DSS: SHA-1 hash of (client_random | server_random | server_params) is signed
 - 1** if RSA: MD5 hash and SHA-1 hash of (client_random | server_random | server_params) are concatenated and encrypted with the private RSA key

Certificate request and server hello done msgs:

certificate_request

- sent if the client needs to authenticate itself
- specifies which type of certificate is requested (rsa_sign, dss_sign, **rsa_fixed_dh**, **dss_fixed_dh**, ...)

server_hello_done

- sent to indicate that the server is finished its part of the key exchange
- after sending this message the server waits for client response
- the client should verify that the server provided a valid certificate and the server parameters are acceptable

SSL Handshake Protocol / Phase 3

Client authentication and key exchange

certificate

- sent only if requested by the server
- may contain
 - 0** public RSA or DSS key suitable for signing only, or
 - 1** fix DH parameters

client_key_exchange

- always sent (but it is empty if the key exchange method is fix DH)
 - may contain
 - 0 RSA encrypted pre-master secret, or
 - 1 client one-time public DH value, or
 - 2 Fortezza key exchange parameters
 - certificate_verify
 - sent only if the client sent a certificate
 - provides client authentication
 - contains signed hash of all the previous handshake messages
 - 0 if DSS: SHA-1 hash is signed
 - 1 if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key
- MD5(master_secret|pad_2|MD5(handshake_messages | master_secret | pad_1))
 SHA(master_secret|pad_2|SHA(handshake_messages | master_secret | pad_1))

SSL Handshake Protocol / Phase 4:

Finished messages:

- finished
- sent immediately after the change_cipher_spec message
 - first message that uses the newly negotiated algorithms, keys, IVs, etc.
 - used to verify that the key exchange and authentication was successful
 - contains the MD5 and SHA-1 hash of all the previous handshake messages:
- MD5(master_secret|pad_2|MD5(handshake_messages|sender|master_secret|pad_1)) |SHA(master_secret|pad_2|SHA(handshake_messages|sender|master_secret| pad_1))
- where "sender" is a code that identifies that the sender is the client or the server (client: 0x434C4E54; server: 0x53525652)

Cryptographic Computations:

Two further items are of interest: (1) the creation of a shared master secret by means of the key exchange and (2) the generation of cryptographic parameters from the master secret.

MASTER SECRET CREATION The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a *pre_master_secret* is exchanged. Second, the *master_secret* is calculated by both parties. For *pre_master_secret* exchange, there are two possibilities.

RSA: A 48-byte *pre_master_secret* is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the *pre_master_secret*.

Diffie-Hellman: Both client and server generate a Diffie-Hellman public key. After these are exchanged, each side performs the Diffie-Hellman calculation to create the shared *pre_master_secret*.

Both sides now compute the *master_secret* as

```
master_secret=MD5(pre_master_secret||SHA('A'||pre_master_secret|| ClientHello.random ||
ServerHello.random))||MD5(pre_master_secret||SHA('BB'||pre_master_secret||
ClientHello.random ||ServerHello.random))||MD5(pre_master_secret || SHA('CCC' ||
pre_master_secret || ClientHello.random ||ServerHello.random))
```

where ClientHello.random and ServerHello.random are the two nonce values exchanged in the initial hello messages.

GENERATION OF CRYPTOGRAPHIC PARAMETERS CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters. The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret as

```
key_block = MD5(master_secret || SHA('A' || master_secret || ServerHello.random ||
ClientHello.random)) || MD5(master_secret || SHA('BB' || master_secret || ServerHello.random
|| ClientHello.random)) || MD5(master_secret || SHA('CCC' || master_secret ||
ServerHello.random || ClientHello.random)) || ...
```

until enough output has been generated. The result of this algorithmic structure is a pseudorandom function. We can view the master_secret as the pseudorandom seed value to the function. The client and server random numbers can be viewed as salt values to complicate cryptanalysis (see Chapter 9 for a discussion of the use of salt values).

Transport Layer Security:

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.

Differences in the:

- version number
- message authentication code
- pseudorandom function
- alert codes
- cipher suites
- client certificate types
- certificate_verify and finished message
- cryptographic computations
- ← padding

Version Number

- The TLS Record Format is the same as that of the SSL Record Format (Figure 5.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the major version is 3 and the minor version is 3.

Message Authentication Code

- 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. Recall from Chapter 3 that HMAC is defined as

$$\text{HMAC}_K(M) = H[(K^+ \oplus \text{opad}) \parallel H[(K^+ \oplus \text{ipad}) \parallel M]]$$

where

- H = embedded hash function (for TLS, either MD5 or SHA-1)
- M = message input to HMAC
- K^+ = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
- ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)
- opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
MAC(MAC_write_secret, seq_num || TLSCompressed.type ||
    TLSCompressed.version || TLSCompressed.length ||
    TLSCompressed.fragment)
```

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field `TLSCompressed.version`, which is the version of the protocol being employed.

Pseudorandom Function

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 (Figure below) given as

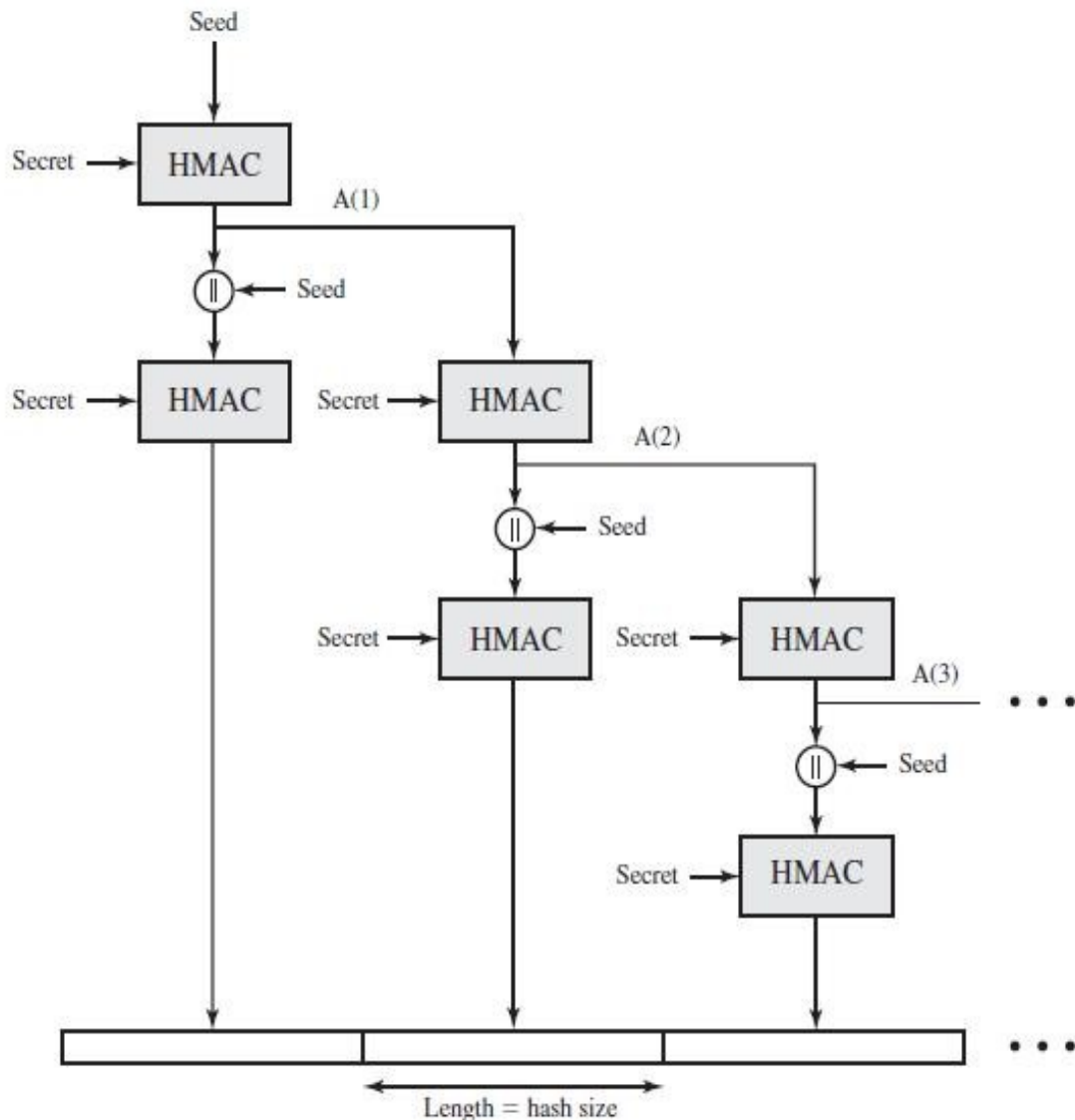
$P_hash(\text{secret}, \text{seed}) = \text{HMAC_hash}(\text{secret}, A(1) \parallel \text{seed}) \parallel \text{HMAC_hash}(\text{secret}, A(2) \parallel \text{seed}) \parallel \text{HMAC_hash}(\text{secret}, A(3) \parallel \text{seed}) \parallel \dots$

where $A()$ is defined as

$A(0) = \text{seed}$

$A(i) = \text{HMAC_hash}(\text{secret}, A(i-1))$

$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = P_MD5(\text{secret_left}, \text{label} \parallel \text{seed}) \mathbin{\&} P_SHA(\text{secret_right}, \text{label} \parallel \text{seed})$



To make PRF as secure as possible, it uses two hash algorithms in a way that should guarantee its security if either algorithm remains secure. PRF is defined as

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_hash}(\text{S1}, \text{label} \parallel \text{seed})$$

PRF takes as input a secret value, an identifying label, and a seed value and produces an output of arbitrary length.

Alert Codes:

TLS supports all of the alert codes defined in SSLv3 with the exception of `no_certificate`.

A number of additional codes are defined in TLS; of these, the following are always fatal.

- **record_overflow:** A TLS record was received with a payload (ciphertext) whose length exceeds $2^{14}+2048$ bytes, or the ciphertext decrypted to a length of greater than $2^{14}+1024$ bytes.
- **unknown_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- **access_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- **decode_error:** A message could not be decoded, because either a field was out of its specified range or the length of the message was incorrect.
- **protocol_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- **insufficient_security:** Returned instead of `handshake_failure` when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- **unsupported_extension:** Sent by clients that receive an extended server hello containing an extension not in the corresponding client hello.
- **internal_error:** An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.
- **decrypt_error:** A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- **user_canceled:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **no_renegotiation:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Secure Electronic Transactions:

An open encryption and security specification.

Protect credit card transaction on the Internet.

Companies involved:

□ MasterCard, Visa, IBM, Microsoft, Netscape, RSA, Terisa and Verisign

Not a payment system.

Set of security protocols and formats.

SET Services:

Provides a secure communication channel in a transaction.

Provides trust by the use of X.509v3 digital certificates.

Ensures privacy.

SET Overview:

A good way to begin our discussion of SET is to look at the business requirements for SET, its key features, and the participants in SET transactions.

Requirements

Book 1 of the SET specification lists the following business requirements for secure payment processing with credit cards over the Internet and other networks:

Provide confidentiality of payment and ordering information: It is necessary to assure cardholders that this information is safe and accessible only to the intended recipient. Confidentiality also reduces the risk of fraud by either party to the transaction or by malicious third parties. SET uses encryption to provide confidentiality.

Ensure the integrity of all transmitted data: That is, ensure that no changes in content occur during transmission of SET messages. Digital signatures are used to provide integrity.

Provide authentication that a cardholder is a legitimate user of a credit card account: A mechanism that links a cardholder to a specific account number reduces the incidence of fraud and the overall cost of payment processing. Digital signatures and certificates are used to verify that a cardholder is a legitimate user of a valid account.

Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution: This is the complement to the preceding requirement. Cardholders need to be able to identify merchants with whom they can conduct secure transactions. Again, digital signatures and certificates are used.

Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction: SET is a well-tested specification based on highly secure cryptographic algorithms and protocols.

Create a protocol that neither depends on transport security mechanisms nor prevents their use: SET can securely operate over a "raw" TCP/IP stack. However, SET does not interfere with the use of other security mechanisms. Such as IPsec and SSL/TLS.

Facilitate and encourage interoperability among software and network providers: The SET protocols and formats are independent of hardware platform, operating system, and Web software.

Key Features of SET:

- Confidentiality of information
- Integrity of data
- Cardholder account authentication
- Merchant authentication

To meet the requirements just outlined, SET incorporates the following features:

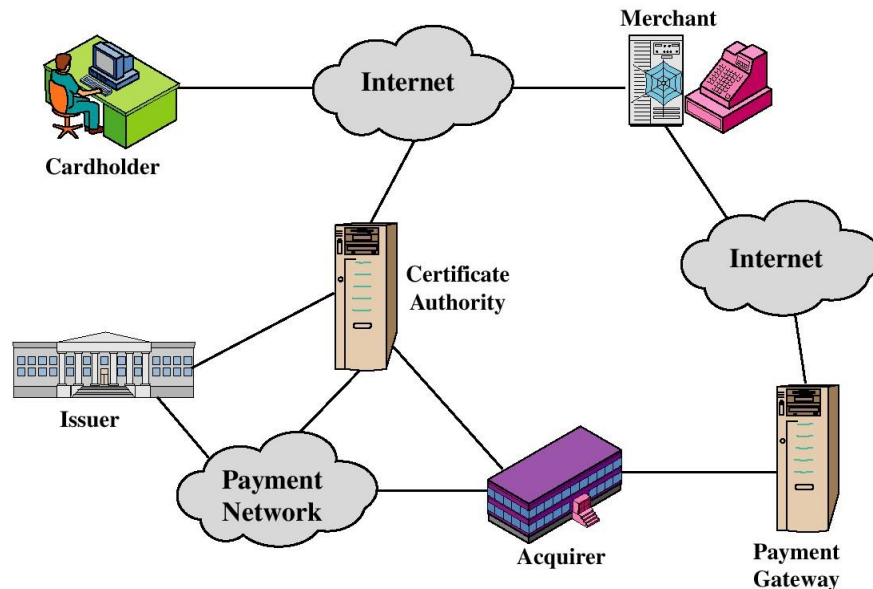
Confidentiality of information: Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.

Integrity of data: Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.

Cardholder account authentication: SET enables merchants to verify that a cardholder is a legitimate user of a valid card account number. SET uses X.509~3 digital certificates with RSA signatures for this purpose.

Merchant authentication: SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509~3 digital certificates with RSA signatures for this purpose.

SET Participants:



Cardholder: In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., Mastercard, Visa) that has been issued by an issuer.

Merchant: A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.

Issuer: This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person. Ultimately, it is the issuer that is responsible for the payment of the debt of the cardholder.

Acquirer: This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.

Payment Gateway: This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.

Certification Authority (CA): This is an entity that is trusted to issue X.509~3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. As was discussed in previous chapters, a hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

We now briefly describe the sequence of events that are required for a transaction. We will then look at some of the cryptographic details.

The customer opens an account. The customer obtains a credit card account, such as Mastercard or Visa, with a bank that supports electronic payment and SET.

The customer receives a certificate. After suitable verification of identity, the customer receives an X.509~3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.

Merchants have their own certificates. A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

The customer places an order. This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.

The merchant is verified. In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.

The order and payment are sent. The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.

The merchant requests payment authorization. The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.

The merchant confirms the order. The merchant sends confirmation of the order to the customer.

The merchant provides the goods or service. The merchant ships the goods or provides the service to the customer.

The merchant requests payment. This request is sent to the payment gateway, which handles all of the payment processing.

Dual Signature:

Before looking at the details of the SET protocol, let us discuss an important innovation introduced in SET: the dual signature. The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, the customer wants to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. However, the two items must be linked in a way that can be used to resolve disputes if necessary. The link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.

To see the need for the link, suppose that the customer sends the merchant two messages—a signed OI and a signed PI—and the merchant passes the PI on to the bank. If the merchant can capture another OI from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this.

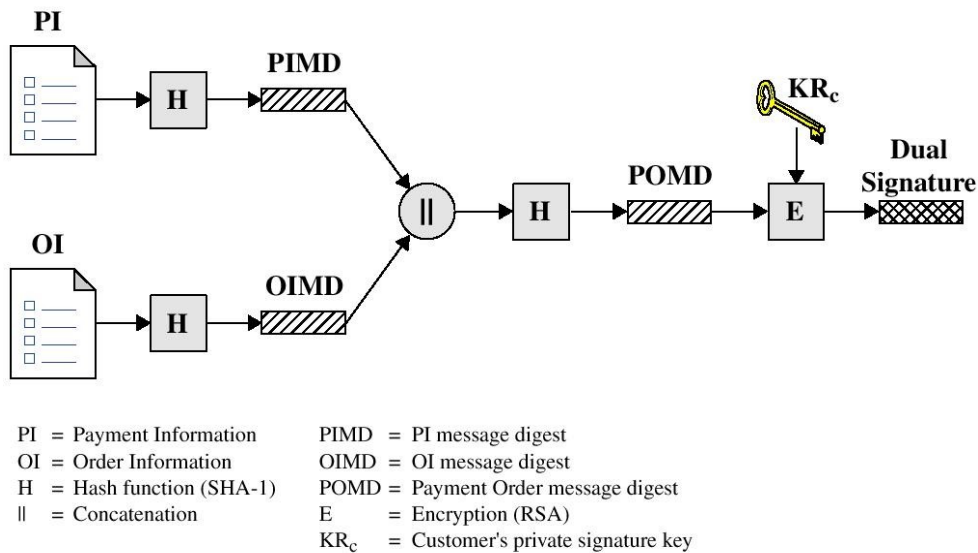


Figure above shows the use of a dual signature to meet the requirement of the preceding paragraph. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as follows:

$$DS = E_{KR_c}[H(H(PI)||H(OI))]$$

where KR_c is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the following two quantities:

$$H(PIMD||H(OI)) \text{ and } D_{KU_c}[DS]$$

where KU_c is the customer's public signature key. If these two quantities are equal, then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute the following:

$$H(H(PI)||OIMD) \text{ and } D_{KU_c}[DS]$$

Again, if these two quantities are equal, then the bank has verified the signature.

In summary,

The merchant has received OI and verified the signature.

The bank has received PI and verified the signature.

The customer has linked the OI and PI and can prove the linkage.

Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The end of this preliminary phase occurs when the merchant sends a completed order form to the customer. All of the preceding occurs without the use of SET.

The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the **Initiate Request** message, sent to the merchant. This message includes the brand of the credit card that the customer is using. The message

also includes an ID assigned to this request/response pair by the customer and a nonce used to ensure timeliness.

The merchant generates a response and signs it with its private signature key. The response includes the nonce from the customer, another nonce for the customer to return in the next message, and a transaction ID for this purchase transaction. In addition to the signed response, the **Initiate Response** message includes the merchant's signature certificate and the payment gateway's key exchange certificate.

The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. The transaction ID assigned by the merchant is placed in both the OI and PI. The OI does not contain explicit order data such as the number and price of items. Rather, it contains an order reference generated in the exchange between merchant and customer during the shopping phase before the first SET message. Next, the cardholder prepares the **Purchase Request** message (Figure 14.10). For this purpose, the cardholder generates a one-time symmetric encryption key, K_r . The message includes the following:

Purchase-related information. This information will be forwarded to the payment gateway by the merchant and consists of

- a The PI
- The dual signature, calculated over the PI and OI, signed with the customer's
- private signature key
- The OI message digest (OIMD)

The OIMD is needed for the payment gateway to verify the dual signature, as explained previously. All of these items are encrypted with K_r . The final item is

- The digital envelope. This is formed by encrypting K_r with the payment gateway's public key-exchange key. It is called a digital envelope because this envelope must be opened (decrypted) before the other items listed previously can be read.

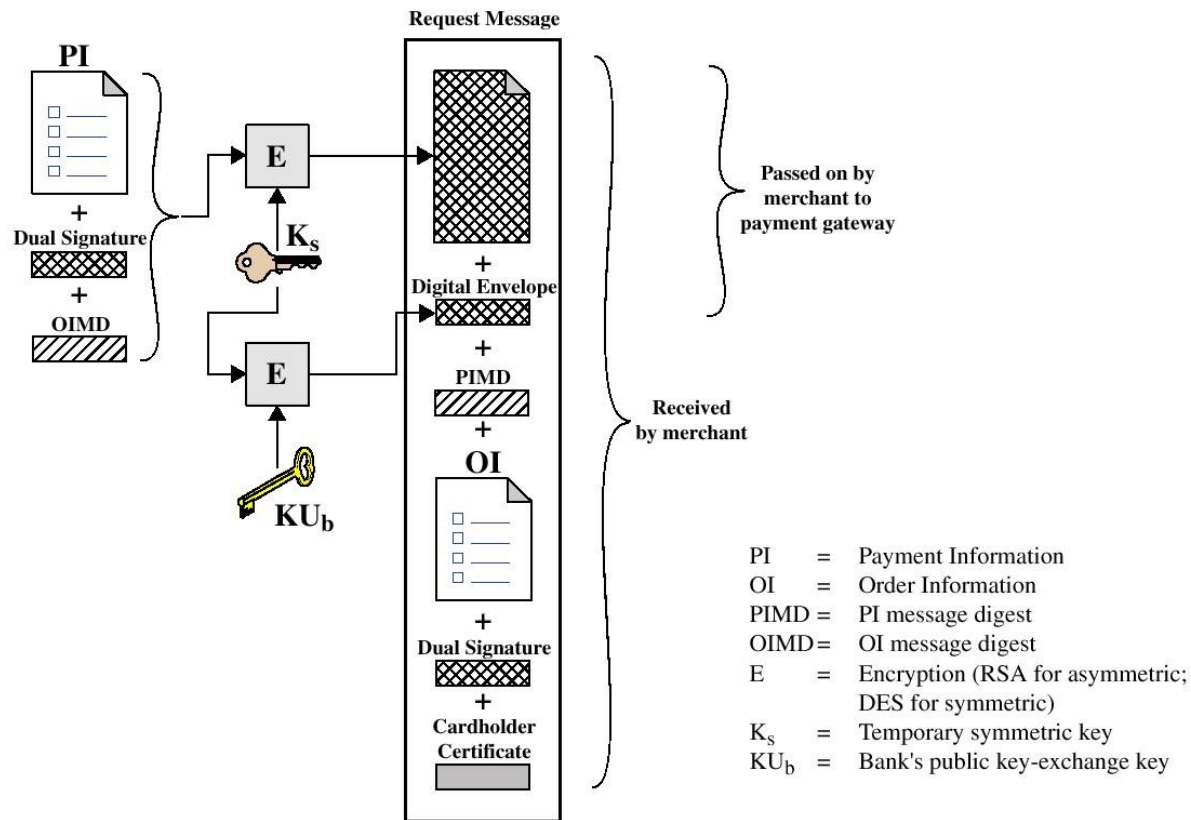
The value of K_r is not made available to the merchant. Therefore, the merchant cannot read any of this payment-related information.

2. **Order-related information.** This information is needed by the merchant and consists of

- The OI
- The dual signature, calculated over the PI and OI, signed with the customer's private signature key
- The PI message digest (PIMD)

The PIMD is needed for the merchant to verify the dual signature. Note that the OI is sent in the clear.

3. **Cardholder certificate.** This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.



When the merchant receives the Purchase Request message, it performs the following actions:

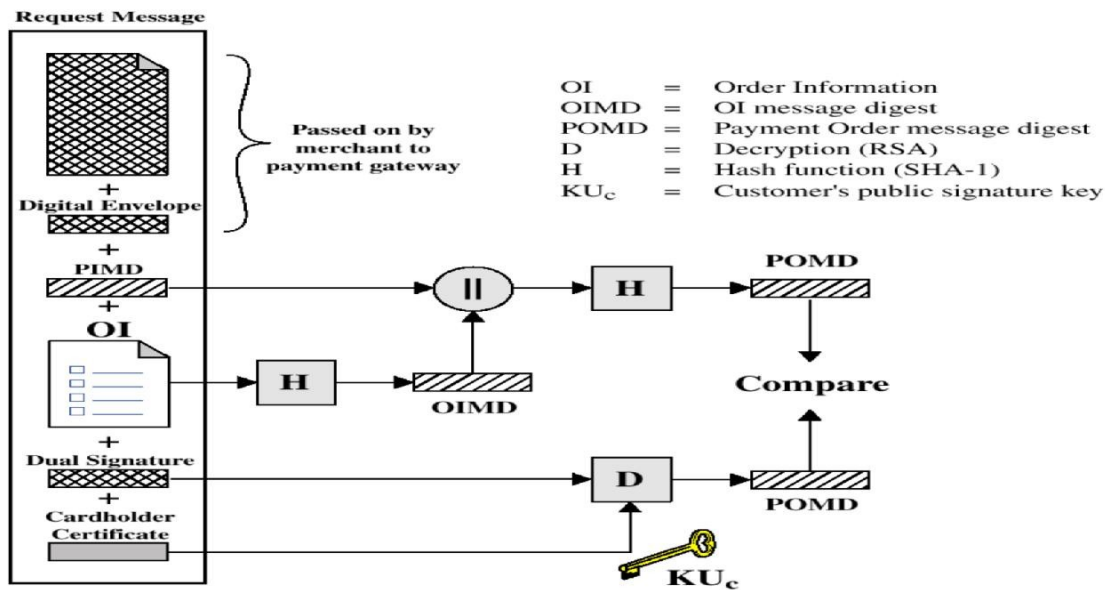
1. Verifies the cardholder certificates by means of its CA signatures.
2. Verifies the dual signature using the customer's public signature key. This ensures that the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.
3. Processes the order and forwards the payment information to the payment gateway for authorization (described later).
4. Sends a purchase response to the cardholder.

The **Purchase Response** message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate.

When the cardholder software receives the purchase response message, it verifies the merchant's certificate and then verifies the signature on the response block. Finally, it takes some action based on the response, such as displaying a message to the user or updating a database with the status of the order.

Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer. This authorization guarantees that the



merchant will receive payment; the merchant can therefore provide the services or goods to the customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response.

The merchant sends an **Authorization Request** message to the payment gateway consisting of

1. **Purchase-related information.** This information was obtained from the customer and consists of:
 - The PI
 - The dual signature, calculated over the PI and OI, signed with the customer's private signature key
 - The OI message digest (OIMD)
 - The digital envelope
2. **Authorization-related information.** This information is generated by the merchant and consists of
 - An authorization block that includes the transaction ID, signed with the merchant's private signature key and encrypted with a one-time symmetric key generated by the merchant
 - A digital envelope. This is formed by encrypting the one-time key with the payment gateway's public key-exchange key.
3. **Certificates.** The merchant includes the cardholder's signature key certificate (used to verify the dual signature), the merchant's signature key certificate (used to verify the merchant's signature), and the merchant's key-exchange certificate (needed in the payment gateway's response).

The payment gateway performs the following tasks:

1. Verifies all certificates
2. Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block
3. Verifies the merchant's signature on the authorization block
4. Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block
5. Verifies the dual signature on the payment block
6. Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer
7. Requests and receives an authorization from the issuer

Having obtained authorization from the issuer, the payment gateway returns an **Authorization Response** message to the merchant. It includes the following elements:

1. **Authorization-related information.** Includes an authorization block, signed with the gateway's private signature key and encrypted with a one-time symmetric key generated by the gateway. Also includes a digital envelope that contains the one-time key encrypted with the merchant's public key-exchange key.
2. **Capture token information.** This information will be used to effect payment later. This block is of the same form as (1)—namely, a signed, encrypted capture token together with a digital envelope. This token is not processed by the merchant. Rather, it must be returned, as is, with a payment request.
3. **Certificate.** The gateway's signature key certificate.

With the authorization from the gateway, the merchant can provide the goods or service to the customer.

Payment Capture

To obtain payment, the merchant engages the payment gateway in a payment capture transaction, consisting of a capture request and a capture response message.

For the **Capture Request** message, the merchant generates, signs, and encrypts a capture request block, which includes the payment amount and the transaction ID. The message also includes the encrypted capture token received earlier (in the Authorization Response) for this transaction, as well as the merchant's signature key and key-exchange key certificates.

When the payment gateway receives the capture request message, it decrypts and verifies the capture request block and decrypts and verifies the capture token block. It then checks for consistency between the capture request and capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request causes funds to be transferred to the merchant's account.

The gateway then notifies the merchant of payment in a **Capture Response** message. The message includes a capture response block that the gateway signs and encrypts. The message also includes the gateway's signature key certificate. The merchant software stores the capture response to be used for reconciliation with payment received from the acquirer.

UNIT-VII: NETWORK MANAGEMENT SECURITY

Basic concepts of SNMP, SNMPv1 Community facility and SNMPv3. System Security: Intruders-Intrusion techniques, Intrusion Detection, Password Management, Botnets. Malicious Software: Viruses and related threats, Virus Counter Measures, Distributed Denial of Service Attacks.

Network Management Security

Outline:

Basic Concepts of SNMP
SNMPv1 Community Facility
SNMPv3

Basic Concepts of SNMP:

An integrated collection of tools for network monitoring and control.

- Single operator interface
- Minimal amount of separate equipment. Software and network communications capability built into the existing equipment

SNMP key elements:

- Management station
- Management agent
- Management information base
- Network Management protocol

○ Get, Set and Notify

A *management station* is typically a standalone device or a shared system.

In either case, the management station serves as the interface for the human network manager into the network management system.

The management station will have, at minimum:

- A set of management applications for data analysis, fault recovery, and so on.
- An interface by which the network manager may monitor and control the network.
- A protocol by which the management station and managed entities exchange control and management information.
- A database of information extracted from the management databases of all the managed entities in the network.

The other active element in the network management system is the *management agent*.

Key platforms, such as hosts, bridges, routers, and hubs, may be equipped with SNMP agent software so that they may be managed from a management station.

The management agent responds to requests for information from a management station, responds to requests for actions from the management station.

In order to manage the resources in a network, these resources are represented as objects. Each object is, essentially, a data variable that represents one aspect of the managed system.

The collection of objects is referred to as a *management information base (MIB)*.

A management station performs the monitoring function by retrieving the value of MIB objects.

A management station can cause an action to take place at an agent or can change the configuration settings of an agent by modifying the value of specific variables.

The management station and agents are linked by a *network management protocol*, which includes the following key capabilities:

- ▢ *Get*: enables the management station to retrieve the values of objects at the agent
- ▢ *Set*: enables the management station to set the values of objects at the agent
- ▢ *Trap*: enables an agent to notify the management station of significant events

Network Management Protocol Architecture:

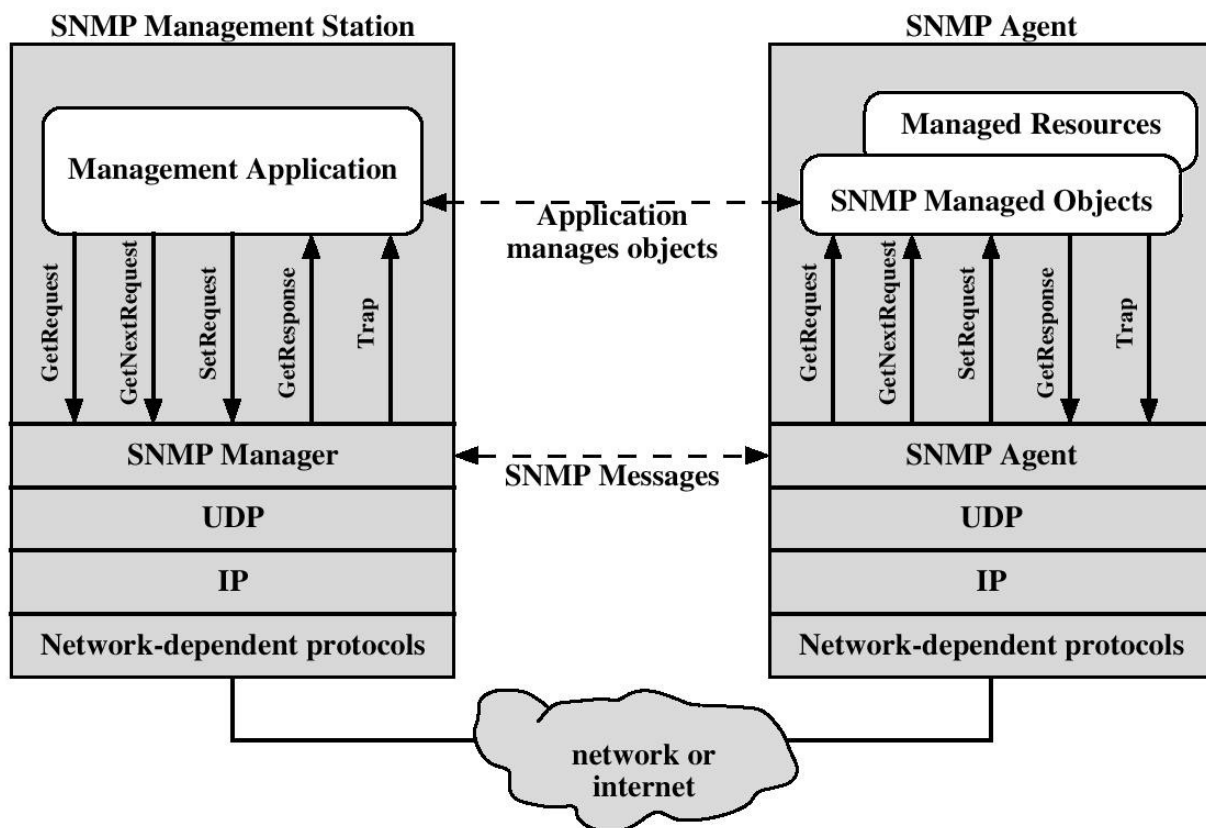
SNMP was designed to be an application-level protocol that is part of the TCP/IP protocol suite.

SNMP typically operates over the user datagram protocol (UDP), although it may also operate over TCP.

For a standalone management station, a manager process controls access to the central MIB at the management station and provides an interface to the network manager.

The manager process achieves network management by using SNMP, which is implemented on top of UDP, IP, and the relevant network-dependent protocols (e.g., Ethernet, FDDI, X.25).

Protocol context of SNMP:



From a management station, three types of SNMP messages are issued on behalf of a management application: GetRequest, GetNextRequest, and SetRequest.

The first two are variations of the get function.

All three messages are acknowledged by the agent in the form of a GetResponse message, which is passed up to the management application.

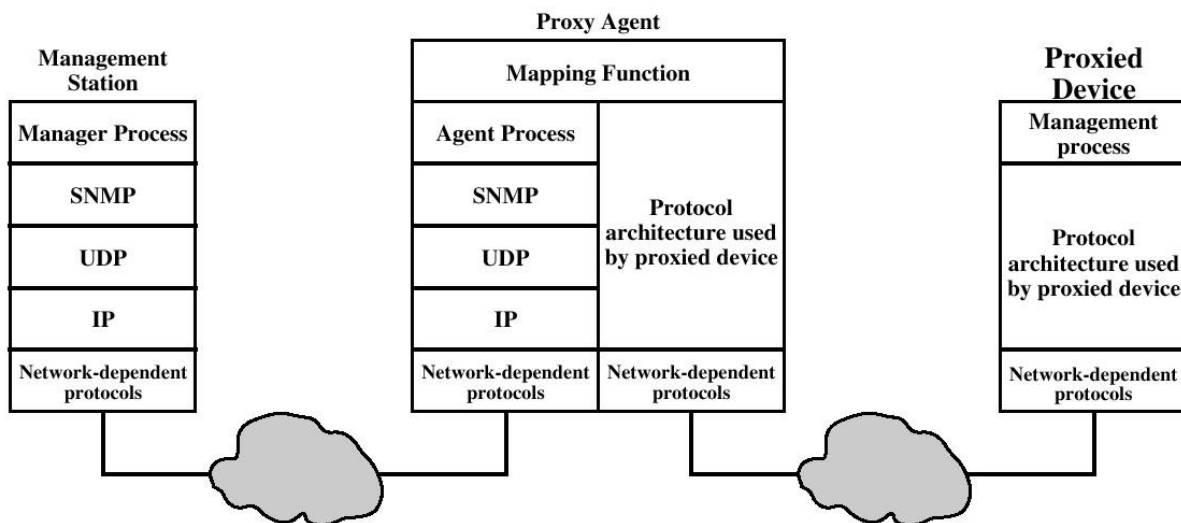
In addition, an agent may issue a trap message in response to an event that affects the MIB and the underlying managed resources.

SNMP relies on UDP, which is a connectionless protocol, and SNMP is itself connectionless.

No ongoing connections are maintained between a management station and its agents.

Instead, each exchange is a separate transaction between a management station and an agent.

Proxy Configuration:



SNMPv2:

SNMPv1 has the following deficiencies:

- Lack of support for distributed network management
- Functional deficiencies
- Security deficiencies

The first two categories of deficiencies are addressed in SNMPv2.

The security deficiencies are addressed in SNMPv3.

Distributed Network Management:

In a traditional centralized network management scheme, one host in the configuration has the role of a network management station.

There may be two or other management stations in a backup role.

The remainder of the devices on the network contain agent software and an MIB to allow monitoring and control from network station.

As networks grow in size and traffic load, such as centralized system is unworkable.

In a decentralized network management scheme, there may be multiple top level management stations.(management servers)

Each server directly manage a portion of the total pool of agents.

The intermediate manager plays an agent role to provide information and accept control from a higher-level management server.

This type of architecture spreads the processing burden and reduces total network traffic.

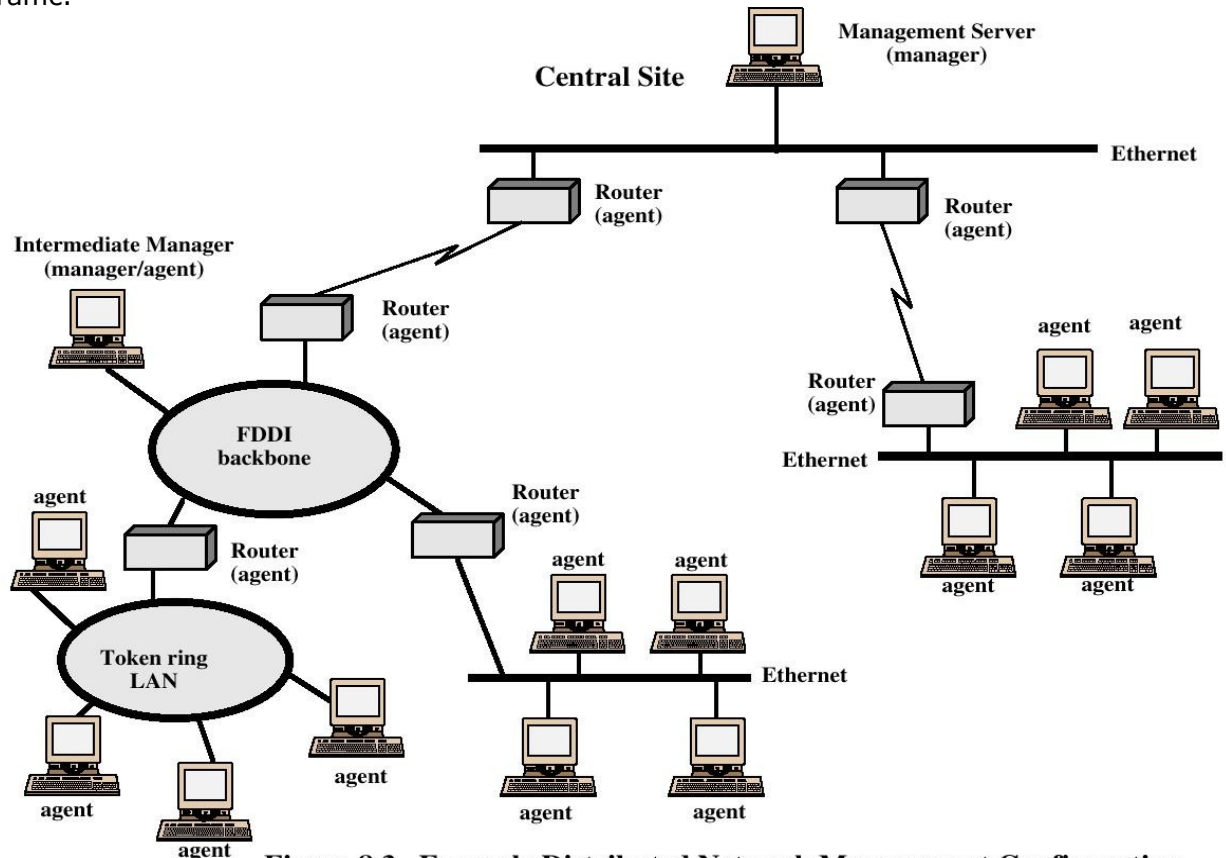


Figure 8.3 Example Distributed Network Management Configuration

SNMP v1 and v2:

Trap – an unsolicited message (reporting an alarm condition)

SNMPv1 is "connectionless" since it utilizes UDP (rather than TCP) as the transport layer protocol.

SNMPv2 allows the use of TCP for "reliable, connection-oriented" service.

Functional Enhancements Comparison of SNMPv1 and SNMPv2

SNMPv1 PDU	SNMPv2 PDU	Direction	Description
GetRequest	GetRequest	Manager to agent	Request value for each listed object
GetRequest	GetRequest	Manager to agent	Request next value for each listed object
-----	GetBulkRequest	Manager to agent	Request multiple values
SetRequest	SetRequest	Manager to agent	Set value for each listed object
-----	InformRequest	Manager to manager	Transmit unsolicited information
GetResponse	Response	Agent to manager or Manager to manager(SNMPv2)	Respond to manager request
Trap	SNMPv2-Trap	Agent to manager	Transmit unsolicited information

SNMPv1 Community Facility:

SNMP Community – Relationship between an SNMP agent and SNMP managers that define authentication, access control, and proxy characteristics.

Three aspect of agent control:

- Authentication service
 - 0** The agent may wish to limit access to the MIB to authorized managers.
- Access policy
 - 1** The agent may wish to gie different access privileges to different managers.
- Proxy service
 - 2** The agent may act as proxy to other agents

The community concept is defined at the agent.

The agent establishes one community for each desired combination of authentication, access control, and proxy characteristics.

Each community is given a unique community name.

The managers with that are provided with and must employ the community name in all get and set operations.

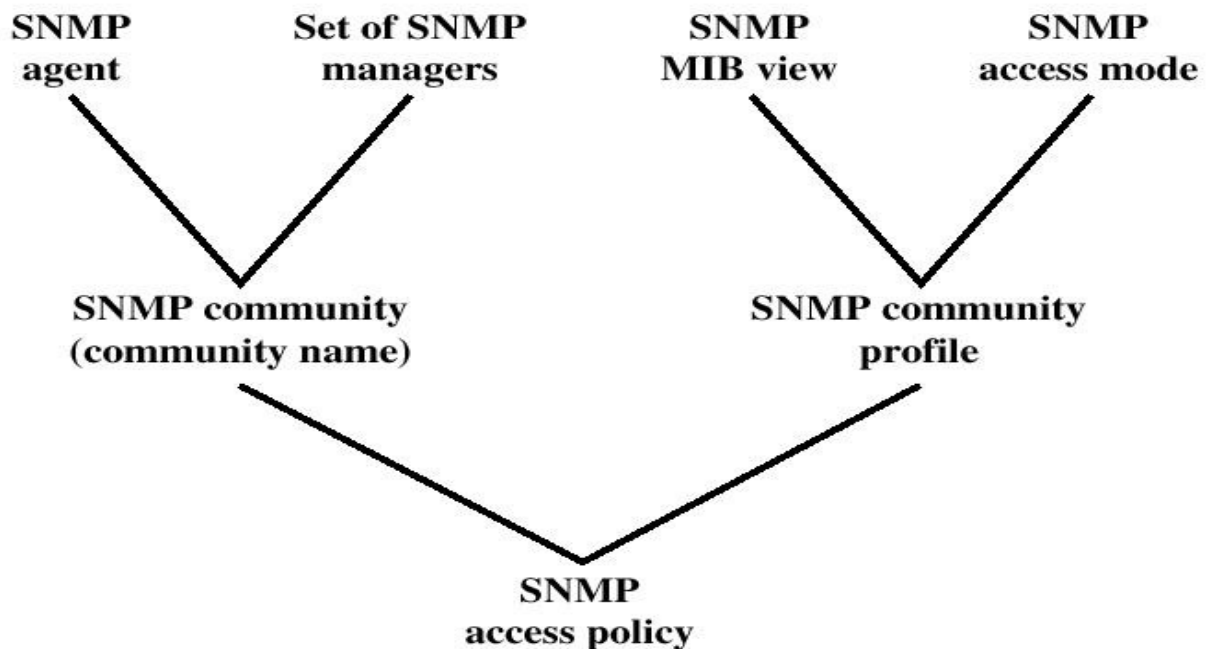
Authentication Service

- Every message(get and put request) from a manager to an agent includes a community name.
- This name functions as a password, and the message is assumed to be authentic if the sender knows the password.

Access policy

- By defining a community, an agent limits access to its MIB to a selected set of managers.
- By the use of more than one community, the agent can provide different categories of MIB access to different managers.

SNMPv1 Administrative Concepts:



SNMPv3:

In this section we discuss

- Introduction to the basic SNMP architecture.
- Confidentiality and authentication facilities provided by the SNMPv3 User security model.
- Access control and view based access control

Intruders

Three classes of intruders (hackers or crackers):

- Masquerader
 - An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account
- Mifeasor
 - A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges
- Clandestine user
 - An individual who seizes supervisory control of the system and uses this control to avoid auditing and access controls or to suppress audit collection

Intrusion Techniques:

System maintain a file that associates a password with each authorized user.

Password file can be protected with:

- One-way encryption**
- Access Control**

One-way encryption: The system stores only an encrypted form of the user's password. When the user presents a password, the system encrypts 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 encryption 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.

Intrusion Techniques:

Techniques for guessing passwords:

- Try default passwords.
- Try all short words, 1 to 3 characters long.
- Try all the words in an electronic dictionary(60,000).
- Collect information about the user's hobbies, family names, birthday, etc.
- Try user's phone number, social security number, street address, etc.
- Try all license plate numbers (MUP103).
- Use a Trojan horse
- Tap the line between a remote user and the host system.

Two principal countermeasures: prevention and detection. 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. Detection is concerned with learning of an attack, either before or after its success.

Intusion Detection

The intruder can be identified and ejected from the system.

An effective intrusion detection can prevent intrusions.

Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Profiles of Behavior of Intruders and Authorized Users

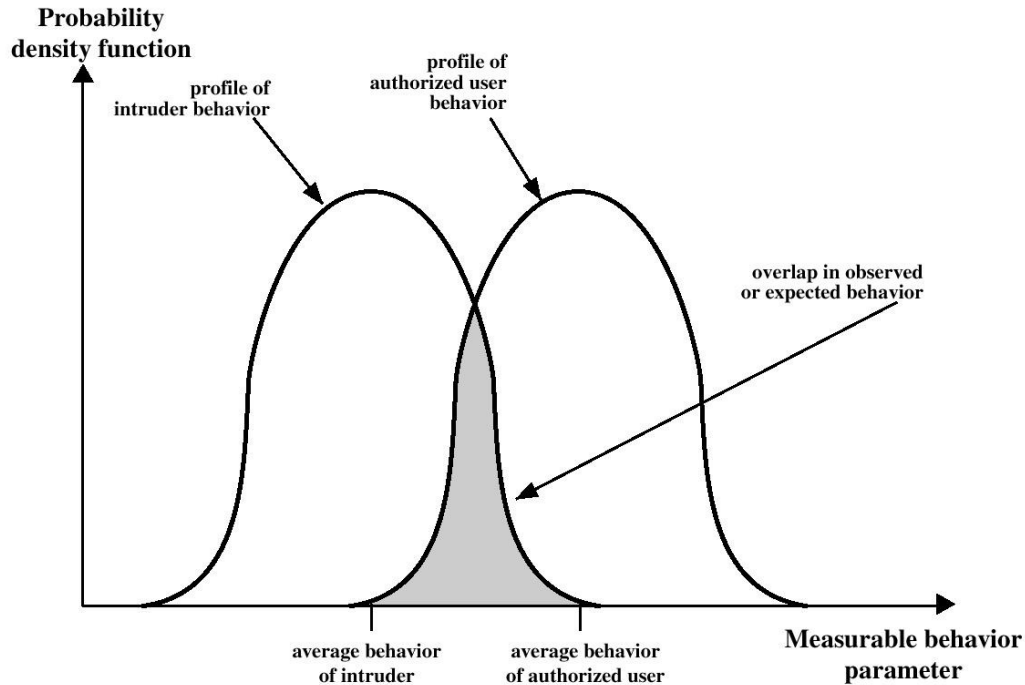


Figure above suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of 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.

Following approaches to intrusion detection:

Statistical anomaly detection

- Threshold detection

- This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.

- Profile based

- A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.

Rule based detection

- Anomaly detection

- Rules are developed to detect deviation from previous usage patterns.

- Penetration identification

- An expert system approach that searches for suspicious 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 must be maintained by users 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 is one developed by Dorothy Denning . Each audit record contains the following fields:

Subject: Initiators of actions.

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.

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.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file. Consider the command.

COPY GAME.EXE TO <Library>GAME.EXE

Smith	execute	<Library>COPY.EXE	0	CPU = 00002	11058721678
-------	---------	-------------------	---	-------------	-------------

Smith	read	<Smith>GAME.EXE	0	RECORDS = 0	11058721679
-------	------	-----------------	---	-------------	-------------

Smith	execute	<Library>COPY.EXE	write-viol	RECORDS = 0	11058721680
-------	---------	-------------------	------------	-------------	-------------

The decomposition of a user operation into elementary actions has three advantages:

Because objects are the protectable entities in a system, the use of elementary actions enables an audit of all behavior affecting an object. Thus, the system can detect attempted subversions of access controls (by noting an abnormality in the number of exception conditions returned) and can detect successful subversions by noting an abnormality in the set of objects accessible to the subject.

Single-object, single-action audit records simplify the model and the implementation.

Because of the simple, uniform structure of the detection-specific audit records, it may be relatively easy to obtain this information or at least part of it by a straightforward mapping from existing native audit records to the detection-specific audit records.

Statistical Anomaly Detection

Statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. Threshold detection involves counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function 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. Examples of metrics that are useful for profile-based intrusion detection are the following:

Counter: A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time.

Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.

Gauge: A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity.

Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.

Interval timer: The length of time between two related events. An example is the length of time between successive logins to an account.

Resource utilization: Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. lists the following approaches that may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series
- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period.

This gives a reflection of the average behavior and its variability.

The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures.

But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

The main advantage of the use of statistical profiles is that a prior knowledge of security flaws is not required. The detector program learns what is "normal" behavior and then looks for deviations. The approach is not based on system-dependent characteristics and vulnerabilities. Thus, it should be readily portable among a variety of systems.

Rule-Based Intrusion Detection

Rule-based anomaly detection

- Is similar in terms of its approach and strengths to statistical anomaly detection.
- Historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns.
- Rules may represents past behavior patterns of users, programs, terminals and so on.
- Current behavior is observed and matched against the historically observed pattern of behavior.
- It does not require knowledge of security vulnerabilities with in the system.
- In order for this approach to be effective, a large data base of rules will be needed.

Rule based penetration identification

- Use the use of rules for identifying known penetrations or penetrations that would exploit known weakness.

- These rules are specific to machine and operating systems.
- Rules are generated by experts.
- The strength of this approach is depends on the skill of those involved in setting up the rules.

Example rules:

- Users should not read files in other users personal directories.
- Users must not write other users files.
- Users who log in after hours often access the same files they used earlier.
- Users do not generally open disk devices directly but rely on high level operating system utilities.
- Users should not be logged in more than once to the same system.
- Users do not make copies of system programs.

The Base-Rate Fallacy:

To be of practical use, an intrusion detection system should detect a substantial percentage of intrusions while keeping the false alarm rate at an acceptable level.

If only a modest percentage of actual intrusions are detected, the system provides a false sense of security. On the other hand, if the system frequently triggers an alert when there is no intrusion (a false alarm), then either system managers will begin to ignore the alarms, or much time will be wasted analyzing the false alarms.

Unfortunately, because of the nature of the probabilities involved, it is very difficult to meet the standard of high rate of detections with a low rate of false alarms.

In general, if the actual numbers of intrusions is low compared to the number of legitimate uses of a system, then the false alarm rate will be high unless the test is extremely discriminating.

A study of existing intrusion detection systems indicated that current systems have not overcome the problem of the base-rate fallacy.



Distributed Intrusion Detection:

The typical organization, 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.

Porras points out the following major issues in the design of a distributed intrusion detection system :

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.

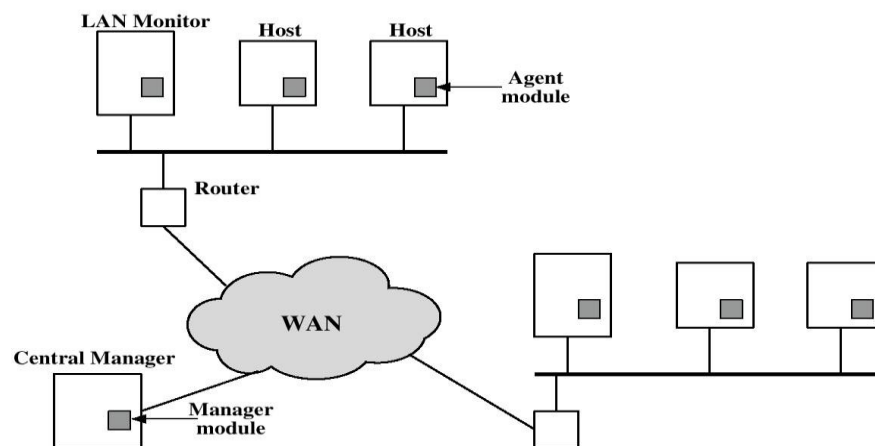
A good example of a distributed intrusion detection system is one developed at the University of California at Davis [HEBE92, SNAP91]. Figure 9.2 shows the overall architecture, which consists of three main components:

Host agent module: An audit collection module operating as a background process on a monitored system. Its purpose is to collect data on security related events on the host and transmit these to the central manager.

LAN monitor agent module: Operates in the same fashion as a host agent module except that it analyzes LAN traffic and reports the results to the central manager.

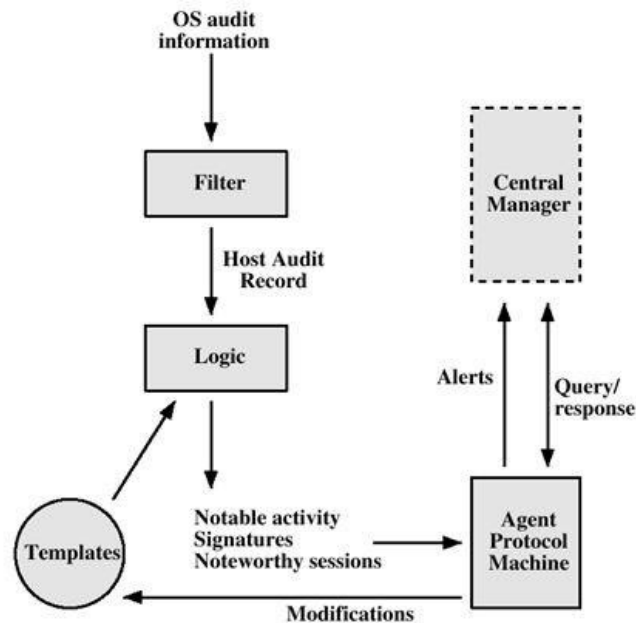
Central manager module: Receives reports from LAN monitor and host agents and processes and correlates these reports to detect intrusion.

Fig: 9.2 Architecture for distributed Intrusion Detection



The scheme is designed to be independent of any operating system or system auditing implementation. Figure 9.3 shows the general approach that is taken. The agent captures each audit record produced by the native audit collection system. A filter is applied that retains only those records that are of security interest. These records are then reformatted into a standardized format referred to as the host audit record (HAR). Next, a template-driven logic module analyzes the records for suspicious activity. At the lowest level, the agent scans for notable events that are of interest independent of any past events. Examples include failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the agent looks for sequences of events, such as known attack patterns (signatures). Finally, the agent looks for anomalous behavior of an individual user based on a historical profile of that user, such as number of programs executed, number of files accessed, and the like.

Fig: 9.3: Agent architecture



Honey pots

A relatively recent innovation in intrusion detection technology is the honeypot. Honeypots are decoy systems that are designed to lure a potential attacker away from critical systems. Honeypots are designed to

- divert an attacker from accessing critical systems
- collect information about the attacker's activity
- encourage the attacker to stay on the system long enough for administrators to respond

These systems are filled with fabricated information designed to appear valuable but that a legitimate user of the system wouldn't access. Thus, any access to the honeypot is suspect. The system is instrumented with sensitive monitors and event loggers that detect these accesses and collect information about the attacker's activities. Because any attack against the honeypot is made to seem successful, administrators have time to mobilize and log and track the attacker without ever exposing productive systems. Once hackers are within the network, administrators can observe their behavior in detail and figure out defenses.

Intrusion Detection Exchange Format

To facilitate the development of distributed intrusion detection systems that can function across a wide range of platforms and environments, standards are needed to support interoperability. Such standards are the focus of the IETF Intrusion Detection Working Group. The purpose of the working group is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to management systems that may need to interact with them. The outputs of this working group include:

- A requirements document, which describes the high-level functional requirements for communication between intrusion detection systems and requirements for communication between intrusion detection systems and with management systems, including the rationale for those requirements. Scenarios will be used to illustrate the requirements.

- A common intrusion language specification, which describes data formats that satisfy the requirements.

- A framework document, which identifies existing protocols best used for communication

between intrusion detection systems, and describes how the devised data formats relate to them.

As of this writing, all of these documents are in an Internet-draft document stage.

PASSWORD MANAGEMENT:

Password Protection

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.

- 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 (Figure 9.4a). 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 [WAGN00].

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.

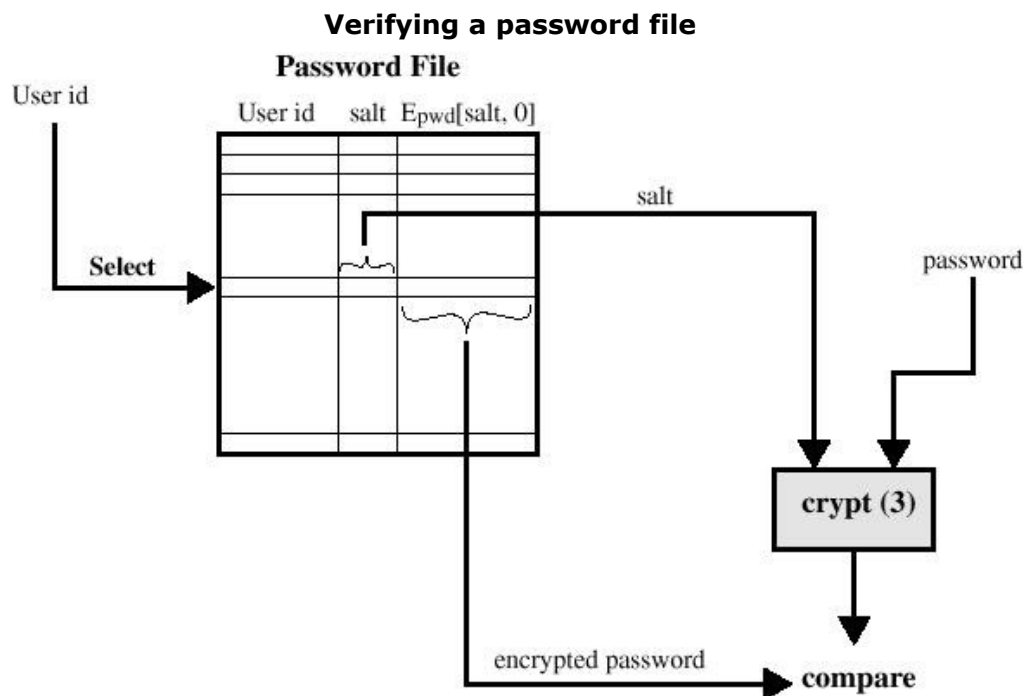
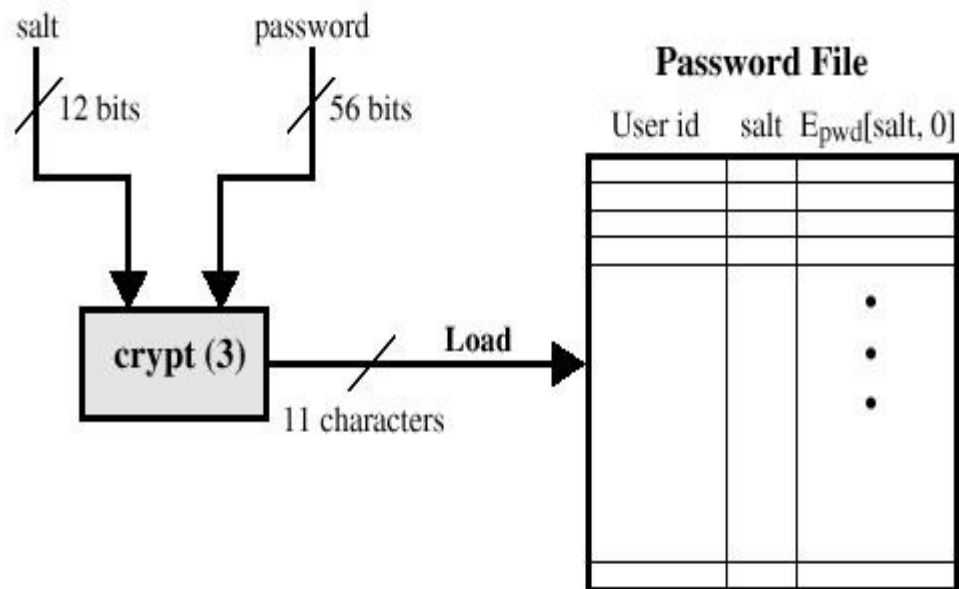
- It effectively increases the length of the password without requiring the user to remember two additional characters. Hence, the number of possible passwords is increased by a factor of 4096, increasing the difficulty of guessing a password.

- 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.

UNIX Password Scheme:

Loading a new password



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. [SPA92a] points out several flaws in this strategy:

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

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. 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.

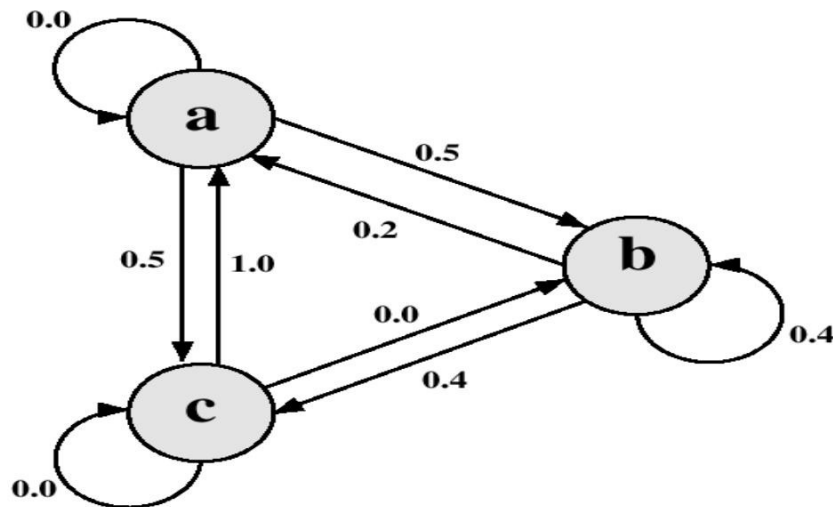
The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based 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.

Two techniques for developing an effective and efficient proactive password checker that is based on rejecting words on a list show promise. One of these develops a Markov model for the generation of guessable passwords [DAVI93]. Figure 9.5 shows a simplified version of such a model. This model shows a language consisting of an alphabet of three characters. The state of the system at any time is the identity of the most recent letter. The value on the transition from one state to another represents the probability that one letter follows another. Thus, the probability that the next letter is b, given that the current letter is a, is 0.5.

In general, a Markov model is a quadruple $[m, A, T, k]$, where m is the number of states in the model, A is the state space, T is the matrix of transition probabilities, and k is the order of the model. For a k th-order model, the probability of making a transition to a particular letter depends on the previous k letters that have been generated. Figure 9.5 shows a simple first-order model.

The authors report on the development and use of a second-order model. To begin, a dictionary of guessable passwords is constructed. Then the transition matrix is calculated as follows:



$M = \{3, \{a, b, c\}, T, 1\}$ where

$$T = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.2 & 0.4 & 0.4 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

e.g., string probably from this language: abbcacaba

e.g., string probably not from this language: aaccbbaaa

Determine the frequency matrix f , where $f(i, j, k)$ is the number of occurrences of the trigram consisting of the i th, j th and k th character.

For each bigram ij , calculate $f(i, j,)$ as the total number of trigrams beginning with ij .

Compute the entries of T as follows:

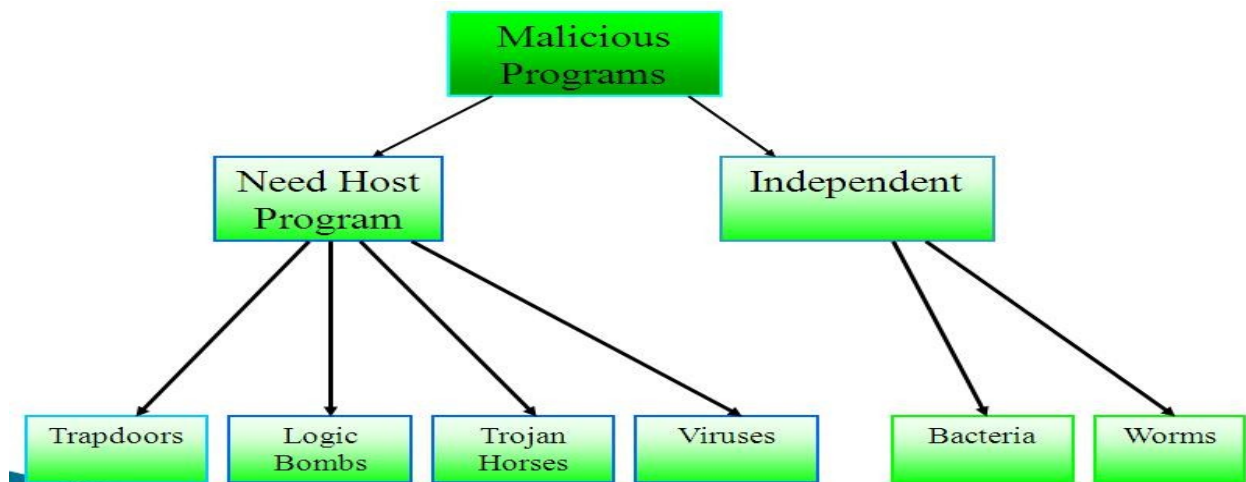
$$T(i, j, k) = \frac{f(i, j, k)}{f(i, j, \infty)}$$

Viruses and "Malicious Programs"

Computer "Viruses" and related programs have the ability to replicate themselves on an ever increasing number of computers. They originally spread by people sharing floppy disks. Now they spread primarily over the Internet (a "Worm").

Other "Malicious Programs" may be installed by hand on a single machine. They may also be built into widely distributed commercial software packages. These are very hard to detect before the payload activates (Trojan Horses, Trap Doors, and Logic Bombs).

Taxonomy of Malicious Programs



Definitions:

Virus - code that copies itself into other programs.

A "**Bacteria**" replicates until it fills all disk space, or CPU cycles.

Payload - harmful things the malicious program does, after it has had time to spread.

Worm - a program that replicates itself across the network (usually riding on email messages or attached documents (e.g., macro viruses).

Trojan Horse - instructions in an otherwise good program that cause bad things to happen (sending your data or password to an attacker over the net).

Logic Bomb - malicious code that activates on an event (e.g., date).

Trap Door (or Back Door) - undocumented entry point written into code for debugging that can allow unwanted users.

Easter Egg - extraneous code that does something "cool." A way for programmers to show that they control the product.

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 virus automatically.

Spammer programs: used to send large volumes of unwanted traffic.

Flooders: used to attack networked computer with a large volumes of traffic to carry out a denial of service attack.

Keyloggers: captures key strokes on a compromised system.

Root kit: set of hacker tools used after attackers has broken into a computer system and gained root-level access.

Zombie: program activated on an infected machine that is activated to launch attacks on other machines.

TYPES OF MALICIOUS SOFTWARE:

Backdoor
Logic Bomb
Trojan Horses

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, changes 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 program or application. 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 of the program 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 [THOM84]. 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.

The Trojan horse was implanted in a graphics routine offered on an electronic bulletin board system. Trojan horses fit into one of three models:

- **Continuing to perform the function of the original program and additionally performing a separate malicious activity**

Continuing to perform the function of the original program but modifying the function to perform malicious activity (e.g., a Trojan horse version of a login program that collects passwords) or to disguise other malicious activity (e.g., a Trojan horse version of a process listing program that does not display certain processes that are malicious)

Performing a malicious function that completely replaces the function of the original program.

VIRUSES:

The Nature of Viruses:

A virus is a piece of software that can “infect” other programs by modifying them. A virus can do anything that other programs do. The difference is that a virus 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 that is allowed by the privileges of the current user. A computer virus has three parts:

Infection mechanism: The means by which a virus spreads, enabling it to replicate. The mechanism is also referred to as the **infection vector**.

Trigger: The event or condition that determines when the payload is activated or delivered.

Payload: What the virus does, besides spreading. The payload may involve damage or may involve benign but noticeable activity.

Virus Phases in its lifecycle:

During its lifetime, a typical virus goes through the following four phases:

Dormant phase - the virus is idle

Propagation phase - the virus places an identical copy of itself into other programs

Triggering phase – the virus is activated to perform the function for which it was intended

Execution phase – the function is performed

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.

```

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:
}

```

The 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 may first seek out uninfected executable files and infect 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 an uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. Figure 10.2 shows in general terms the logic required. The key lines in this virus are numbered.

```

program CV :=
{goto main;
 01234567;

subroutine infect-executable :=
  {loop:
    file := get-random-executable-file;
    if (first-line-of-file = 01234567) then goto loop;
  (1) compress file;
  (2) prepend CV to file;
  }

main: main-program :=
  {if ask-permission then infect-executable;
  (3) uncompress rest-of-file;
  (4) run uncompressed file;
  }
    
```

Figure 10.2 Logic for a Compression Virus

Figure 10.3 illustrates the operation. We assume that program P₁ is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file P₂ that is found, the virus first compresses that file to produce P'₂, which is shorter than the original program by the size of the virus.
2. A copy of the virus is prepended to the compressed program.
3. The compressed version of the original infected program, P'₁, is uncompressed.
4. The uncompressed original program is executed.

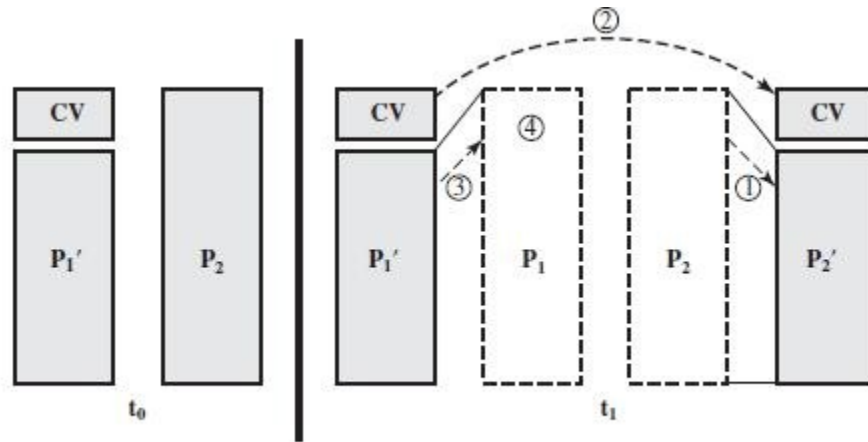


Figure 10.3 A Compression Virus

INITIAL INFECTION Once a virus has gained entry to a system by infecting a single program, it is in a position to potentially infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

Types of Viruses:

Parasitic Virus - attaches itself to executable files as part of their code. Runs whenever the host program runs.

Memory-resident Virus - Lodges in main memory as part of the residual operating system.

Boot Sector Virus - infects the boot sector of a disk, and spreads when the operating system boots up (original DOS viruses).

Stealth Virus - explicitly designed to hide from Virus Scanning programs.

Polymorphic Virus - mutates with every new host to prevent signature detection.

Metamorphic Virus: changes with every infection. May change their behavior as well as their appearance.

Macro Viruses

Microsoft Office applications allow "macros" to be part of the document. The macro could run whenever the document is opened, or when a certain command is selected (Save File).

Platform independent.

Infect documents, delete files, generate email and edit letters.

Easily spread, a very common method is by electronic mail.

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

The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.

The virus does local damage on the user's system.

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 it is activated (either by opening an e-mail attachment or by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for

antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

Worms:

A worm is a program that can replicate itself and send copies from computer to computer across network connections.

To replicate itself, a network worm uses some sort of network vehicle. Example includes 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 remote system as a user and then uses commands to copy itself from one system to the other.

The Morris worm:

Released onto the internet by Robert Morris in 1998.

Designed to spread on Unix systems.

It attempted to log onto a remote host as a legitimate user.

In this method, the worm first attempted to crack the local password file, and then used 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

- Each user's account name and simple permutations of it.
- A list of 432 built-in passwords that Morris thought to be likely candidates.
- All the words in the local system directory

If exploited a bug in the finger protocol, which reports the whereabouts of a remote user.

It exploited a trapdoor in the debug option of the remote process that receives and send mail.

Antivirus Approaches :

The ideal solution to the threat of viruses is prevention.

The next best approach is:

- **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 virus from the infected program and restore it to its original state.

Antivirus Approaches:

1st Generation, Scanners: searched files for any of a library of known virus "signatures." Checked executable files for length changes.

2nd Generation, Heuristic Scanners: looks for more general signs than specific signatures (code segments common to many viruses). Checked files for checksum or hash changes.

3rd Generation, Activity Traps: stay resident in memory and look for certain patterns of software behavior (e.g., scanning files).

4th Generation, Full Featured: combine the best of the techniques above.

Advanced Antivirus Techniques Generic Decryption (GD)

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses while maintaining fast scanning speeds.

a GD scanner, contains the following elements:

- ▮ **CPU Emulator**
 - 0 A software based virtual computer.
 - 1 Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor.
 - 2 So that underlying is unaffected.
- ▮ **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 of degraded system performance.

Digital Immune System

Developed by IBM

To identify Internet-based virus propagation.

Two major trends in internet technology will have an increase impact on the rate of virus propagation.

- ▮ Integrated mail systems:
 - 0 Ex: LOTUS Notes, Microsoft Outlook.
- ▮ Mobile-program systems:
 - 0 Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.

Figure 10.4 illustrates the typical steps in digital immune system operation:

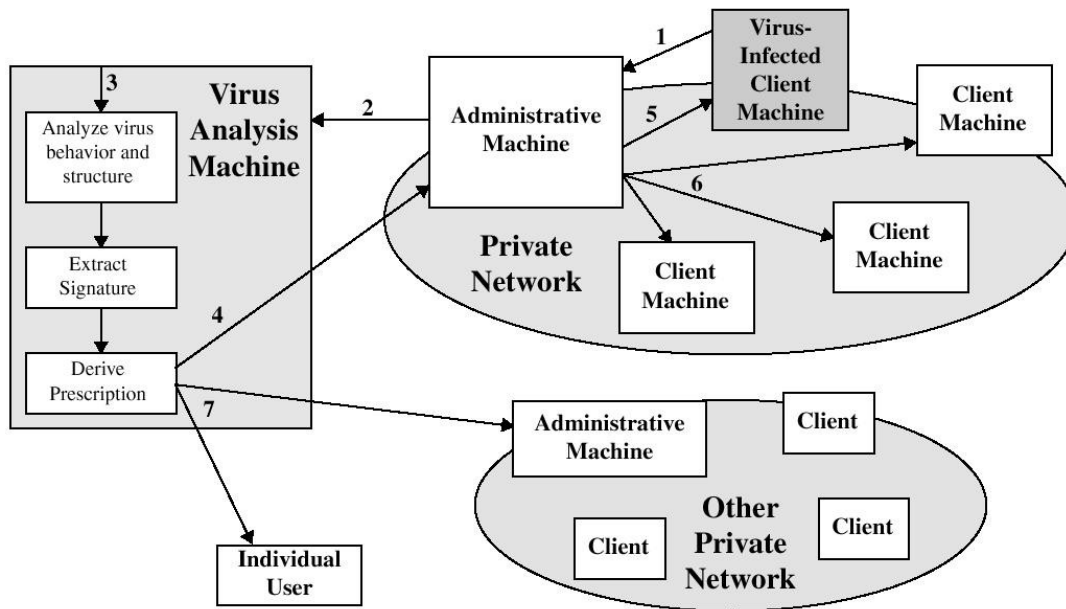


Figure 9.11 Digital Immune System

A monitoring program on each PC identifies if virus is present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within organization.

The administrative machine encrypts the sample to send it to a central virus analysis machine.

This machine creates an environment in which the infected program can be safely run for analysis. The virus analysis machine then produces a prescription for identifying and removing the virus.

The resulting prescription is sent back to the administrative machine.

The administrative machine forwards the prescription to the infected client.

The prescription is also forwarded to other clients in the organization.

Subscribers around the world receive regular antivirus updates that protect them from the new virus.

The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

BEHAVIOR-BLOCKING SOFTWARE:

Behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real-time for malicious actions.

The behavior blocking software then blocks potentially malicious actions before they have a chance to affect the system.

Monitored behaviors can include

- Attempts to open, view, delete, and/or modify files;
- Attempts to format disk drives and other unrecoverable disk operations;
- Modifications to the logic of executable files or macros;
- Modification of critical system settings, such as start-up settings;

Scripting of e-mail and instant messaging clients to send executable content; and
Initiation of network communications.

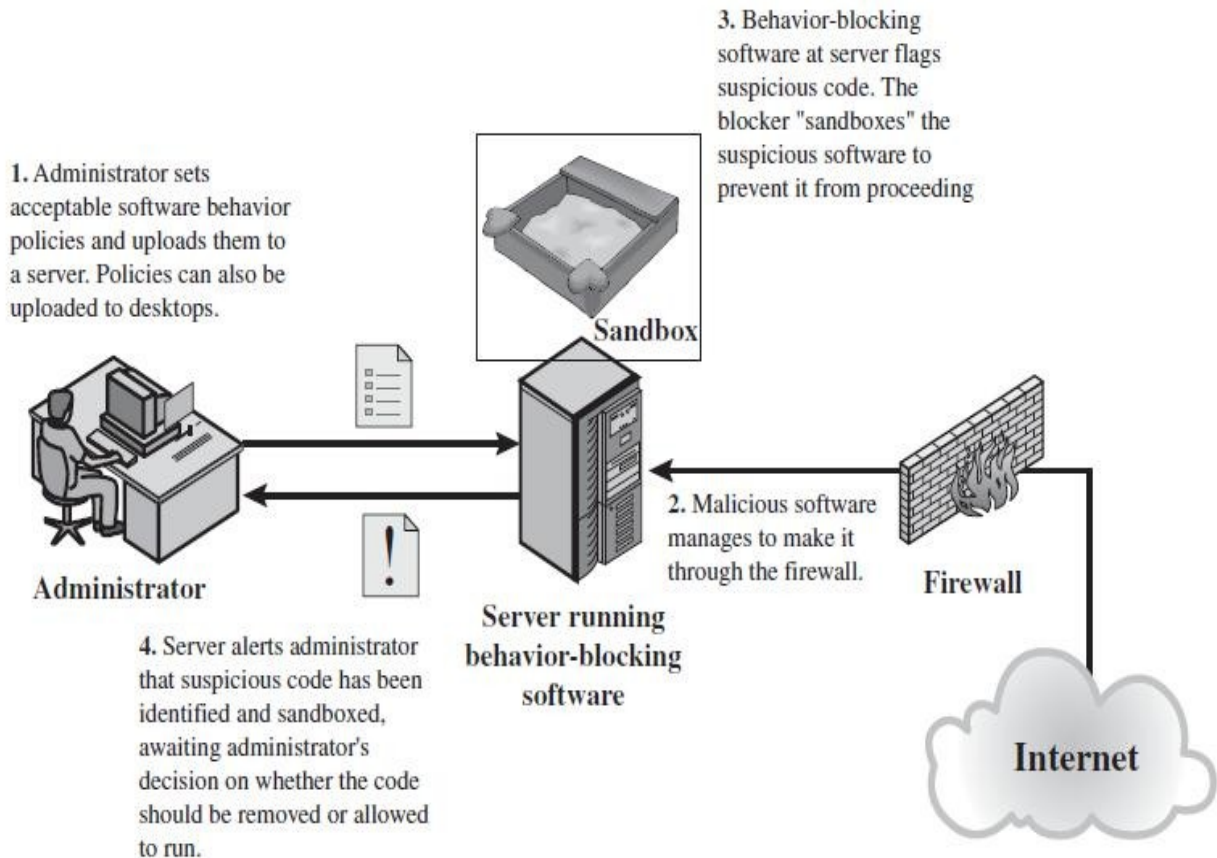


Figure 10.5 Behavior-Blocking Software Operation

Behavior blocking alone has limitations.

Because the malicious code must run on the target machine before all its behaviors can be identified, it can cause harm before it has been detected and blocked.

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.

Multi-exploit: 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 bots.

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.

Mobile Phone Worms:

Worms first appeared on mobile phones in 2004.

These worms communicate through Bluetooth wireless connections or via the multimedia messaging service (MMS).

The target is the smartphone, which is a mobile phone that permits users to install software applications from sources other than the cellular network operator.

Mobile phone malware can completely disable the phone, delete data on the phone, or force the device to send costly messages to premium-priced numbers.

An example of a mobile phone worm is CommWarrior, which was launched in 2005.

DISTRIBUTED DENIAL OF SERVICE ATTACKS:

DDoS attacks make computer systems inaccessible by flooding servers, networks, or even end user systems with useless traffic so that legitimate users can no longer gain access to those resources.

In a typical DDoS attack, a large number of compromised hosts are amassed to send useless packets.

In a DDoS attack, an attacker is able to recruit a number of hosts throughout the Internet to simultaneously or in a coordinated fashion launch an attack upon the target.

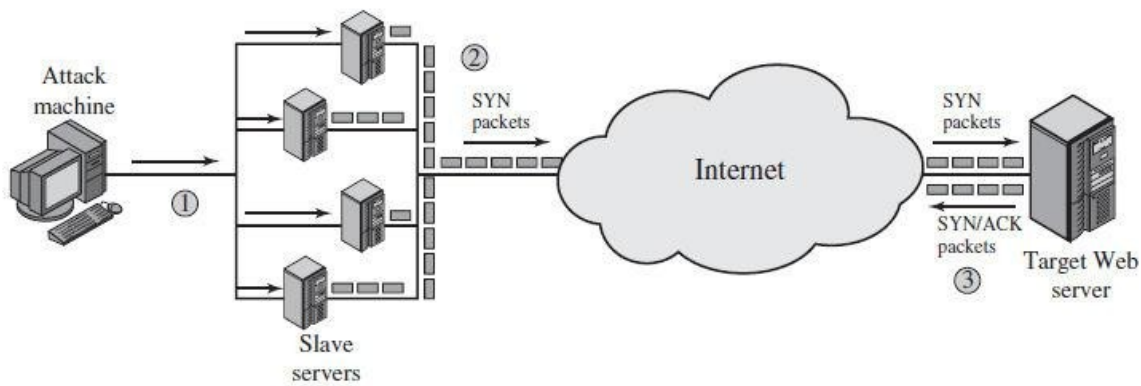
DDoS Attack Description:

One way to classify DDoS attacks is in terms of the type of resource that is consumed.

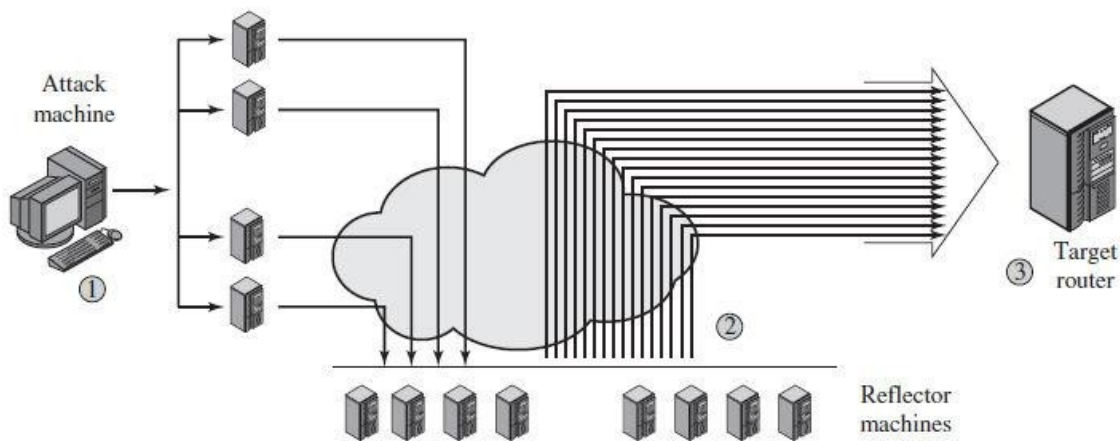
The resource consumed is either an internal host resource on the target system or data transmission capacity in the local network to which the target is attacked.

A simple example of an internal resource attack is the SYN flood attack. the steps involved.

- The attacker takes control of multiple hosts over the Internet, instructing them to contact the target Web server.
- The slave hosts begin sending TCP/IP SYN (synchronize/initialization) packets, with erroneous return IP address information, to the target.
- Each SYN packet is a request to open a TCP connection. For each such packet, the Web server responds with a SYN/ACK (synchronize/acknowledge) packet, trying to establish a TCP connection with a TCP entity at a spurious IP address.



(a) Distributed SYN flood attack



(a) Distributed ICMP attack

Figure 10.9 Examples of Simple DDoS Attacks

Other ways of DDOS:

In many systems, a limited number of data structures are available to hold process information (process identifiers, process table entries, process slots, etc.). An intruder may be able to consume these data structures by writing a simple program or script that does nothing but repeatedly create copies of itself.

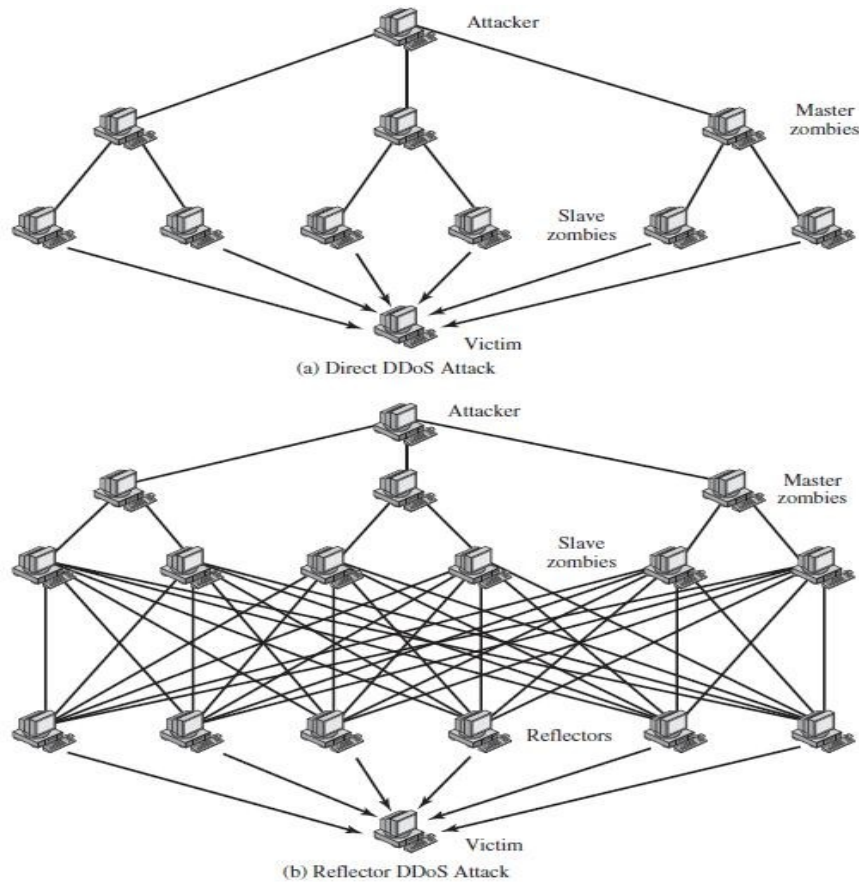
An intruder may also attempt to consume disk space in other ways, including

- generating excessive numbers of mail messages
- intentionally generating errors that must be logged
- placing files in anonymous ftp areas or network-shared areas

an example of an attack that consumes data transmission resources. The following steps are involved:

- The attacker takes control of multiple hosts over the Internet, instructing them to send ICMP ECHO packets with the target's spoofed IP address to a group of hosts that act as reflectors, as described subsequently.
- Nodes at the bounce site receive multiple spoofed requests and respond by sending echo reply packets to the target site.
- The target's router is flooded with packets from the bounce site, leaving no data transmission capacity for legitimate traffic.

Another way to classify DDoS attacks is as either **direct** or **reflector** DDoS attacks.



Constructing the Attack Network:

The essential ingredients in this phase of the attack are the following:

- Software that can carry out the DDoS attack.
- A vulnerability in a large number of systems.
- A strategy for locating vulnerable machines, a process known as scanning.

lists the following types of scanning strategies:

Random:

- Each compromised host probes random addresses in the IP address space, using a different seed.
- This technique produces a high volume of Internet traffic, which may cause generalized disruption even before the actual attack is launched.

Hit-List:

- The attacker first compiles a long list of potential vulnerable machines.
- This can be a slow process done over a long period to avoid detection that an attack is underway.
- Once the list is compiled, the attacker begins infecting machines on the list.
- Each infected machine is provided with a portion of the list to scan.
- This strategy results in a very short scanning period, which may make it difficult to detect that infection is taking place.

Topological:

- This method uses information contained on an infected victim machine to find more hosts to scan.

Local subnet:

If a host can be infected behind a firewall, that host then looks for targets in its own local network.

The host uses the subnet address structure to find other hosts that would otherwise be by the firewall.

DDoS Countermeasures:

Attack prevention and preemption (before the attack):

- Techniques include enforcing policies for resource consumption and providing backup resources available on demand.
- In addition, prevention mechanisms modify systems and protocols on the Internet to reduce the possibility of DDoS attacks.

Attack detection and filtering (during the attack):

- These mechanisms attempt to detect the attack as it begins and respond immediately.
- This minimizes the impact of the attack on the target.
 - 0** Detection involves looking for suspicious patterns of behavior.
 - 1** Response involves filtering out packets likely to be part of the attack.

Attack source traceback and identification (during and after the attack):

- This is an attempt to identify the source of the attack as a first step in preventing future attacks.
- However, this method typically does not yield results fast enough, if at all, to mitigate an ongoing attack.

UNIT-VIII: FIREWALLS

Firewall Design principles, Trusted Systems, Common Criteria for Information Technology Security Evolution.

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

Firewall design principles:

- Centralized data processing system, with a central mainframe supporting a number of directly connected terminals

- Local area networks (LANs) interconnecting PCs and terminals to each other and the mainframe

- Premises network, consisting of a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two.

- Enterprise-wide network, consisting of multiple, geographically distributed premises networks interconnected by a private wide area network (WAN)

- Internet connectivity, in which the various premises networks all hook into the Internet and may or may not also be connected by a private WAN.

FIREWALL CHARACTERISTICS:

Design goals for a firewall:

- 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.

- 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 chapter.

- The firewall itself is immune to penetration. This implies the use of a hardened system with a secured operating system. Trusted computer systems are suitable for hosting a firewall and often required in government applications.

Along with above there are four general techniques that firewalls use to control access and enforce the site's security policy. Originally, firewalls focused primarily on service control, but they have since evolved to provide all four:

- 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, protocol, or 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.

Capabilities of a firewall:

A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks. The use of a single choke point simplifies security

management because security capabilities are consolidated on a single system or set of systems.

A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.

A firewall is a convenient platform for several Internet functions that are not security related. These include a network address translator, which maps local addresses to Internet addresses, and a network management function that audits or logs Internet usage.

A firewall can serve as the platform for IPsec. the firewall can be used to implement virtual private networks.

Firewalls limitations:

The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.

The firewall may not protect fully against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.

An improperly secured wireless LAN may be accessed from outside the organization. An internal firewall that separates portions of an enterprise network cannot guard against wireless communications between local systems on different sides of the internal firewall.

A laptop, PDA, or portable storage device may be used and infected outside the corporate network, and then attached and used internally.

TYPES OF FIREWALLS

A firewall may act as a packet filter. It can operate as a positive filter, allowing to pass only packets that meet specific criteria, or as a negative filter, rejecting any packet that meets certain criteria. Depending on the type of firewall, it may examine one or more protocol headers in each packet, the payload of each packet, or the pattern generated by a sequence of packets.

Three types:

- Packet filtering router
- Application level gateway
- Circuit level gateway

Packet filtering router:

A packet filtering firewall applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet (fig. shown below). The firewall 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 firewall with three or more ports, which interface of the firewall the packet came from or which interface of the firewall 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.

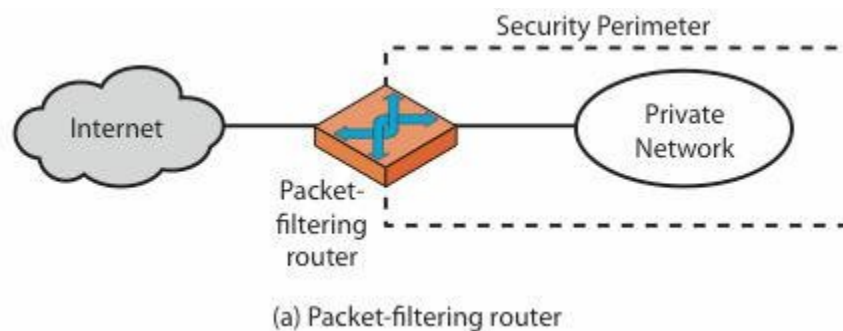


Table 11.1, gives some examples of packet filtering rule sets. In each set, the rules are applied top to bottom. The "*" in a field is a wildcard designator that matches everything. We assume that the default = discard policy is in force.

Inbound mail is allowed (port 25 is for SMTP incoming), but only to a gateway host. However, packets from a particular external host, SPIGOT, are blocked because that host has a history of sending massive files in e-mail messages.

This is an explicit statement of the default policy. All rule sets include this rule implicitly as the last rule.

This rule set is intended to specify that any inside host can send mail to the outside. A TCP packet with a destination port of 25 is routed to the SMTP server on the destination machine. The problem with this rule is that the use of port 25 for SMTP receipt is only a default; an outside machine could be configured to have some other application linked to port 25. As this rule is written, an attacker could gain access to internal machines by sending packets with a TCP source port number of 25.

- This rule set achieves the intended result that was not achieved in C. The rules take advantage of a feature of TCP connections. Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side. Thus, this rule set states that it allows IP packets where the source IP address is one of a list of designated internal hosts and the destination TCP port number is 25. It also allows incoming packets with a source port number of 25 that include the ACK flag in the TCP segment. Note that we explicitly designate source and destination systems to define these rules explicitly.

This rule set is one approach to handling FTP connections. With FTP, two TCP connections are used: a control connection to set up the file transfer and a data connection for the actual file transfer. The data connection uses a different port number that is dynamically assigned for the transfer. Most servers, and hence most attack targets, use low-numbered ports; most outgoing calls tend to use a higher-numbered port, typically above 1023. Thus, this rule set allows

- Packets that originate internally
- Reply packets to a connection initiated by an internal machine
- Packets destined for a high-numbered port on an internal machine

This scheme requires that the systems be configured so that only the appropriate port numbers are in use.

Table 11.1 Packet-Filtering Examples

Rule Set A

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

Rule Set B

action	ourhost	port	theirhost	port	comment
block	*	*	*	*	default

Rule Set C

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connection to their SMTP port

Rule Set D

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		our packets to their SMTP port
allow	*	25	*	*	ACK	their replies

Rule Set E

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

Advantage of a packet filtering firewall:

- Is its simplicity.
- Typically are transparent to users
- Very fast.

Weaknesses of packet filter firewalls:

Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions. For example, a packet filter firewall cannot block specific application commands; if a packet filter firewall allows a given application, all functions available within that application will be permitted.

Because of the limited information available to the firewall, the logging functionality present in packet filter firewalls is limited. Packet filter logs normally contain the same information used to make access control decisions (source address, destination address, and traffic type).

Most packet filter firewalls do not support advanced user authentication schemes. Once again, this limitation is mostly due to the lack of upper-layer functionality by the firewall.

Packet filter firewalls are generally vulnerable to attacks and exploits that take advantage of problems within the TCP/IP specification and protocol stack, such as *network layer address spoofing*. Many packet filter firewalls cannot detect a network packet in which the OSI Layer 3 addressing information has been altered. Spoofing attacks are generally employed by intruders to bypass the security controls implemented in a firewall platform.

Finally, due to the small number of variables used in access control decisions, packet filter firewalls are susceptible to security breaches caused by improper configurations. In other words, it is easy to accidentally configure a packet filter firewall to allow traffic types, sources, and destinations that should be denied based on an organization's information security policy.

Attacks that can be made on packet filtering firewalls and the appropriate countermeasures:

IP address spoofing: The intruder transmits packets from the outside with a source IP address field containing an address of an internal host. The attacker hopes that the use of a spoofed address will allow penetration of systems that employ simple source address security, in which packets from specific trusted internal hosts are accepted. The countermeasure is to discard packets with an inside source address if the packet arrives on an external interface. In fact, this countermeasure is often implemented at the router external to the firewall.

Source routing attacks: The source station specifies the route that a packet should take as it crosses the Internet, in the hopes that this will bypass security measures that do not analyze the source routing information. The countermeasure is to discard all packets that use this option.

Tiny fragment attacks: The intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into a separate

packet fragment. This attack is designed to circumvent filtering rules that depend on TCP header information. Typically, a packet filter will make a filtering decision on the first fragment of a packet. All subsequent fragments of that packet are filtered out solely on the basis that they are part of the packet whose first fragment was rejected. The attacker hopes that the filtering firewall examines only the first fragment and that the remaining fragments are passed through. A tiny fragment attack can be defeated by enforcing a rule that the first fragment of a packet must contain a predefined minimum amount of the transport header. If the first fragment is rejected, the filter can remember the packet and discard all subsequent fragments.

Stateful Inspection Firewalls

A traditional packet filter makes filtering decisions on an individual packet basis and does not take into consideration any higher layer context. To understand what is meant by *context* and why a traditional packet filter is limited with regard to context, a little background is needed. Most standardized applications that run on top of TCP follow a client/server model. For example, for the Simple Mail Transfer Protocol (SMTP), e-mail is transmitted from a client system to a server system. The client system generates new e-mail messages, typically from user input. The server system accepts incoming e-mail messages and places them in the appropriate user mailboxes. SMTP operates by setting up a TCP connection between client and server, in which the TCP server port number, which identifies the SMTP server application, is 25. The TCP port number for the SMTP client is a number between 1024 and 65535 that is generated by the SMTP client.

In general, when an application that uses TCP creates a session with a remote host, it creates a TCP connection in which the TCP port number for the remote (server) application is a number less than 1024 and the TCP port number for the local (client) application is a number between 1024 and 65535. The numbers less than 1024 are the "well-known" port numbers and are assigned permanently to particular applications (e.g., 25 for server SMTP). The numbers between 1024 and 65535 are generated dynamically and have temporary significance only for the lifetime of a TCP connection.

A simple packet filtering firewall must permit inbound network traffic on all these high-numbered ports for TCP-based traffic to occur. This creates a vulnerability that can be exploited by unauthorized users.

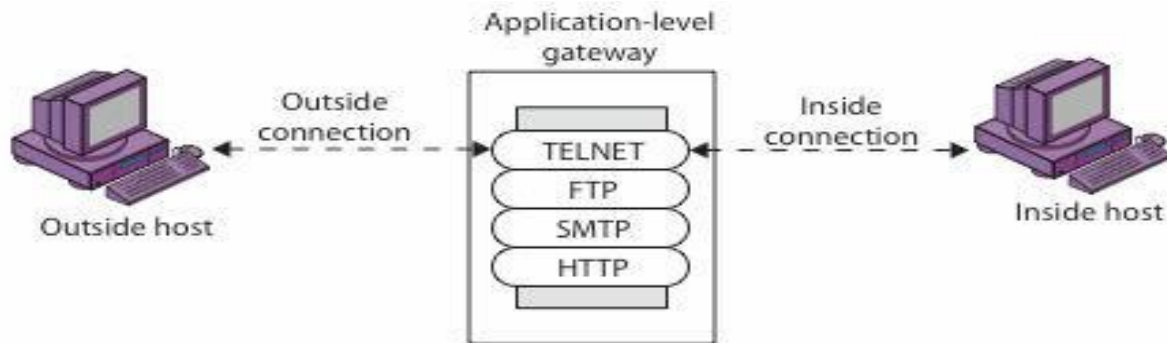
A stateful inspection packet firewall tightens up the rules for TCP traffic by creating a directory of outbound TCP connections. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory.

A stateful packet inspection firewall reviews the same packet information as a packet filtering firewall, but also records information about TCP connections. Some stateful firewalls also keep track of TCP sequence numbers to prevent attacks that depend on the sequence number, such as session hijacking. Some even inspect limited amounts of application data for some well-known protocols like FTP, IM and SIP commands, in order to identify and track related connections.

Application-Level Gateway

An application-level gateway, also called an **application proxy**, acts as a relay of application-level traffic (Figure below). 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.



(b) Application-level gateway

Advantages:

- ▢ Application-level gateways tend to be more secure than packet filters.
- ▢ The application-level gateway need only scrutinize a few allowable applications.
- ▢ It is easy to log and audit all incoming traffic at the application level instead of dealing with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level

Disadvantages:

- ▢ A prime disadvantage of this type of gateway is the additional processing overhead on each connection.

Circuit-Level Gateway

Stand-alone system or it can be a specialized function performed by an application-level gateway

Sets up two TCP connections

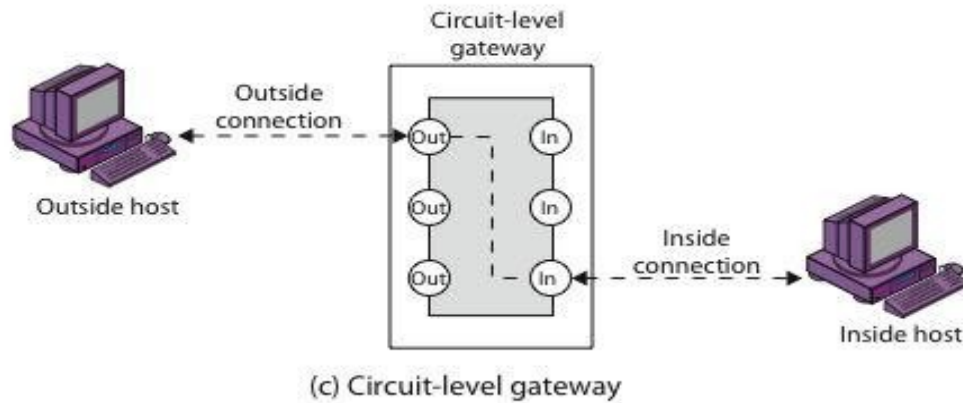
One between itself and an inner host and one between itself and an outside host

The gateway typically relays TCP segments from one connection to the other without examining the contents

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users.

The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections.

In this configuration, the gateway can incur the processing overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.



Bastion Host:

A bastion host is a system identified by the firewall administrator as a critical strong point in the network's security.

The bastion host serves as a platform for an application-level or circuit-level gateway.

Characteristics of Bastion Host:

Bastion host hardware platform executes a secure version of its OS.

Only services that are considered essential are installed on the bastion host.

Requires additional authentication before a user is allowed access to the proxy services

Each proxy is configured to support only a subset of the standard application's command set

Each proxy is configured to allow access only to specific host systems

Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection

The audit log is an essential tool for discovering and terminating intruder attacks.

Each proxy is independent of other proxies on the bastion host.

Each proxy runs as a non-privileged user in a private and secured directory on the bastion host.

Firewall Configurations:

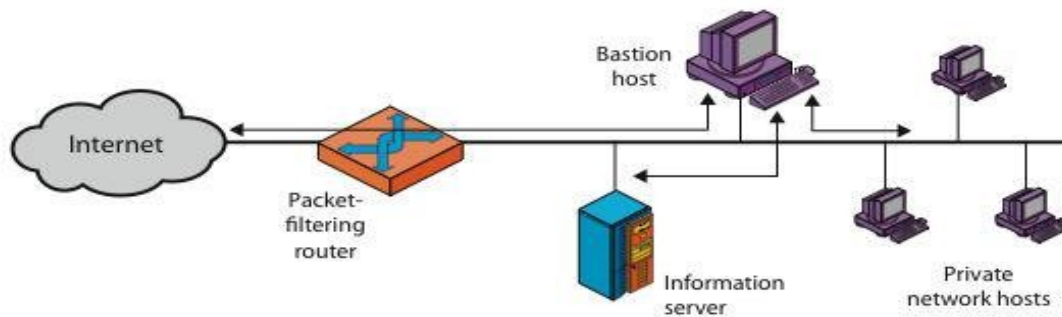
Three common firewall configurations:

- Screened host firewall, single-homed bastion

- Screened host firewall, dual-homed bastion

- Screened subnet firewall

Screened host firewall, single-homed bastion:



(a) Screened host firewall system (single-homed bastion host)

The firewall consists of Two systems : Packet filtering router, bastion host

For traffic from the internet only the IP packets destined to the bastion host are allowed in.

For traffic from the internal network, only IP packets from the bastion host are allowed out

Bastion Host performs authentication and proxy functions

Better security

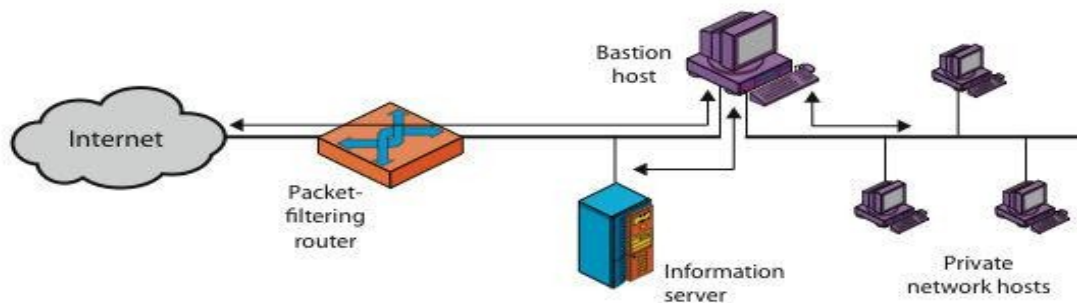
Implements both packet and application level filtering

Intruder must penetrate two separate systems before the security of the internal network is compromised.

Flexibility in providing direct internet access to the information server which does not require high level of security

If packet filter is compromised traffic flows directly to the private network hosts

Screened host firewall, dual-homed bastion:

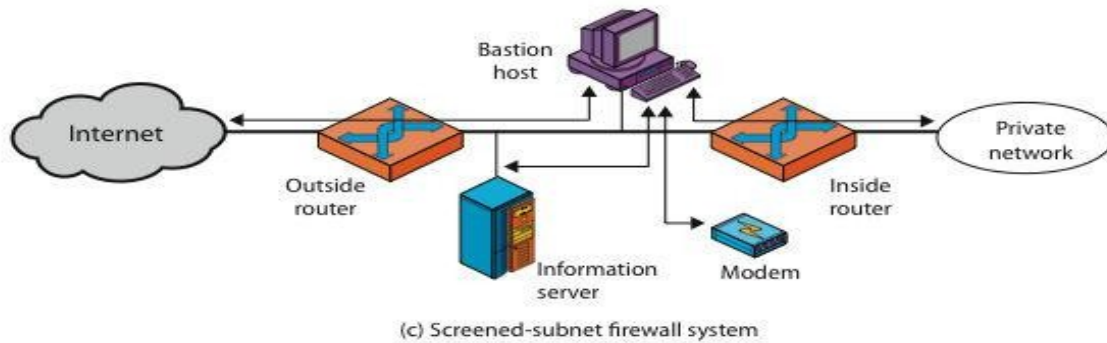


(b) Screened host firewall system (dual-homed bastion host)

In this configuration the external and internal networks are physically separated
Two systems must be compromised to breach security.

Information server or other hosts can be allowed direct communication with the router but are now separated from the internal network.

Screened subnet firewall:



Most secure

Two packet filters, Bastion Host

Creates an isolated subnetwork, which may consist of simply the bastion host but may also include one or more information servers and modems for dial-in capability

Both the Internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked

Three levels of defense to thwart intruders

Internal network is invisible to the Internet, because the outside router advertises only the existence of the screened subnet to the internet

The systems on the inside network cannot construct direct routes to the Internet because the inside router advertises only the existence of the screened subnet to the internal network.

Why do Firewalls Fail?

Assume all bad guys are on outside, and everyone inside can be trusted.

Firewalls can be defeated if malicious code can be injected into corporate network

- E.g. trick someone into launching an executable from an email message or into downloading something from the net.

Often make it difficult for legitimate users to get their work done.

- Misconfiguration, failure to recognize new app

If firewall allows anything through, people figure out how to do what they need by disguising their traffic as allowed traffic

- E.g. file transfer by sending it through email.

If size of emails limited, then user breaks them into chunks, etc.

Firewall friendly traffic (e.g. using http for other purposes)

- Defeats effort of sys admin to control traffic

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology. Data Access Control:

Through the user access control procedure (log on), a user can be identified to the system

Associated with each user, there can be a profile that specifies permissible operations and file accesses

The operation system can enforce rules based on the user profile

General models of access control:

- Access matrix
- Access control list
- Capability list

Access Matrix:

	Program1	...	SegmentA	SegmentB
Process1	Read Execute		Read Write	
Process2				Read
•				
•				
•				

The basic elements of the model are as follows:

Subject: An entity capable of accessing objects, the concept of subject equates with that of process

Object: Anything to which access is controlled (e.g. files, programs)

Access right: The way in which an object is accessed by a subject (e.g. read, write, execute)

One axis of the matrix consists of identified subjects that may attempt data access. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, hosts, or applications instead of or in addition to users. The other axis lists the objects that may be accessed. At the greatest level of detail, objects may be individual data fields. More aggregate groupings, such as records, files, or even the entire database, may also be objects in the matrix. Each entry in the matrix indicates the access rights of that subject for that object.

Access Control List: Decomposition of the matrix by columns.

Access Control List for Program1: Process1 (Read, Execute)
Access Control List for SegmentA: Process1 (Read, Write)
Access Control List for SegmentB: Process2 (Read)

Thus, for each object, an access control list lists users and their permitted access rights. The access control list may contain a default, or public, entry. This allows users that are not explicitly listed as having special rights to have a default set of rights. Elements of the list may include individual users as well as groups of users.

Capability ticket: Decomposition of the matrix by rows.

Capability List for Process1: Program1 (Read, Execute) SegmentA (Read, Write)
Capability List for Process2: SegmentB (Read)

A capability ticket specifies authorized objects and operations for a user. Each user has a number of tickets and may be authorized to loan or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access control lists. In particular, the ticket must be unforgeable. One way to accomplish this is to have the operating system hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users.

The Concept of Trusted Systems:

Much of what we have discussed so far has been concerned with protecting a given message or item from passive or active attack by a given user. A somewhat different but widely applicable requirement is to protect data or resources on the basis of levels of security. This is commonly found in the military, where information is categorized as unclassified (U), confidential (C), secret (S), top secret (TS), or beyond. This concept is equally applicable in other areas, where information can be organized into gross categories and users can be granted clearances to access certain categories of data. For example, the highest level of security might be for strategic corporate planning documents and data, accessible by only corporate officers and their staff; next might come sensitive financial and personnel data, accessible only by administration personnel, corporate officers, and so on.

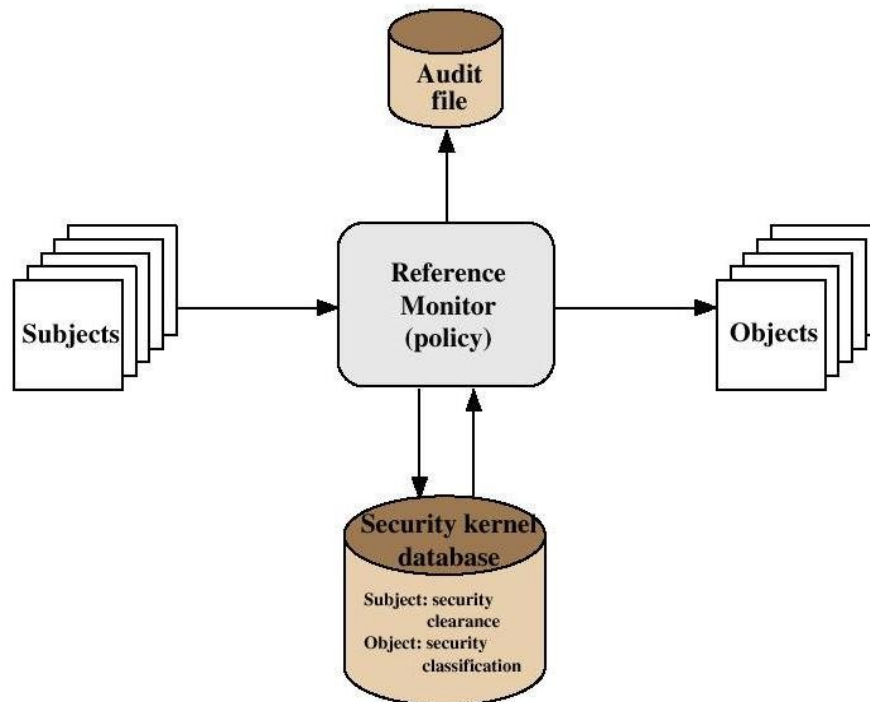
Multilevel security

- Definition of multiple categories or levels of data

A multilevel secure system must enforce:

- No read up: A subject can only read an object of less or equal security level (Simple Security Property)
- No write down: A subject can only write into an object of greater or equal security level (*-Property)

Reference Monitor Concept: Multilevel security for a data processing system



Reference Monitor

- Controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on basis of security parameters
- The monitor has access to a file (security kernel database)
- The monitor enforces the security rules (no read up, no write down)

Properties of the Reference Monitor

- Complete mediation: Security rules are enforced on every access
- Isolation: The reference monitor and database are protected from unauthorized modification
- Verifiability: The reference monitor's correctness must be provable (mathematically)

A system that can provide such verifications (properties) is referred to as a trusted system

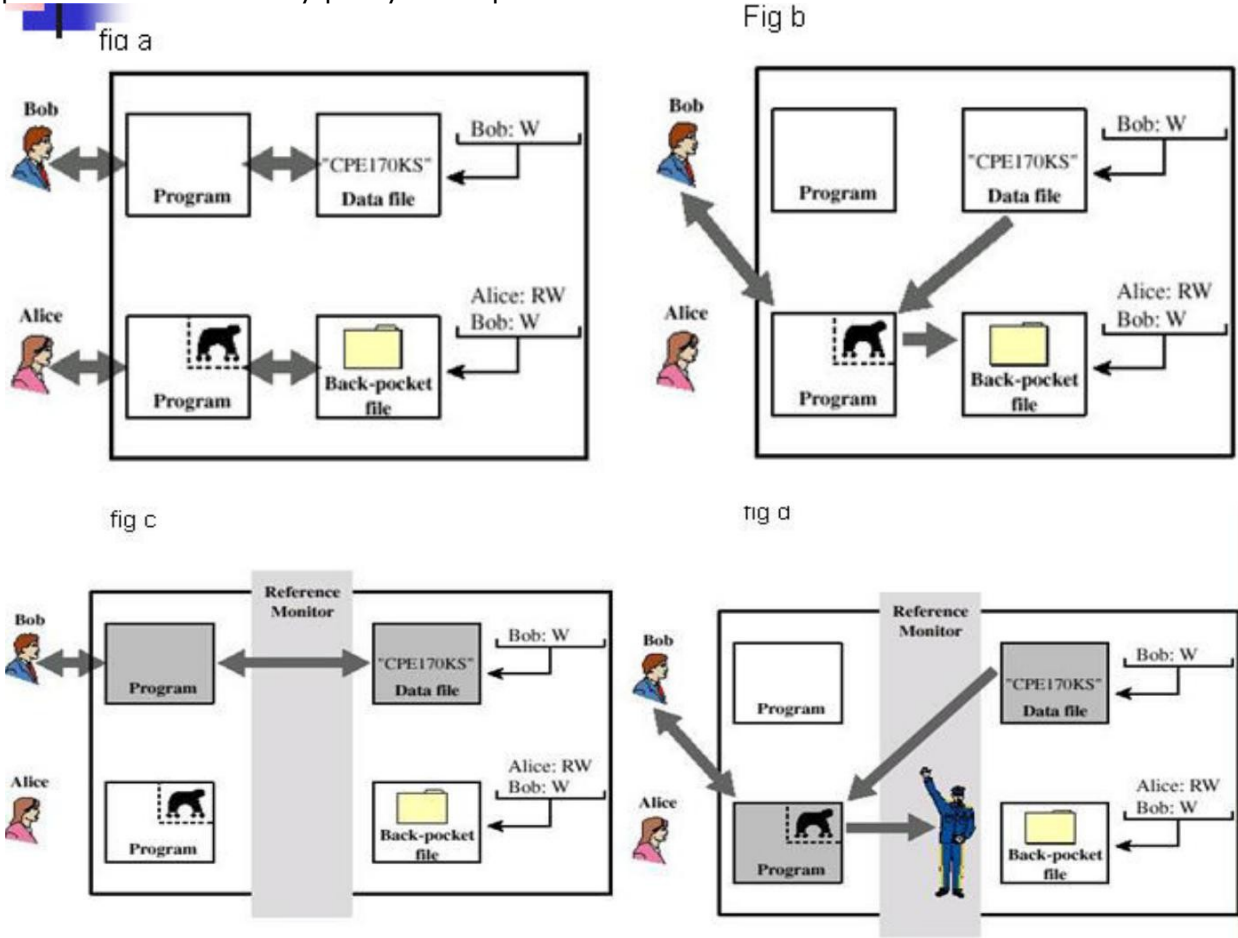
Trojan Horse Defense

Secure, trusted operating systems are one way to secure against Trojan Horse attacks. Figure below illustrates an example. In this case, a Trojan horse is used to get around the standard security mechanism used by most file management and operating systems: the access control list. In this example, a user named Doe interacts through a program with a data file containing the critically sensitive character string "CPEI704TKS." User Doe has created the file with read/write permission provided only to programs executing on his own behalf: that is, only processes that are owned by Doe may access the file.

The Trojan horse attack begins when a hostile user named Drake Gains legitimate access to the system, and installs both a Trojan horse program and a private file to be used in the attack as a "back pocket." Drake gives read/write permission to himself for this file and gives Doe write-only permission (Figure a). Drake now induces Doe to invoke the Trojan horse program, perhaps by advertising it as a useful utility. When the program detects that it is being executed by Doe, it copies the sensitive character string from Doe's file and copies it into Drake's back-pocket file (Figure b). Both the read and write operations satisfy

the constraints imposed by access control lists. Drake then has only to access his file at a later time to learn the value of the string.

Now consider the use of a secure operating system in this scenario (Figure c). Security levels are assigned to subjects at logon on the basis of criteria such as the terminal from which the computer is being accessed and the user involved, as identified by password/ID. In this example, there are two security levels. Sensitive and public, ordered so that sensitive is higher than public. Processes owned by Doe and Doe's data file are assigned the security level sensitive. Drake's file and processes are restricted to public. If Doe invokes the Trojan horse program (Figure d), that program acquires Doe's security level. It is therefore able, under the simple security property, to observe the sensitive character string. When the program attempts to store the string in a public file (the back-pocket file), however, the *-Property is violated and the attempt is disallowed by the reference monitor. Thus, the attempt to write into the back-pocket file is denied even though the access control list permits it: The security policy takes precedence over the access control list mechanism.



Evaluated Computer Systems:

Trusted systems need to be evaluated against a suitable set of criteria by an approved government agency.

The original standard developed by the US DoD & NSA was TCSEC in the early 80's. Later standards were developed by other countries, harmonized in the EU with IPSEC (which was also used in Australia) now internationally with the Common Criteria.

These standards define a number of "levels" of evaluation with increasingly stringent checking, to which an evaluation center evaluates commercially

available products as meeting the security requirements specified, within a given functionality area.

These evaluations are needed for Defense procurements but are published and freely available, & can serve as guidance to commercial customers for the purchase of commercially available, off-the-shelf equipment.

Common Criteria:

The Common Criteria (CC) for Information Technology and Security Evaluation is an international initiative by standards bodies in a number of countries to develop international standards for specifying security requirements and defining evaluation criteria. It provides standards for the evaluation criteria, the methodology for the application of these criteria, the administrative procedures used for evaluation, certification and accreditation schemes.

- The CC defines a common set of potential security requirements for use in evaluation. The term target of evaluation (TOE) refers to that part of the product or system that is subject to evaluation. The requirements fall in two categories:
- Functional requirements: define desired security behavior, have a set of security functional components that provide a standard way of expressing the security functional requirements for a TOE
- Assurance requirements: basis for gaining confidence that the claimed security measures are effective and implemented correctly
- Both functional requirements and assurance requirements are organized into classes, being a collection of requirements that share a common focus or intent. Each of these classes contains a number of families which share security objectives, & in turn contain one or more components.

Common Criteria Requirements:

Functional Requirements

- security audit, crypto support, communications, user data protection, identification & authentication, security management, privacy, protection of trusted security functions, resource utilization, TOE access, trusted path

Assurance Requirements

- configuration management, delivery & operation, development, guidance documents, life cycle support, tests, vulnerability assessment, assurance maintenance.

The CC also defines two kinds of documents that can be generated using the CC-defined requirements

Protection profiles (PPs): define an implementation-independent reusable set of security requirements and objectives for a category of products or systems that meet similar consumer needs for IT security, reflecting user security requirements

Security targets (STs): contain the IT security objectives and requirements of a specific identified TOE and defines the functional and assurance measures offered by that TOE to meet stated requirements, and forms the basis for an evaluation

Stallings Figure 20.6 illustrates the relationship between requirements and profiles and targets.

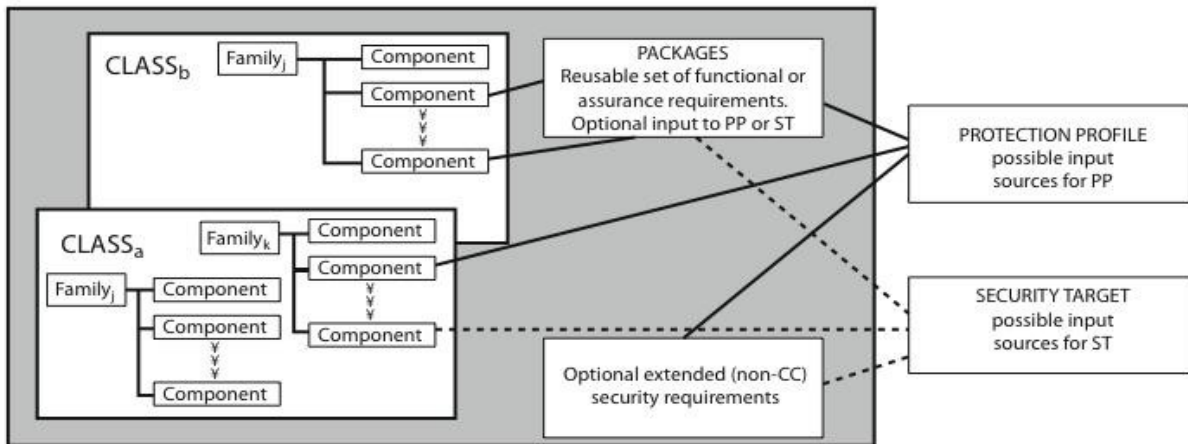


Fig: organization and construction of common criteria Requirements

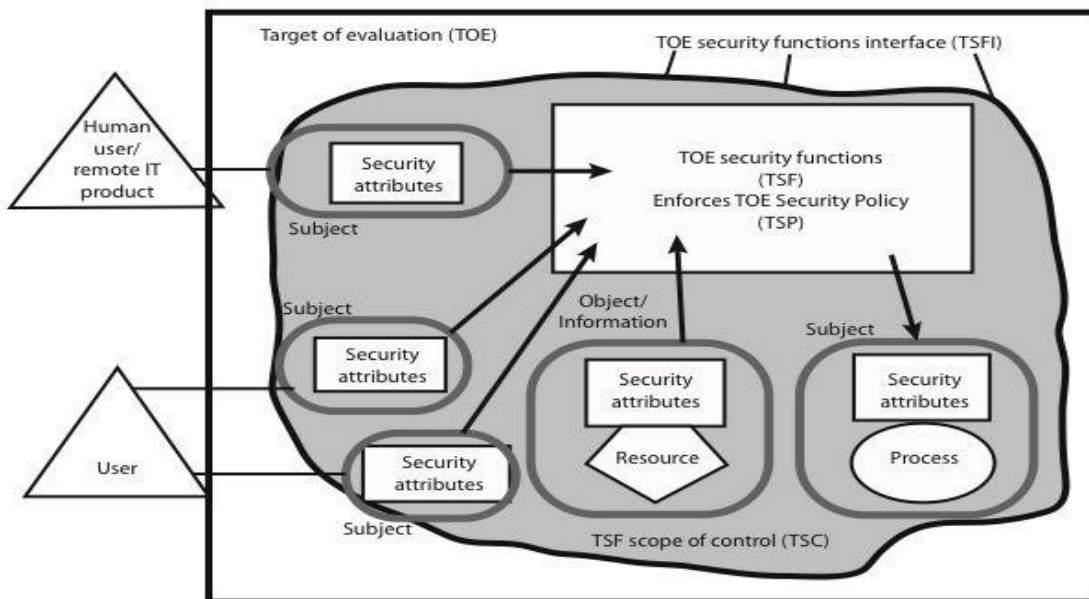


Fig: security functional requirements paradigm.