A Course Material on

CRYPTOGRAPHY AND NETWORK SECURITY

By

Mr. S. SHANTHA KUMAR

ASSOCIATE PROFESSOR

DEPARTMENT OF INFORMATION TECHNOLOGY & COMPUTER APPLICATIONS

SASURIE COLLEGE OF ENGINEERING

VIJAYAMANGALAM – 638 056

QUALITY CERTIFICATE

This is to certify that the e-course material

Subject Code	: **IT2352**
Scubject	**: CRYPTOGRAPHY AND NETWORK SECURITY**
Class	**: III B.Tech (INFORMATION TECHNOLOGY)**

being prepared by me and it meets the knowledge requirement of the university curriculum.

Signature of the Author

Name: S. Shantha Kumar
Designation: Assistant Professor/IT

This is to certify that the course material being prepared by Mr.S.Shantha Kumar is of adequate quality. He has referred more than five books amont them minimum one is from aborad author.

Signature of HD

Name: S. Ashok Kumar , HD/IT

SEAL

UNIT 1

1.1 INTRODUCTION

Computer data often travels from one computer to another, leaving the safety of its protected physical surroundings. Once the data is out of hand, people with bad intention could modify or forge your data, either for amusement or for their own benefit.

Cryptography can reformat and transform our data, making it safer on its trip between computers. The technology is based on the essentials of secret codes, augmented by modern mathematics that protects our data in powerful ways.

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
- **Network Security** - measures to protect data during their transmission
- **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks

1.2 THE OSI SECURITY ARCHITECTURE

To assess effectively the security needs of an organization and to evaluate and choose various security products and policies, the manager responsible for security needs some systematic way of defining the requirements for security and characterizing the approaches to satisfying those requirements. The OSI security architecture was developed in the context of the OSI protocol architecture, which is described in Appendix H. However, for our purposes in this chapter, an understanding of the OSI protocol architecture is not required.

For our purposes, the OSI security architecture provides a useful, if abstract, overview of many of the concepts.. The OSI security architecture focuses on security attacks, mechanisms, and services. These can be defined briefly as follows:

Threats and Attacks (RFC 2828)

Threat

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

Attack

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

Security Attacks, Services And Mechanisms

To assess the security needs of an organization effectively, the manager responsible for security needs some systematic way of defining the requirements for security and characterization of approaches to satisfy those requirements. One approach is to consider three aspects of information security:

- **Security attack** – Any action that compromises the security of information owned by an organization.
- **Security mechanism** – A mechanism that is designed to detect, prevent or recover from a security attack.
- **Security service** – A service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks and they make use of one or more security mechanisms to provide the service.

1.2.1 SECURITY SERVICES

The classification of security services are as follows:

- **Confidentiality:** Ensures that the information in a computer system and transmitted information are accessible only for reading by authorized parties.
  Eg., printing, displaying and other forms of disclosure.

- **Authentication:** Ensures that the origin of a message or electronic document is correctly identified, with an assurance that the identity is not false.

- **Integrity:** Ensures that only authorized parties are able to modify computer system assets and transmitted information. Modification includes writing, changing status, deleting, creating and delaying or replaying of transmitted messages.
- **Non repudiation**: Requires that neither the sender nor the receiver of a message be able to deny the transmission.
- **Access control**: Requires that access to information resources may be controlled by or the target system.
- **Availability**: Requires that computer system assets be available to authorized parties when needed.

Security Services (X.800)

AUTHENTICATION

The assurance that the communicating entity is the one that it claims to be.

- **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.

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).

DATA CONFIDENTIALITY

The protection of data from unauthorized disclosure.

- **Connection Confidentiality**

The protection of all user data on a connection.

- **Connectionless Confidentiality**

The protection of all user data in a single data block

AUTHENTICATION

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

- **Traffic Flow Confidentiality**

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

- **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.

NONREPUDIATION

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

1.2.2 SECURITY MECHANISMS

One of the most specific security mechanisms in use is cryptographic techniques. Encryption or encryption-like transformations of information are the most common means of providing security. Some of the mechanisms are:

- Encipherment
- Digital Signature
- Access Control

1.2.3 SECURITY ATTACKS

There are four general categories of attack which are listed below.

- **Interruption**

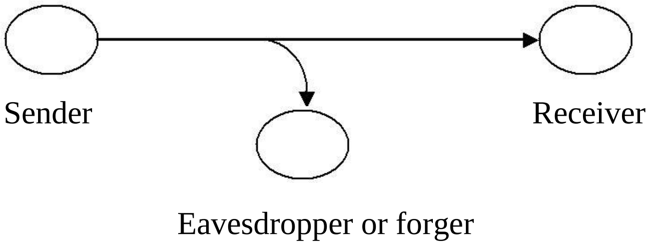
  An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability.

  e.g., destruction of piece of hardware, cutting of a communication line or disabling of file management system.

Sender
Receive

- **Interception**

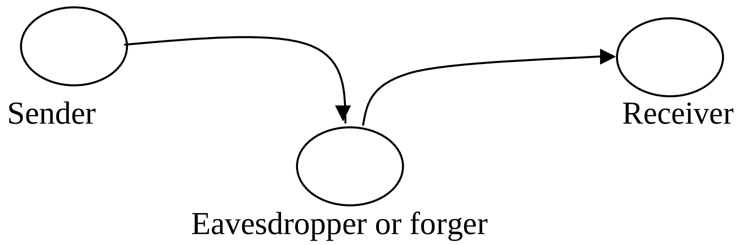
  An unauthorized party gains access to an asset. This is an attack on confidentiality. Unauthorized party could be a person, a program or a computer.e.g., wire tapping to capture data in the network, illicit copying of files



- **Modification**

  An unauthorized party not only gains access to but tampers with an asset. This is an attack on integrity.

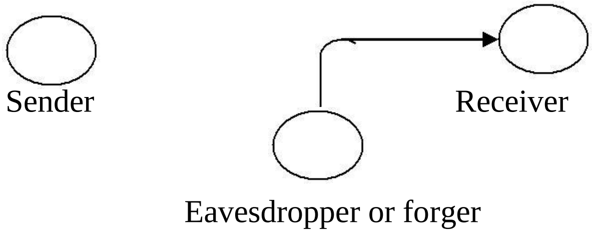
  e.g., changing values in data file, altering a program, modifying the contents of messages being transmitted in a network.



- **Fabrication**

An unauthorized party inserts counterfeit objects into the system. This is an attack on authenticity.

e.g., insertion of spurious message in a network or addition of records to a file.



A useful categorization of these attacks is in terms of

- Passive attacks
- Active attacks

Passive attack

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Passive attacks are of two types:

- **Release of message contents:** A telephone conversation, an e-mail message and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.
- **Traffic analysis**: If we had encryption protection in place, an opponent might still be able to observe the pattern of the message. The opponent could determine the location and identity of communication hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of data. However, it is feasible to prevent the success of these attacks.

Active attacks

These attacks involve some modification of the data stream or the creation of a false stream. These attacks can be classified in to four categories:

- **Masquerade** – One entity pretends to be a different entity.
- **Replay** – involves passive capture of a data unit and its subsequent transmission to produce an unauthorized effect.
- **Modification of messages** – Some portion of message is altered or the messages are delayed or recorded, to produce an unauthorized effect.
- **Denial of service** – Prevents or inhibits the normal use or management of communication facilities. Another form of service denial is the disruption of an entire network, either by disabling the network or overloading it with messages so as to degrade performance.

It is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them.

Symmetric and public key algorithms

Encryption/Decryption methods fall into two categories.

- Symmetric key
- Public key

In symmetric key algorithms, the encryption and decryption keys are known both to sender and receiver. The encryption key is shared and the decryption key is easily calculated from it. In many cases, the encryption and decryption keys are the same.

In public key cryptography, encryption key is made public, but it is computationally infeasible to find the decryption key without the information known to the receiver.

1.3 CLASSICAL CRYPTO SYSTEMS

1.3.1 CONVENTIONAL ENCRYPTION

- referred conventional / private-key / single-key
- sender and recipient share a common key
- all classical encryption algorithms are private-key
- was only type prior to invention of public-key in 1970‟**plaintext** - the original message

Some basic terminologies used :

- **ciphertext** - the coded message
- **cipher** - algorithm for transforming plaintext to ciphertext
- **key** - info used in cipher known only to sender/receiver
- **encipher (encrypt)** - converting plaintext to ciphertext
- **decipher (decrypt)** - recovering ciphertext from plaintext
- **cryptography** - study of encryption principles/methods
- **cryptanalysis (codebreaking)** - the study of principles/ methods of deciphering ciphertext *without* knowing key
- **cryptology** - the field of both cryptography and cryptanalysis

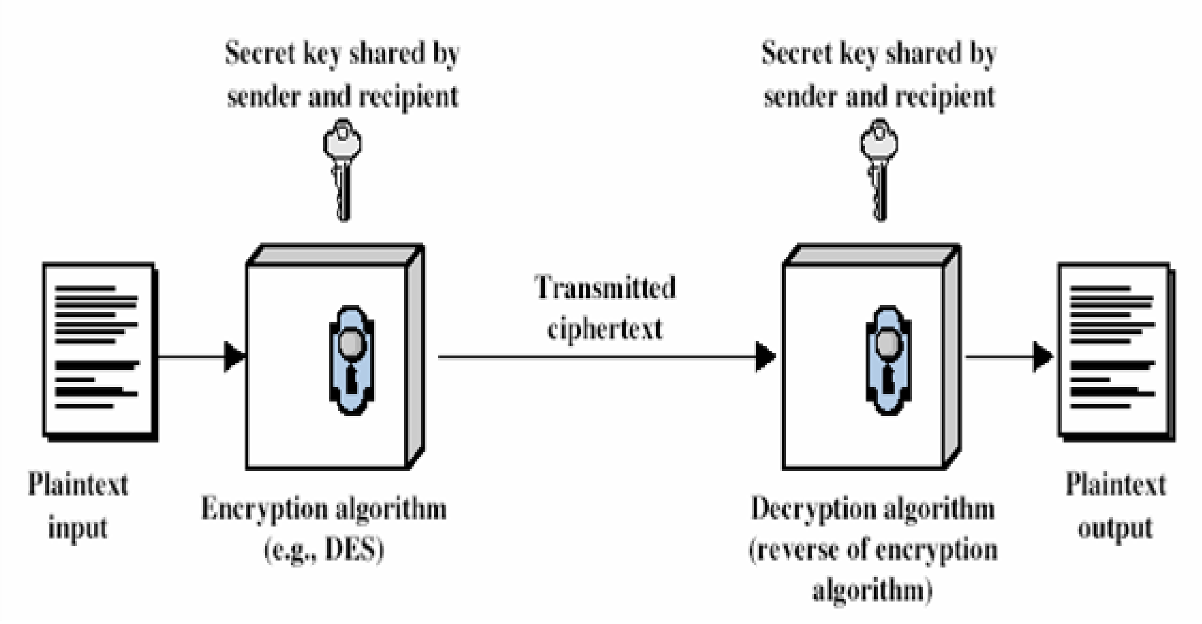

Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Figure 1.3.1.1: Conventional Encryption

Here the original message, referred to as plaintext, is converted into apparently random nonsense, referred to as cipher text. The encryption process consists of an algorithm and a key. The key is a value independent of the plaintext. Changing the key changes the output of the algorithm. Once the cipher text is produced, it may be transmitted. Upon reception, the cipher text can be transformed back to the original plaintext by using a decryption algorithm and the same key that was used for encryption.

The security depends on several factors. First, the encryption algorithm must be powerful enough that it is impractical to decrypt a message on the basis of cipher text alone. Beyond that, the security depends on the secrecy of the key, not the secrecy of the algorithm.

- **Two requirements for secure use of symmetric encryption:**
  - a strong encryption algorithm
  - a secret key known only to sender / receiver

    $Y = \mathrm{E}K(X)$

    $X = \mathrm{D}K(Y)$
- **assume encryption algorithm is known**
- **implies a secure channel to distribute key**

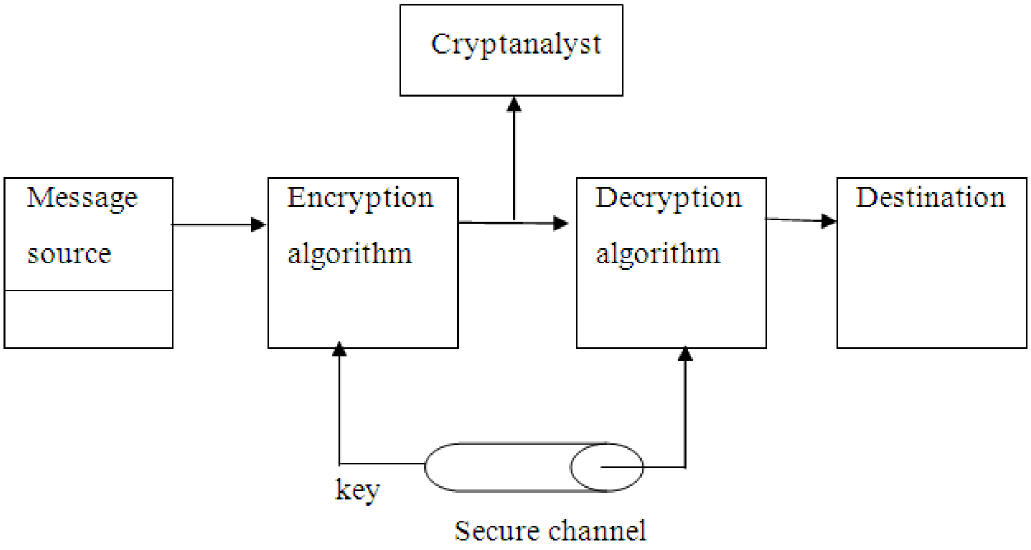


Figure1.3.1.2: conventional cryptosystem

A source produces a message in plaintext, X = [X1, X2, … , XM] where M are the number of letters in the message. A key of the form K = [K1, K2, …, KJ] is generated. If the key is generated at the source, then it must be provided to the destination by means of some secure channel.

With the message X and the encryption key K as input, the encryption algorithm forms the cipher text Y = [Y1, Y2, …, YN]. This can be expressed as

$$Y = E_K(X)$$

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

$$X = D_K(Y)$$

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

Cryptography

Cryptographic systems are generally classified along 3 independent dimensions:

- **Type of operations used for transforming plain text to cipher text**

  All the encryption algorithms are abased on two general principles: **substitution**, in which each element in the plaintext is mapped into another element, and **transposition**, in which elements in the plaintext are rearranged.

- **The number of keys used**

  If the sender and receiver uses same key then it is said to be **symmetric key (or) single key (or) conventional encryption**.

  If the sender and receiver use different keys then it is said to be **public key encryption**.

- **The way in which the plain text is processed**

  A **block cipher** processes the input and block of elements at a time, producing output block for each input block.

  A **stream cipher** processes the input elements continuously, producing output element one at a time, as it goes along.

Cryptanalysis

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

There are various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst.

- **Cipher text only** – A copy of cipher text alone is known to the cryptanalyst.
- **Known plaintext** – The cryptanalyst has a copy of the cipher text and the corresponding plaintext.
- **Chosen plaintext** – The cryptanalysts gains temporary access to the encryption machine. They cannot open it to find the key, however; they can encrypt a large number of suitably chosen plaintexts and try to use the resulting cipher texts to deduce the key.
- **Chosen cipher text** – The cryptanalyst obtains temporary access to the decryption machine, uses it to decrypt several string of symbols, and tries to use the results to deduce the key.

1.3.2 STEGANOGRAPHY

A plaintext message may be hidden in any one of the two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.

A simple form of steganography, but one that is time consuming to construct is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message.

Example:

(i) the sequence of first letters of each word of the overall message spells out the real (hidden) message.

(ii) Subset of the words of the overall message is used to convey the hidden message.

Various other techniques have been used historically, some of them are:

- **Character marking** – selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held to an angle to bright light.

- **Invisible ink** – a number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
- **Pin punctures** – small pin punctures on selected letters are ordinarily not visible unless the paper is held in front of the light.
- **Typewritten correction ribbon** – used between the lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Drawbacks of steganography

- Requires a lot of overhead to hide a relatively few bits of information.
- Once the system is discovered, it becomes virtually worthless.

1.4 CLASSICAL ENCRYPTION TECHNIQUES

There are two basic building blocks of all encryption techniques: substitution and transposition.

1.4.1 SUBSTITUTION TECHNIQUES

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with cipher text bit patterns.

(i)**Caesar cipher (or) shift cipher**

The earliest known use of a substitution cipher and the simplest was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing 3 places further down the alphabet.

e.g., Plain text : pay more mone

Cipher text: SDB PRUH PRQHB

Note that the alphabet is wrapped around, so that letter following „z‟ is „a‟.

For each plaintext letter p, substitute the cipher text letter c such that C =

E(p) = (p+3) mod 26

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

C = E (p) = (p+k) mod 26

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

P = D(C) = (C-k) mod 26

(ii)Playfair cipher

The best known multiple letter encryption cipher is the playfair, which treats digrams in the plaintext as single units and translates these units into cipher text digrams. The playfair algorithm is based on the use of 5x5 matrix of letters constructed using a keyword. Let the keyword be „monarchy‟. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetical order.

The letter „i‟ and „j‟ count as one letter. Plaintext is encrypted two letters at a time according to the following rules:

- Repeating plaintext letters that would fall in the same pair are separated with a filler letter such as „x‟.
- Plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row following the last.
- Plaintext letters that fall in the same column are replaced by the letter beneath, with the top element of the column following the last.
- Otherwise, each plaintext letter is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter.

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

Plaintext = meet me at the school house

Splitting two letters as a unit => me et me at th es ch ox ol ho us ex Corresponding

cipher text => CL KL CL RS PD IL HY AV MP HF XL IU

Strength of playfair cipher

- Playfair cipher is a great advance over simple mono alphabetic ciphers.
- Since there are 26 letters, 26x26 = 676 diagrams are possible, so identification of individual digram is more difficult.
- Frequency analysis is much more difficult.

(iii)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 cipher. All the techniques have the following features in common.

- A set of related monoalphabetic substitution rules are used
- A key determines which particular rule is chosen for a given transformation.

(iv)Vigenere cipher

In this scheme, the set of related monoalphabetic substitution rules consisting of 26 caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter. e.g.,
Caesar cipher with a shift of 3 is denoted by the key value 'd‟ (since a=0, b=1, c=2 and so on). To aid in understanding the scheme, a matrix known as vigenere tableau is constructed.

Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of encryption is simple: Given a key letter X and a plaintext letter y, the cipher text 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.

e.g., key = d e c e p t i v e d e c e p t i v e d e c e p t i v e
PT = w e a r e d i s c o v e r e d s a v e y o u r s e l f
CT = ZICVTWQNGRZGVTWAVZHCQYGLMGJ

	PLAIN TEXT															
K		a	b	c	d	e	f	g	h	i	j	k	…	x	y	z
	a	A	B	C	D	E	F	G	H	I	J	K	…	X	Y	Z
Y	b	B	C	D	E	F	G	H	I	J	K	L	…	Y	Z	A
	c	C	D	E	F	G	H	I	J	K	L	M	…	Z	A	B
L	d	D	E	F	G	H	I	J	K	L	M	N	…	A	B	C
E	e	E	F	G	H	I	J	K	L	M	N	O	…	B	C	D
T	f	F	G	H	I	J	K	L	M	N	O	P	…	C	D	E
T	g	G	H	I	J	K	L	M	N	O	P	Q	…	D	E	F
E R	: :	: :	: :	: :	: :	: :	: :	: :	: :	: :	: :	: :	…	: :	: :	: :
S	x	X	Y	Z	A	B	C	D	E	F	G	H	…			W
	y	Y	Z	A	B	C	D	E	F	G	H	I	…			X
	z	Z	A	B	C	D	E	F	G	H	I	J	…			Y

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

Strength of Vigenere cipher

- There are multiple ciphertext letters for each plaintext letter
- Letter frequency inforamiton is obscured.

One Time Pad Cipher

It is an unbreakable cryptosystem. It represents the message as a sequence of 0s and 1s. this can be accomplished by writing all numbers in binary, for example, or by using ASCII. The key is a random sequence of 0‟s and 1‟s of same length as the message.

Once a key is used, it is discarded and never used again. The system can be expressed as follows:

$$C_i = P_i \oplus K_i$$

C_i - ith binary digit of cipher text P_i - ith binary digit of plaintext

K_i - ith binary digit of key $\oplus$ – exclusive OR opearaiton

Thus the cipher text is generated by performing the bitwise XOR of the plaintext and the key. Decryption uses the same key. Because of the properties of XOR, decryption simply involves the same bitwise operation:

$$P_i = C_i \oplus K_i$$

```
e.g.,   plaintext  = 0 0 1 0 1 0 0 1
          Key      = 1 0 1 0 1 1 0 0
                     -------------------
        ciphertext = 1 0 0 0 0 1 0 1
```

Advantage:

- Encryption method is completely unbreakable for a ciphertext only attack.

Disadvantages

- It requires a very long key which is expensive to produce and expensive to transmit.
- Once a key is used, it is dangerous to reuse it for a second message; any knowledge on the first message would give knowledge of the second.

1.4.2 TRANSPOSITION TECHNIQUES

All the techniques examined so far involve the substitution of a cipher text symbol for a plaintext symbol. A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.

Rail fence is simplest of such cipher, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

Plaintext = meet at the school house

To encipher this message with a rail fence of depth 2, we write the message as follows:

m e a t e c o l o s
e t t h s H o h u e

The encrypted message is

MEATECOLOSETTHSHOHUE

Row Transposition Ciphers-A more complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of columns then becomes the key of the algorithm.

e.g., plaintext = meet at the school house

Key =	4	3	1	2	5	6	7
PT =	m	e	e	t	a	t	t
	h	e	s	c	h	o	o
	l	h	o	u	s	e	

CT = ESOTCUEEHMHLAHSTOETO

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is more complex permutation that is not easily reconstructed.

1.5 LINEAR FEEDBACK SHIFT REGISTER (LFSR)

- An LFSR of length m consists of m stages numbered 1, 2, . . . ,m, each storing one bit and having one input and one output; together with a clock which controls the movement of data.
- The vector (k1, k2, · · · , km) would be used to initialize the shift register. During each unit of time the following operations would be performed concurrently

  (i) k1 would be tapped as the next keystream bit

  (ii) k2, · · · , km would each be shifted one stage to the left

  (iii) the “new” value of km would be computed to be

$$\sum_{j=1}^{m-1} \mathbf{c_j k_j + 1}$$

the linear feedback is carried out by tapping certain stages of the register (as specified by the constants cj having the value “1”) and computing a sum modulo 2 (which is an exclusive-or).

Applications:

- Pseudo-random number
- Pseudo-noise sequences
- Digital counters

1.6 **INTRODUCTION TO NUMBER THEORY**

1.6.1 Primality Testing and RSA

- The first stage of key-generation for RSA involves finding two large primes p, q
- Because of the size of numbers used, must find primes by trial and error
- Modern primality tests utilize properties of primes eg:
  - $a^{n-1} = 1 \bmod n$ where GCD(a,n)=1
  - all primes numbers 'n' will satisfy this equation
  - some composite numbers will also satisfy the equation, and are called pseudo-primes.
- Most modern tests guess at a prime number 'n', then take a large number (eg 100) of numbers 'a', and apply this test to each. If it fails the number is composite, otherwise it is is probably prime.
- There are a number of stronger tests which will accept fewer composites as prime than the above test. eg:

$$\mathrm{GCD}(a,n) = 1, \quad \text{and} \quad \left(\frac{a}{n}\right) (\bmod\ n) = a^{\frac{(n-1)}{2}} \ (\bmod\ n)$$

$$\text{where} \quad \left(\frac{a}{n}\right) \text{ is the Jacobi symbol}$$

RSA Implementation in Practice

- Software implementations
  - generally perform at 1-10 bits/second on block sizes of 256-512 bits
  - two main types of implementations:
    - - on micros as part of a key exchange mechanism in a hybrid scheme
    - - on larger machines as components of a secure mail system
- Harware Implementations
  - generally perform 100-10000 bits/sec on blocks sizes of 256-512 bits
  - all known implementations are large bit length conventional ALU units

1.6.2 Euler Totient Function [[phi]](n)

- if consider arithmetic modulo n, then a **reduced set of residues** is a subset of the complete set of residues modulo n which are relatively prime to n
    - eg for n=10,
    - the complete set of residues is {0,1,2,3,4,5,6,7,8,9}
    - the reduced set of residues is {1,3,7,9}
- the number of elements in the reduced set of residues is called the **Euler Totient function [[phi]](n)**
- there is no single formula for [[phi]](n) but for various cases count how many elements are excluded:

p (p prime)	[[phi]](p) =p-1
pr (p prime)	[[phi]](p) =pr-1(p-1)
p.q (p,q prime)	[[phi]](p.q) =(p-1)(q-1)

several important results based on [[phi]](n) are:

- **Theorem (Euler's Generalization)**
    - let gcd(a,n)=1 then
    - $a^{[[phi]](n)}$ mod n = 1

- Fermat's Theorem
    - let p be a prime and gcd(a,p)=1 then
    - a^{p-1} mod p = 1

- Algorithms to find **Inverses** a^{-1} mod n
    1. search 1,...,n-1 until an a^{-1} is found with a.a^{-1} mod n
    2. if [[phi]](n) is known, then from Euler's Generalization

       $\mathbf{a^{-1} = a^{[[phi]](n)-1}}$ **mod n**
    3. otherwise use Extended Euclid's algorithm for inverse

1.6.3 Computing with Polynomials in GF(qn)

- have seen arithmetic modulo a prime number GF(p)
- also can do arithmetic modulo q over polynomials of degree n, which also form a **Galois Field GF(q^n)**
- its elements are polynomials of degree (n-1) or lower
  - $a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$
- have residues for polynomials just as for integers
  - $p(x)=q(x)d(x)+r(x)$
  - and this is unique if $deg[r(x)]<deg[d(x)]$
- if $r(x)=0$, then d(x) **divides** p(x), or is a **factor** of p(x)
- addition in GF(q^n) just involves summing equivalent terms in the polynomial modulo q (XOR if q=2)
  - $a(x)+b(x)=(a_{n-1}+b_{n-1})x^{n-1}+...+(a_1+b_1)x+(a_0+b_0)$

Multiplication with Polynomials in GF(qn)

- multiplication in GF(q^n) involves
  - multiplying the two polynomials together (cf longhand multiplication; here use shifts & XORs if q=2)
  - then finding the residue modulo a given **irreducible polynomial** of degree n
- an **irreducible polynomial** d(x) is a 'prime' polynomial, it has no polynomial divisors other than itself and 1
- modulo reduction of p(x) consists of finding some r(x) st: $p(x)=q(x)d(x)+r(x)$
  - nb. in GF(2^n) with $d(x)=x^3+x+1$ can do simply by replacing x^3 with $x+1$
- eg in GF(2^3) there are 8 elements:
  - $0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1$
- with irreducible polynomial $d(x)=x^3+x+1$* arithmetic in this field can be summarised as:
- can adapt GCD, Inverse, and CRT algorithms for GF(q^n)
  - [[phi]](p(x)) = 2^n-1 since every poly except 0 is relatively prime to p(x)
- arithmetic in GF(q^n) can be much faster than integer arithmetic, especially if the irreducible polynomial is carefully chosen
  - eg a fast implementation of GF(2^{127}) exists

- has both advantages and disadvantages for cryptography, calculations are faster, as are methods for breaking

RSA and the Chinese Remainder Theorem

- a significant improvement in decryption speed for RSA can be obtained by using the Chinese Remainder theorem to work modulo p and q respectively
  - since p,q are only half the size of R=p.q and thus the arithmetic is much faster
- CRT is used in RSA by creating two equations from the decryption calculation:

$$M = Cd \bmod R$$

as follows:

$$M1 = M \bmod p = (C \bmod p)d \bmod (p-1)$$

$$M2 = M \bmod q = (C \bmod q)d \bmod (q-1)$$

then the pair of equations

$$M = M1 \bmod p \qquad M = M2 \bmod q$$

has a unique solution by the CRT, given by:

$$M = [((M2 + q - M1)u \bmod q]\ p + M1$$

where

$$p.u \bmod q = 1$$

1.7 FINITE FIELDS

1.7.1 Groups, Rings and Field:

- **Group**: A set of elements that is closed with respect to some operation.
- Closed-> The result of the operation is also in the set
- The operation obeys:
  - ✓ Obeys associative law: **(a.b).c = a.(b.c)**
  - ✓ Has identity e: **e.a = a.e = a**
  - ✓ Has inverses **a-1: a.a-1 = e**
- **Abelian Group**: The operation is commutative

a.b = b.a

- Example: Z8, + modular addition, identity =0

Cyclic Group

Exponentiation: Repeated application of operator

- example: **a3 = a.a.a**
- Cyclic Group: Every element is a power of some fixed element, i.e., **b = ak** for some a and every b in group a is said to be a generator of the group
- Example: {1, 2, 4, 8} with mod 12 multiplication, the generator is 2.
- 20=1, 21=2, 22=4, 23=8, 24=4, 25=8

Ring:

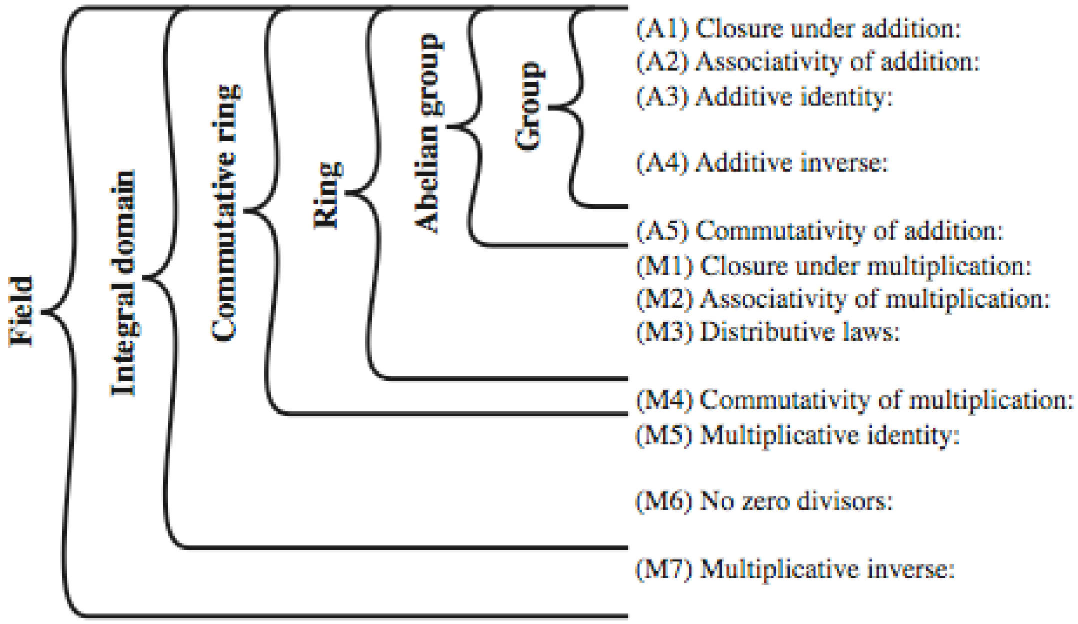
- A group with two operations: addition and multiplication
- The group is abelian with respect to addition: **a+b=b+a**
- Multiplication and additions are both associative:

  a+(b+c)=(a+b)+c

  a.(b.c)=(a.b).c
- Multiplication distributes over addition, **a.(b+c)=a.b+a.c**
- Commutative Ring: Multiplication is commutative, i.e., **a.b = b.a**
- Integral Domain: Multiplication operation has an identity and no zero divisors

Field:

An integral domain in which each element has a multiplicative inverse.


Field
Integral domain
Commutative ring
Ring
Abelian group
Group
(A1) Closure under addition:
(A2) Associativity of addition:
(A3) Additive identity:
(A4) Additive inverse:
(A5) Commutativity of addition:
(M1) Closure under multiplication:
(M2) Associativity of multiplication:
(M3) Distributive laws:
(M4) Commutativity of multiplication:
(M5) Multiplicative identity:
(M6) No zero divisors:
(M7) Multiplicative inverse:

1.7.2 Modular Arithmetic

- modular arithmetic is 'clock arithmetic'
- a **congruence** a = b mod n says when divided by *n* that a and b have the same remainder
    - 100 = 34 mod 11
    - usually have 0<=b<=n-1
    - -12mod7 = -5mod7 = 2mod7 = 9mod7
    - b is called the **residue** of a mod n
- can do arithmetic with integers modulo n with all results between 0 and n

Addition

a+b mod n

Subtraction

a-b mod n = a+(-b) mod n

Multiplication

a.b mod n

- derived from repeated addition
- can get a.b=0 where neither a,b=0
    - eg 2.5 mod 10

Division

a/b mod n

- is multiplication by inverse of b: a/b = a.b^{-1} mod n
- if n is prime b^{-1} mod n exists s.t b.b^{-1} = 1 mod n
    - eg 2.3=1 mod 5 hence 4/2=4.3=2 mod 5
- integers modulo n with addition and multiplication form a commutative ring with the laws of

Associativity : (a+b)+c = a+(b+c) mod n

Commutativity : a+b = b+a mod n

Distributivity : (a+b).c = (a.c)+(b.c) mod n

- also can chose whether to do an operation and then reduce modulo n, or reduce then do the operation, since reduction is a homomorphism from the ring of integers to the ring of integers modulo n
    - a+/-b mod n = [a mod n +/- b mod n] mod n
    - (the above laws also hold for multiplication)
- if n is constrained to be a prime number *p* then this forms a **Galois Field modulo p** denoted **GF(p)** and all the normal laws associated with integer arithmetic work

Greatest Common Divisor

- the greatest common divisor (a,b) of a and b is the largest number that divides evenly into both a and b
- **Euclid's Algorithm** is used to find the Greatest Common Divisor (GCD) of two numbers a and n, a<n
    - use fact if a and b have divisor d so does a-b, a-2b

```
GCD (a,n) is given by:
  let g0=n
     g1=a
     gi+1 = gi-1 mod gi
  when gi=0 then (a,n) = gi-1
eg find (56,98)
  g0=98
  g1=56
  g2 = 98 mod 56 = 42
  g3 = 56 mod 42 = 14
  g4 = 42 mod 14 = 0
hence (56,98)=14
```

Finite Fields or Galois Fields

- Finite Field: A field with finite number of elements
- Also known as Galois Field
- The number of elements is always a power of a prime number. Hence, denoted as GF(pn)
- GF(p) is the set of integers {0,1, …, p-1} with arithmetic operations modulo prime p
- Can do addition, subtraction, multiplication, and division without leaving the field GF(p)
- GF(2) = Mod 2 arithmetic
  GF(8) = Mod 8 arithmetic
- There is no GF(6) since 6 is not a power of a prime

Polynomial Arithmetic

$\boldsymbol{f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = \Sigma\, a_i x^i}$

1. Ordinary polynomial arithmetic:

- Add, subtract, multiply, divide polynomials,
- Find remainders, quotient.
- Some polynomials have no factors and are prime.

2. Polynomial arithmetic with mod p coefficients

3. Polynomial arithmetic with mod p coefficients and mod m(x) operations

Polynomial Arithmetic with Mod 2 Coefficients

- All coefficients are 0 or 1, e.g.,

  let $f(x) = x3 + x2$ and $g(x) = x2 + x + 1$

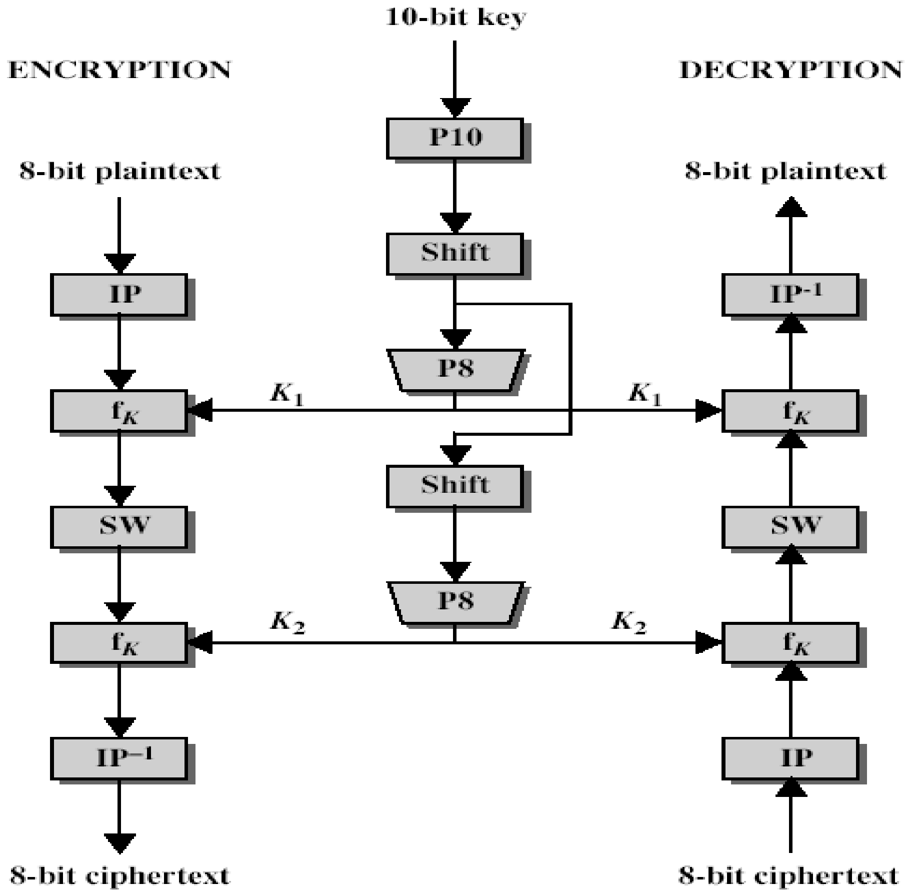
  $f(x) + g(x) = x3 + x + 1$

  $f(x)$ x $g(x) = x5 + x2$
- Polynomial Division: $f(x) = q(x)\, g(x) + r(x)$
- can interpret $r(x)$ as being a remainder
- $r(x) = f(x)$ mod $g(x)$
- if no remainder, say $g(x)$ divides $f(x)$
- if $g(x)$ has no divisors other than itself & 1 say it is irreducible (or prime) polynomial
- Arithmetic modulo an irreducible polynomial forms a finite field
- Can use Euclid’s algorithm to find gcd and inverses.

UNIT II

2.1 SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES)

The overall structure of the simplified DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.



The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function f_k again
- a permutation function that is the inverse of the initial permutation

The function f_k takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition1 of functions:

$$IP^{-1} \text{ o } f_{K2} \text{ o } SW \text{ o } f_{k1} \text{ o } IP$$

Which can also be written as

$$\text{Ciphertext} = IP^{-1} (f_{K2} (SW (f_{k1} (IP (\text{plaintext})))))$$

Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as

$$\text{Plaintext} = IP^{-1} (f_{K1} (SW (f_{k2} (IP (\text{ciphertext})))))$$

2.1.1 S-DES key generation

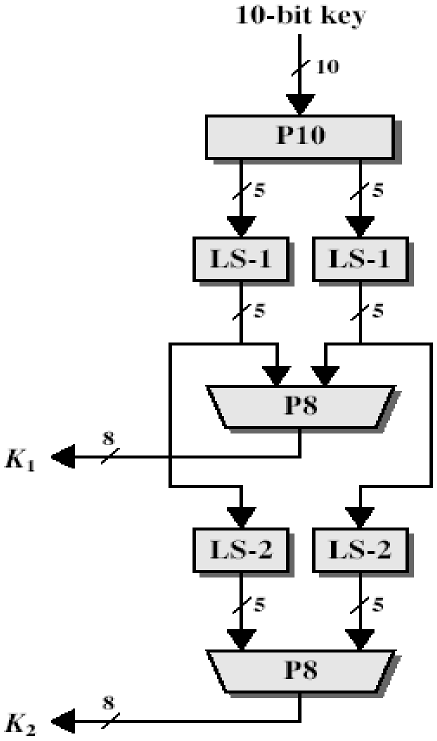


Figure: key generation for S-DES

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:
P10 (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, K2, k7, k4, k10 10, k1, k9, k8, k6) P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

2.1.2 S-DES encryption

Encryption involves the sequential application of five functions.

Initial and Final Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up.

Consider the plaintext to be 11110011.

Permuted output = 10111101

At the end of the algorithm, the inverse permutation is used:

IP –1							
4	1	3	5	7	2	8	6

The Function f_k

The most complex component of S-DES is the function f_k, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f K, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

Where SK is a subkey and ⊕ is the bit-by-bit exclusive-OR function.

e.g., permuted output = 1011 1101 and suppose F (1101, SK) = (1110) for some key SK.

Then f K(10111101) = 1011⊕1110, 1101

= 01011101

We now describe the mapping F. The input is a 4-bit number (n1 n2 n3 n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

e.g., R= 1101

E/P output = 11101011

It is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey K1 = (k11, k12 12, k13 13, k14 14, k15 15, k16 16, k17 17, k18) is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits: $\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if (p0,0 p0,3) =) (00) and (p0,1 p0,2) = (10), then the output is from row 0, column 2 of S0, which is 3, or (11) in) binary. Similarly, (p1,0 p1,3) and (p1,1 p1,2) are used to index into a row and column of S1 to produce an additional 2 bits. Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

2.1.3 The Switch Function

The function f K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext.

2.2 BLOCK CIPHER PRINCIPLES

Virtually, all symmetric block encryption algorithms in current use are based on a structure referred to as Fiestel block cipher. For that reason, it is important to examine the design principles of the Fiestel cipher. We begin with a **comparison of stream cipher with block cipher.**

- **A stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. E.g, vigenere cipher. **A block cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically a block size of 64 or 128 bits is used.

2.2.1 Block cipher principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently
- block ciphers look like an extremely large substitution
- would need table of 264 entries for a 64-bit block
- instead create from smaller building blocks
- using idea of a product cipher in 1949 Claude Shannon introduced idea of substitution-permutation (S-P) networks called modern substitution-transposition product cipher these form the basis of modern block ciphers
- S-P networks are based on the two primitive cryptographic operations we have seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
- provide *confusion* and *diffusion* of message
- **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

2.2.2 Feistel cipher structure

The input to the encryption algorithm are a plaintext block of length 2w bits and a key K. the plaintext block is divided into two halves L_0 and R_0. The two halves of the data pass through „n‟ rounds of processing and then combine to produce the ciphertext block. Each round „i‟ has inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as the subkey K_i, derived from the overall key K. in general, the subkeys K_i are different from K and from each other.

All rounds have the same structure. A substitution is performed on the left half of the data (as similar to S-DES). This is done by applying a round function F to the right half of the data and then taking the 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 parameterized by the round subkey k_i. Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network.

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

- **Block size** - Increasing size improves security, but slows cipher
- **Key size** - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **Number of rounds** - Increasing number improves security, but slows cipher
- **Subkey generation** - Greater complexity can make analysis harder, but slows cipher
- **Round function** - Greater complexity can make analysis harder, but slows cipher
- **Fast software en/decryption & ease of analysis -** are more recent concerns for practical use and testing.

The process of decryption is essentially the same as the encryption process. The rule is as follows: use the cipher text as input to the algorithm, but use the subkey k_i in reverse order. i.e., k_n in the first round, k_{n-1} in second round and so on. For clarity, we use the notation LE_i and RE_i for data traveling through the decryption algorithm. The diagram below indicates that, at each round, the intermediate value of the decryption process is same (equal) to the corresponding value of the encryption process with two halves of the value swapped.

i.e., $RE_i \| LE_i$ (or) equivalently $RD_{16-i} \| LD_{16-i}$

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

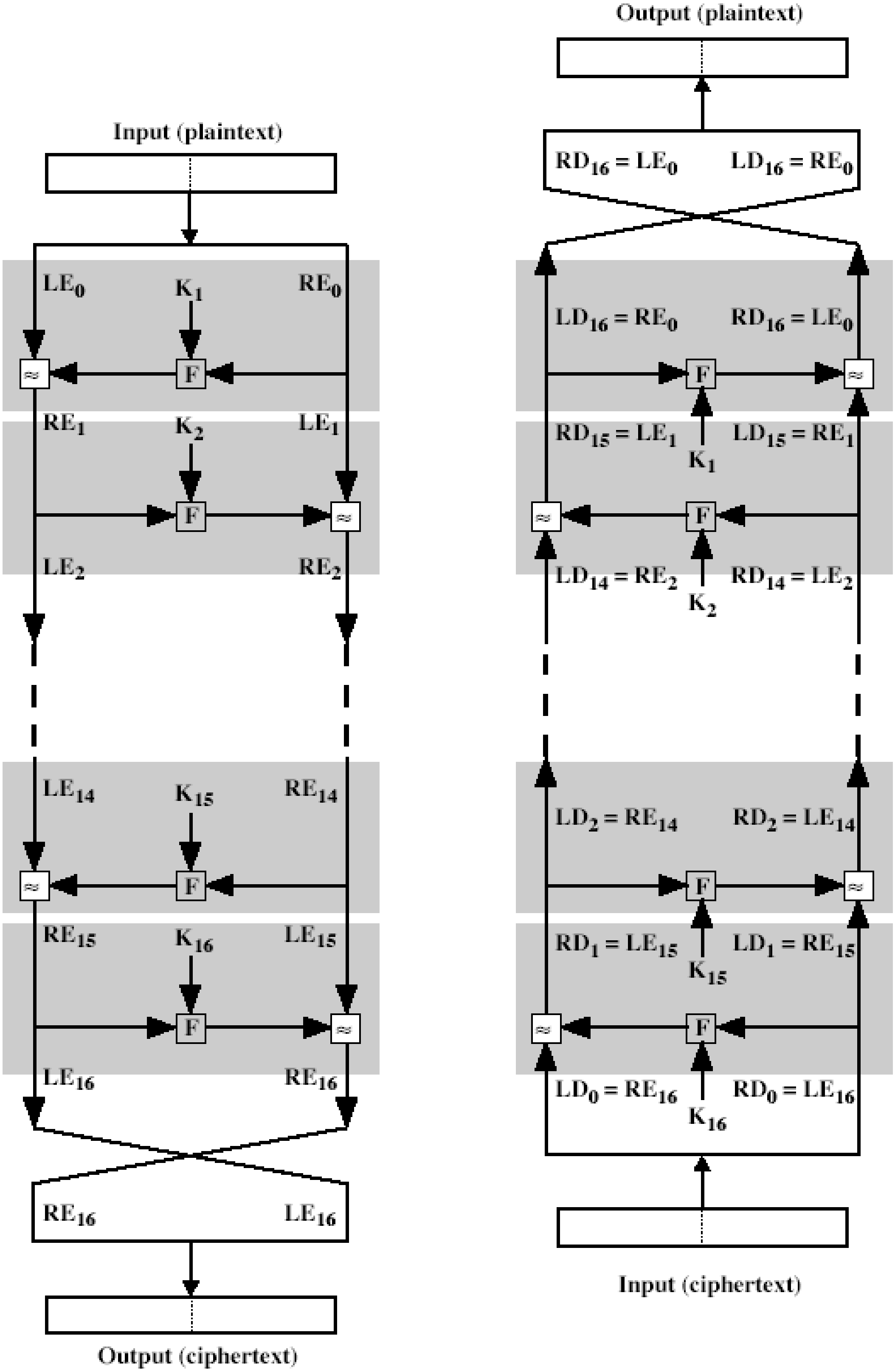


Fig2.2.2.1: Feistel encryption and decryption

Now we will see how the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First consider the encryption process,

$LE_{16} = RE_{15}$

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

decryption side,

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

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

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

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

$= LE_{15}$

Therefore, $LD_1 = RE_{15}$ $RD_1 =$

LE_{15}

In general, for the i^{th} iteration of the encryption algorithm,

$LE_i = RE_{i-1}$

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

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

2.3 MULTIPLE ENCRYPTION & DES

- clear a replacement for DES was needed
  - ✓ theoretical attacks that can break it
  - ✓ demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

2.3.1 Double-DES

- could use 2 DES encrypts on each block

$$\mathbf{C = E_{K2}(E_{K1}(P))}$$

- issue of reduction to single stage and have “meet-in-the-middle” attack
  - ✓ works whenever use a cipher twice
  - ✓ since $X = E_{K1}(P) = D_{K2}(C)$
  - ✓ attack by encrypting P with all keys and store
  - ✓ then decrypt C with keys and match X value
  - ✓ can show takes $O(2^{56})$ steps

2.3.2 Triple-DES with Two-Keys

- hence must use 3 encryptions
  - ✓ would seem to need 3 distinct keys
- but can use 2 keys with E-D-E sequence

$$\mathbf{C = E_{K1}(D_{K2}(E_{K1}(P)))}$$

  - ✓ nb encrypt & decrypt equivalent in security
  - ✓ if K1=K2 then can work with single DES
- standardized in ANSI X9.17 & ISO8732
- no current known practical attacks

2.3.3 Triple-DES with Three-Keys

- although are no practical attacks on two key Triple-DES have some indications
- can use Triple-DES with Three-Keys to avoid even these

$$\mathbf{C = E_{K3}(D_{K2}(E_{K1}(P)))}$$

- has been adopted by some Internet applications
- eg PGP, S/MIME

2.4 Modes of Operation

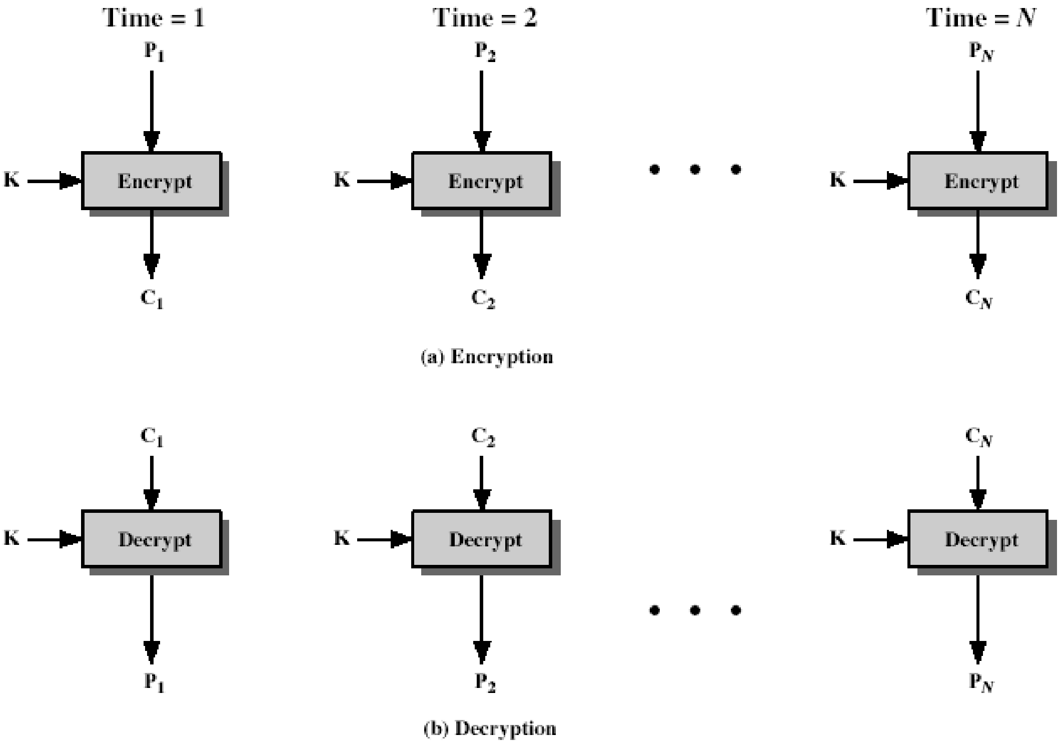
- block ciphers encrypt fixed size blocks
- eg. DES encrypts 64-bit blocks with 56-bit key
- need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- subsequently 5 defined for AES & DES
- have **block** and **stream** modes

2.4.1 Electronic Codebook Book (ECB)

- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks

$$\mathbf{C_i = DES_{K1}(P_i)}$$

- uses: secure transmission of single values



Electronic Codebook Book (ECB)

Advantages and Limitations of ECB

- message repetitions may show in ciphertext
  - ✓ if aligned with message block
  - ✓ particularly with data such graphics
  - ✓ or with messages that change very little, which become a code-book analysis problem
- weakness is due to the encrypted message blocks being independent
- main use is sending a few blocks of data

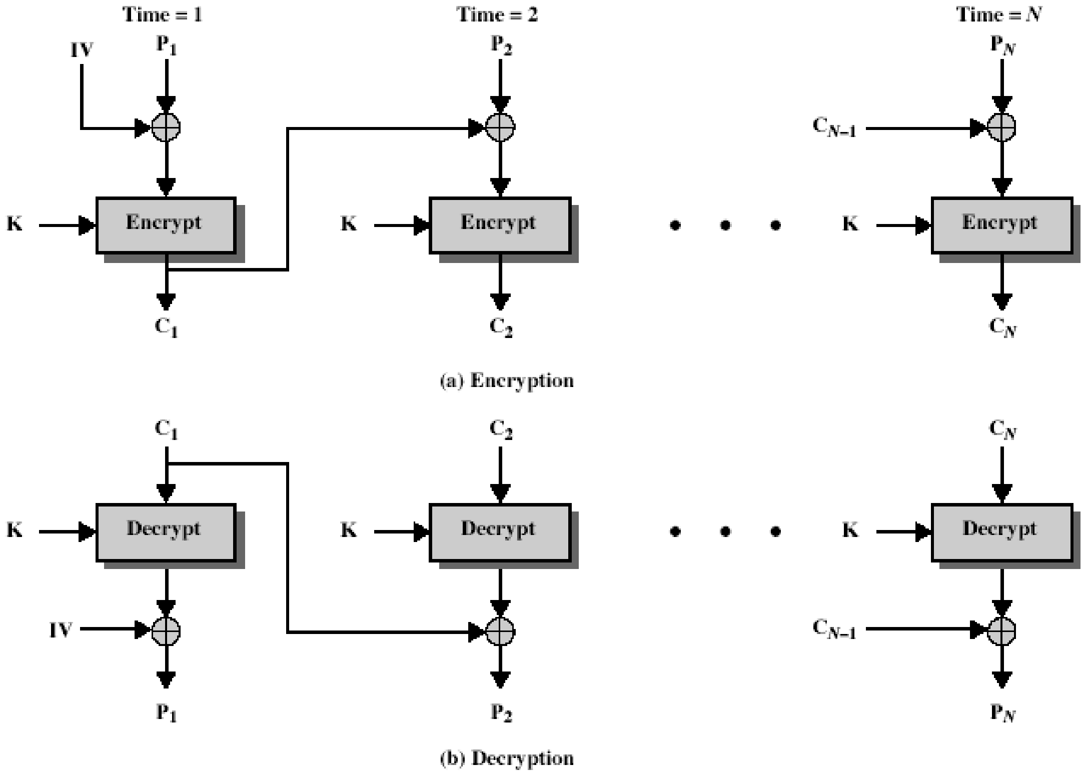
2.4.2 Cipher Block Chaining (CBC)

- message is broken into blocks
- linked together in encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

$$\mathbf{C_i = DES_{K1}(P_i\ \ XOR\ \ C_{i-1})}$$

$$\mathbf{C_{-1} = IV}$$

- uses: bulk data encryption, authentication



Cipher Block Chaining (CBC)

2.4.3 Message Padding

- at end of message must handle a possible last short block
  - ✓ which is not as large as blocksize of cipher
  - ✓ pad either with known non-data value (eg nulls)
  - ✓ or pad last block along with count of pad size
    - o eg. [b1 b2 b3 0 0 0 0 5]
    - o means have 3 data bytes, then 5 bytes pad+count
  - ✓ this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks
- need **Initialization Vector** (IV)
  - ✓ which must be known to sender & receiver
  - ✓ if sent in clear, attacker can change bits of first block, and change IV to compensate
  - ✓ hence IV must either be a fixed value (as in EFTPOS)
  - ✓ or must be sent encrypted in ECB mode before rest of Message

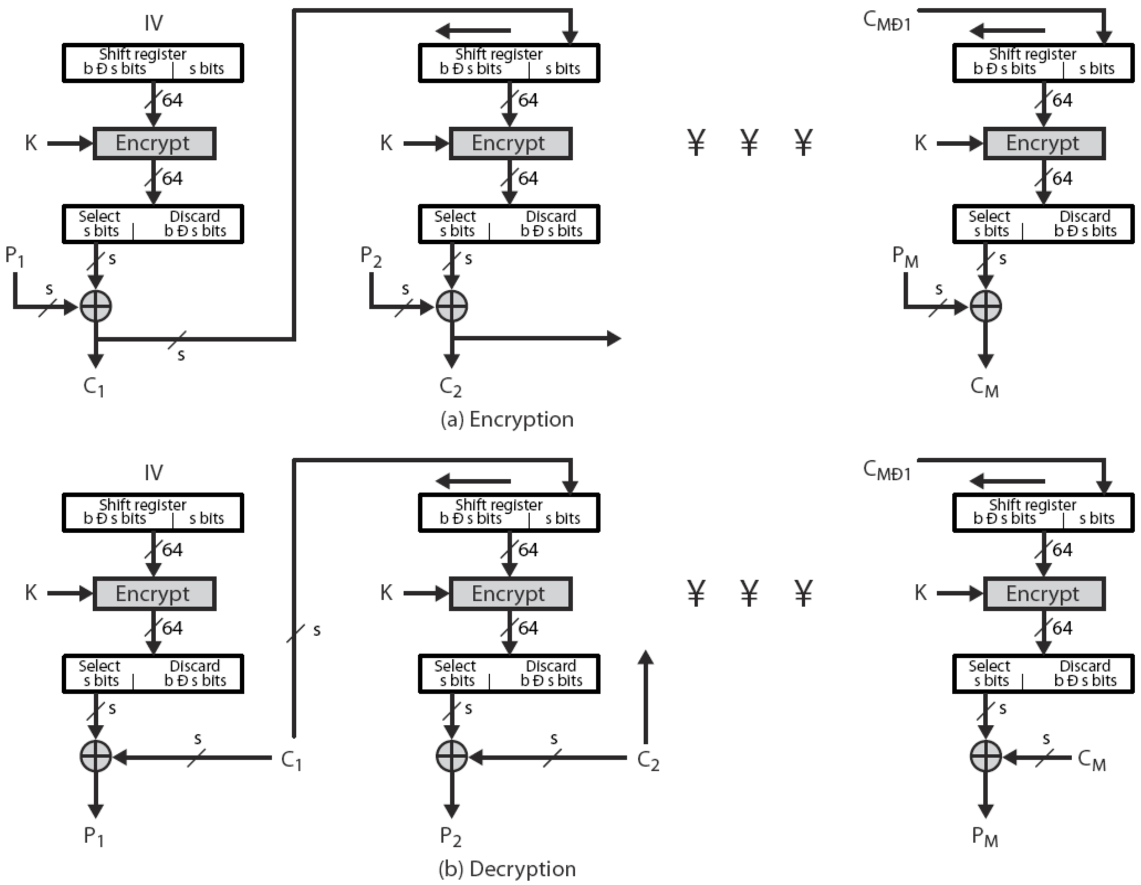
2.4.4 Cipher FeedBack (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - ✓ denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)

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

$$\mathbf{C_{-1} = IV}$$

- uses: stream data encryption, authentication



(a) Encryption

(b) Decryption

Cipher FeedBack (CFB)

Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- limitation is need to stall while do block encryption after every n-bits
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propogate for several blocks after the error

2.4.5 Output FeedBack (OFB)

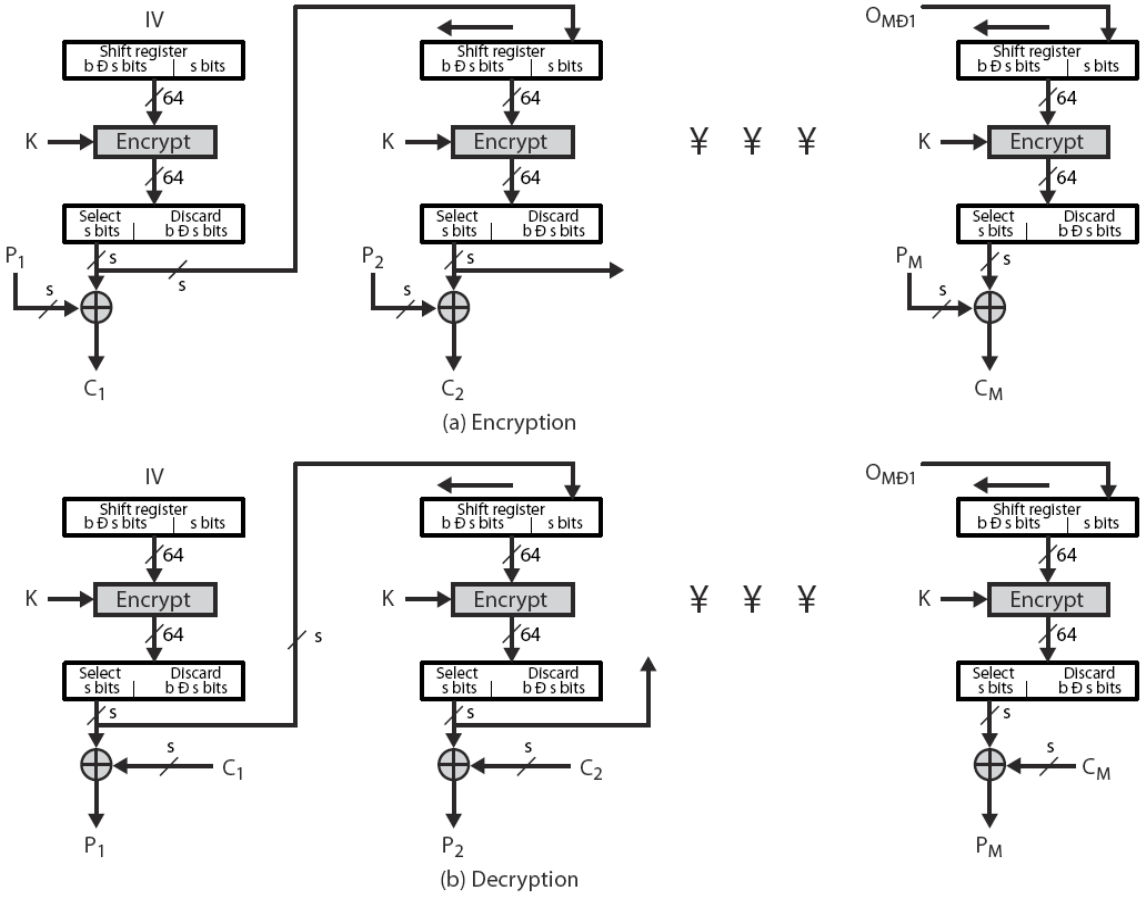
- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance

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

$$\mathbf{O_i = DES_{K1}(O_{i-1})}$$

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

- uses: stream encryption on noisy channels



Output FeedBack (OFB)

Advantages and Limitations of OFB

- bit errors do not propagate
- more vulnerable to message stream modification
- a variation of a Vernam cipher
  - ✓ hence must **never** reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- originally specified with m-bit feedback
- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

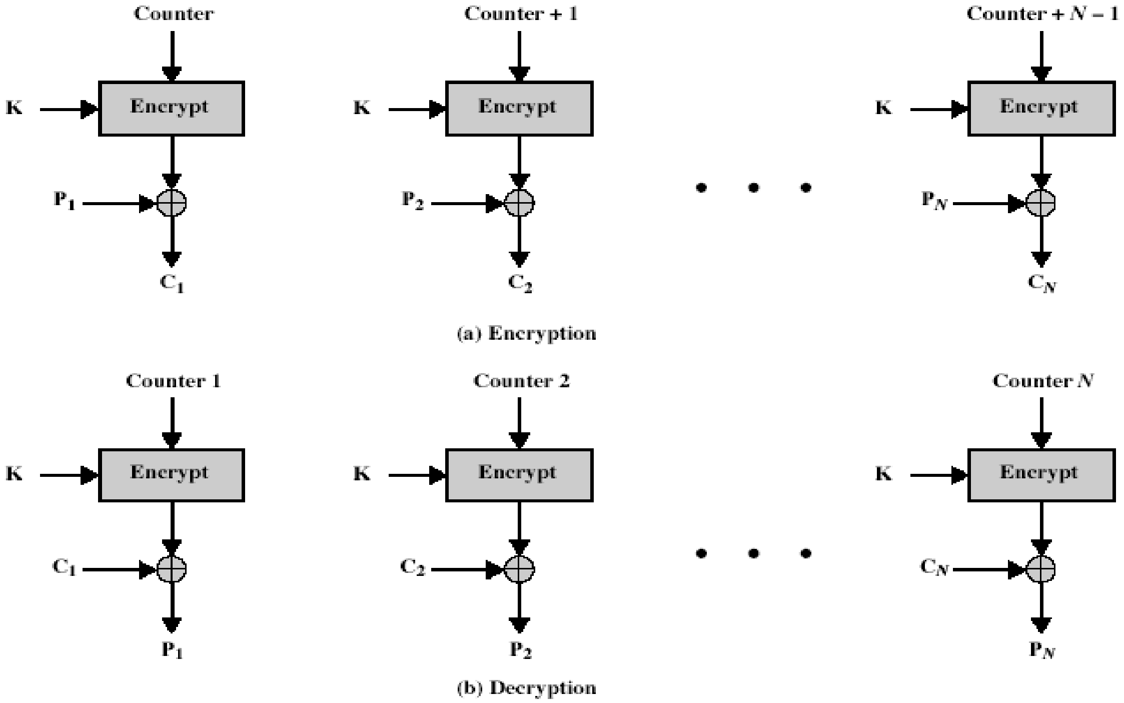
2.4.6 Counter (CTR)

- a "new" mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

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

$$\mathbf{Oi = DES_{K1}(i)}$$

- uses: high-speed network encryptions



Counter (CTR)

Advantages and Limitations of CTR

- efficiency
  - ✓ can do parallel encryptions in h/w or s/w
  - ✓ can preprocess in advance of need
  - ✓ good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

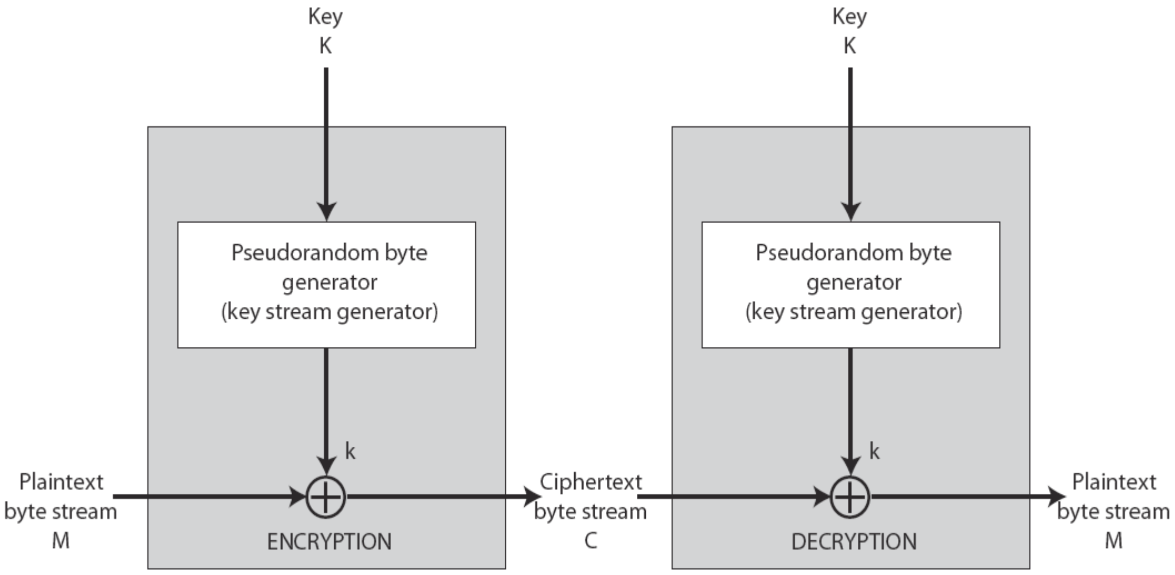
2.5 STREAM CIPHERS

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message

$$C_i = M_i \text{ XOR StreamKey}_i$$

- but must never reuse stream key
- otherwise can recover messages (cf book cipher)

2.5.1 Stream Cipher Structure


Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
ENCRYPTION
Ciphertext byte stream C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream M

2.5.2 Stream Cipher Properties

- some design considerations are:
  - ✓ long period with no repetitions
  - ✓ statistically random
  - ✓ depends on large enough key
  - ✓ large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

2.6 RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

2.6.1 RC4 Key Schedule

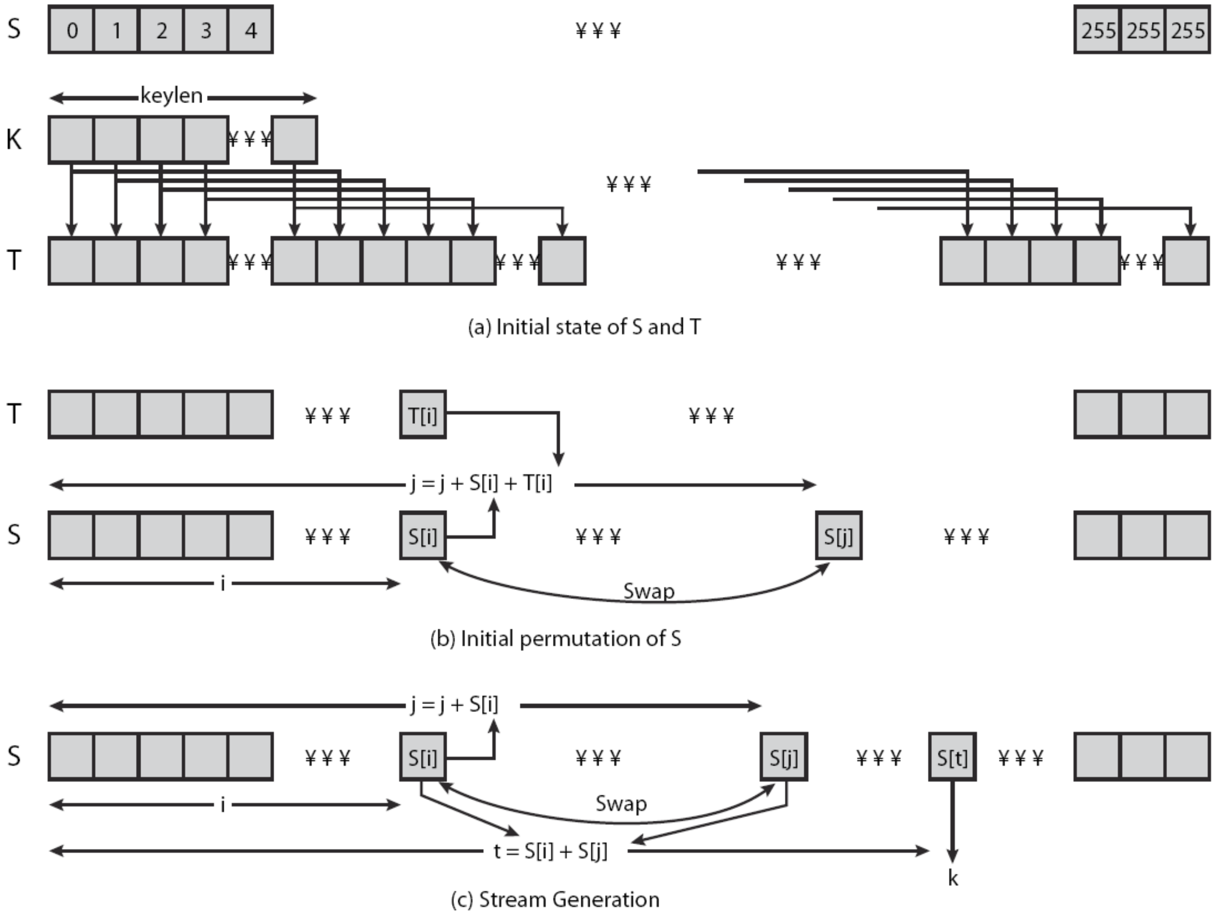
- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher

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

2.6.2 RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] with next byte of message to en/decrypt

```
$i = j = 0$
for each message byte $M_i$
    $i = (i + 1) \pmod{256}$
    $j = (j + S[i]) \pmod{256}$
    swap(S[i], S[j])
    $t = (S[i] + S[j]) \pmod{256}$
    $C_i = M_i$ XOR $S[t]$
```



(a) Initial state of S and T

(b) Initial permutation of S

(c) Stream Generation

RC4 Overview

2.6.3 RC4 Security

- claimed secure against known attacks
  - ✓ have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- have a concern with WEP, but due to key handling rather than RC4 itself.

2.7 PRINCIPLES OF PUBLIC KEY CRYPTOGRAPHY

The concept of public key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption.

- Key distribution under symmetric key encryption requires either (1) that two communicants already share a key, which someone has been distributed to them or (2) the use of a key distribution center.
- Digital signatures.

2.7.1 Public key cryptosystems

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

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

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

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

The essential steps are the following:

- Each user generates a pair of keys to be used for encryption and decryption of messages.
- 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.
- If A wishes to send a confidential message to B, A encrypts the message using B's public key.
- When B receives the message, it decrypts using its private key. No other recipient can decrypt the message because only B knows B's private key.

With this approach, all participants have access to public keys and private keys are generated locally by each participant and therefore, need not be distributed. As long as a system controls its private key, its incoming communication is secure.

Let the plaintext be X=[X1, X2, X3, …,Xm] where m is the number of letters in some finite alphabets. Suppose A wishes to send a message to B. B generates a pair of keys: a public key KU_b and a private key KR_b. KR_b is known only to B, whereas KU_b is publicly available and therefore accessible by A.

With the message X and encryption key KU_b as input, A forms the cipher text

Y=[Y1, Y2, Y3, … Yn]., i.e., Y=E KU_b(X)

The receiver can decrypt it using the private key KR_b. i.e., X=D KR_b(). The encrypted message serves as a **digital signature.**

It is important to emphasize that the encryption process just described does not provide confidentiality. There is no protection of confidentiality because any observer can decrypt the message by using the sender’s public key.

It is however, possible to provide both the authentication and confidentiality by a double use of the public scheme.

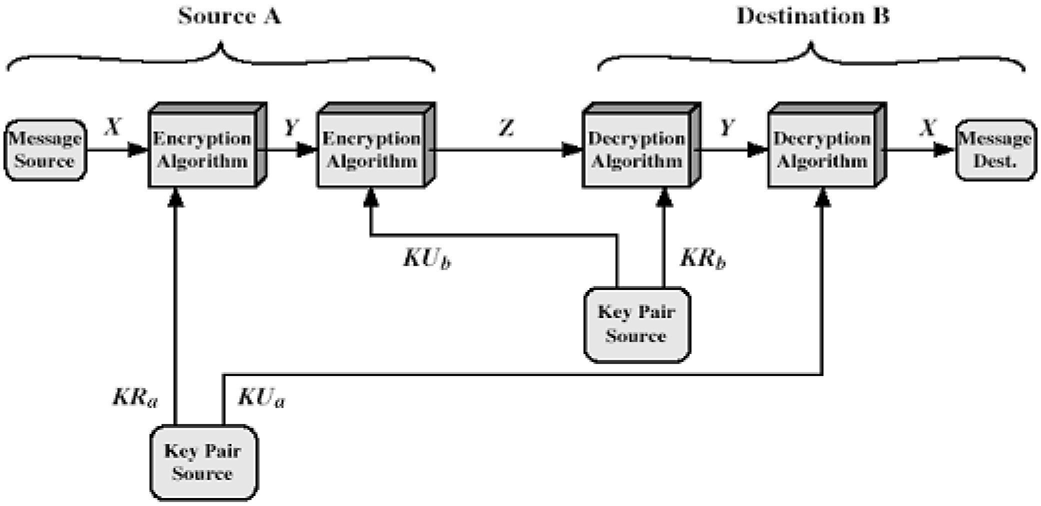


Fig.2.7.1.1 Public Key Cryptosystem

Ciphertext Z = EKU_b [EKR_a (X)]

Plaintext X = EKU_a[EKR_b (Y)]

Initially, the message is encrypted using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus confidentiality is provided.

2.7.2 Requirements for public key cryptography

- It is computationally easy for a party B to generate a pair [KU_b , KR_b].
- It is computationally easy for a sender A, knowing the public key and the message to be encrypted M, to generate the corresponding ciphertext: $C=EKU_b(M)$.
- It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: $M = DKR_b (C) = DKR_b [EKU_b (M)]$
- It is computationally infeasible for an opponent, knowing the public key KU_b, to determine the private key KR_b.
- It is computationally infeasible for an opponent, knowing the public key KU_b, and a ciphertext C, to recover the original message M.
- The encryption and decryption functions can be applied in either order:

$$M = EKU_b [DKR_b (M) = DKU_b [EKR_b (M)]$$

Public key cryptanalysis

Public key encryption scheme is vulnerable to a brute force attack. The counter measure is to use large keys.

2.8 RSA Algorithm

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

$$C = M^e \bmod n$$

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

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

$= M^{ed} \bmod n$

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

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

Let us focus on the first requirement. We need to find the relationship of the form:

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

A corollary to Euler's theorem fits t0068e bill: Given two prime numbers p and q and two integers, n and m, such that n=pq and 0<m<n, and arbitrary integer k, the following relationship holds

$$_{m}k\Phi(n) +1 _{= m}k(p-1)(q-1) +1 _{= m \bmod n}$$

where Φ(n) – Euler totient function, which is the number of positive integers less than n and relatively prime to n. we can achieve the desired relationship, if ed=kΦ(n)+1

This is equivalent to saying:

$$\mathbf{ed \equiv 1 \bmod \Phi(n)}$$
$$\mathbf{d = e^{-1} \bmod \Phi(n)}$$

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

The steps involved in RSA algorithm for generating the key are

- Select two prime numbers, p = 17 and q = 11.
- Calculate n = p*q = 17*11 = 187
- Calculate Φ(n) = (p-1)(q-1) = 16*10 = 160.
- Select e such that e is relatively prime to Φ(n) = 160 and less than Φ(n); we choose e = 7.
- Determine d such that ed ≡ 1 mod Φ(n) and d<160. the correct value is d = 23, because 23*7 = 161 = 1 mod 160.

2.8.1 The RSA algorithm:

Key Generation

Select p, q		p ,q both prime $p \neq q$
Calculate $n = p \times q$		
Calculate $\phi(n) = (p - l)(q - 1)$		
Select integer	e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate	d	$d = e^{-1}{}_{\text{mod}}\ \phi(n)$
Public key		KU = { e,n}
Private key		KR = {d,n}

Encryption

Plaintext	$M < n$
Ciphertext	$C = M^e \pmod n$

Decryption

Ciphertext	C
Plaintext	$M = C^d \pmod n$

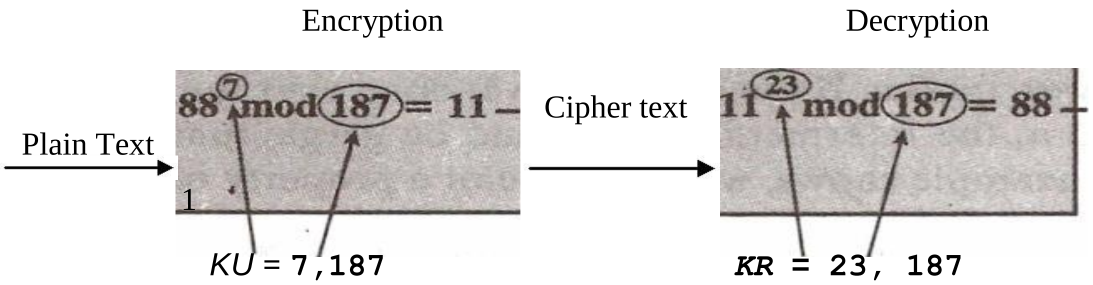


Figure : Example of RSA Algorithm

2.8.2 Security of RSA

There are three approaches to attack the RSA:

- brute force key search (infeasible given size of numbers)
- mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
- timing attacks (on running time of decryption)

Factoring Problem

Mathematical approach takes 3 forms:

- Factor n = p*q, hence find Φ(n) and then d.
- Determine Φ(n)directly without determining p and q and find d.
- Find d directly, without first determination Φ(n).

Timing attacks

It has been proved that the opponent can determine a private key by keeping track of how long a computer takes to decipher messages. Although the timing attack is a serious threat, there are simple countermeasures that can be used:

- Constant exponentiation time – ensures that all exponentiations take the same amount of time before returning a result.
- Random delay – better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack.
- Blinding – multiply the ciphertext by a random number before performing exponentiation.

UNIT III

3.1 DIFFIE-HELLMAN KEY EXCHANGE

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

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. First, we define a primitive root of a prime number p as one whose power generate all the integers from 1 to (p-1) i.e., if ‘a’ is a primitive root of a prime number p, then the numbers a mod p, a^2 mod p, … a^{p-1} mod p are distinct and consists of integers from 1 to (p-1) in some permutation. For any integer ‘b’ and a primitive root ‘a’ of a prime number ‘p’, we can find a unique exponent ‘i’ such that

$$\mathbf{b \equiv a^i \bmod p,} \text{ where } 0 \leq i \leq (p-1)$$

The exponent ‘i’ is referred to as discrete logarithm. With this background, we can define Diffie Hellman key exchange as follows:

There are publicly known numbers: a prime number ‘q’ and an integer α that is primitive root of q. suppose users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha$

^{XB} mod q. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $\mathbf{K = (Y_B)^{XA} \bmod q}$
and User B computes the key as $\mathbf{K = (Y_A)^{XB} \bmod q}$
These two calculations produce identical results.

$$\begin{aligned} K &= (Y_B)^{XA} \bmod q \\ &= (\alpha^{XB} \bmod q)^{XA} \bmod q \\ &= (\alpha^{XB})^{XA} \bmod q \\ &= (\alpha^{XA})^{XB} \bmod q \\ &= (\alpha^{XA} \bmod q)^{XB} \bmod q \\ &= (Y_A)^{XB} \bmod q \end{aligned}$$

The result is that two sides have exchanged a secret key.

The security of the algorithm lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

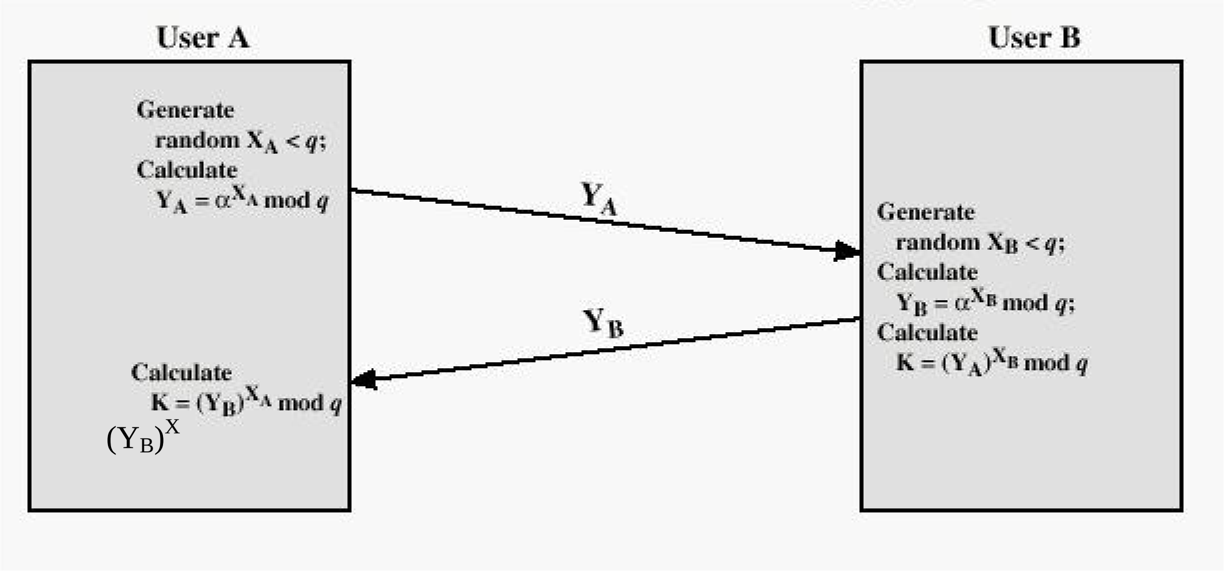

User A
User B
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$
Y_B
Calculate $K = (Y_B)^{X_A} \bmod q$
$(Y_B)^X$

Fig3.1.1: Diffie Hellman Key exchange

The protocol depicted in figure is insecure against a **man-in-the-middle attack**. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

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

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2.

All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message M: E(K2, M).
2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth

simply wants to eavesdrop on the communication without altering it.

In the second case, Darth wants to modify the message going to Bob. The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

3.1.1 Analog of Diffie-Hellman Key Exchange

Key exchange using elliptic curves can be done in the following manner. First pick a large integer q, which is either a prime number p or an integer of the form 2^m and elliptic curve parameters a and b. This defines the elliptic group of points $E_q(a, b)$. Next, pick a base point G = (x_1, y_1) in $E_p(a, b)$ whose order is a very large value n. The order n of a point G on an elliptic curve is the smallest positive integer n such that nG = O. $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants.

1. A selects an integer n_A less than n. This is A's private key. A then generates a public key $P_A = n_A$ x G; the public key is a point in Eq(a, b).
2. B similarly selects a private key n_B and computes a public key P_B.
3. A generates the secret key K = n_A x P_B. B generates the secret key K = n_B x P_A.

3.2 ELLIPTIC CURVE CRYPTOGRAPHY

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, we need to find a "hard problem" corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation Q = kP where Q, P in $E_p(a, b)$ and k < p. It is relatively easy to calculate Q given k and P, but it is relatively hard to determine k given Q and P. This is called the discrete logarithm problem for elliptic curves.

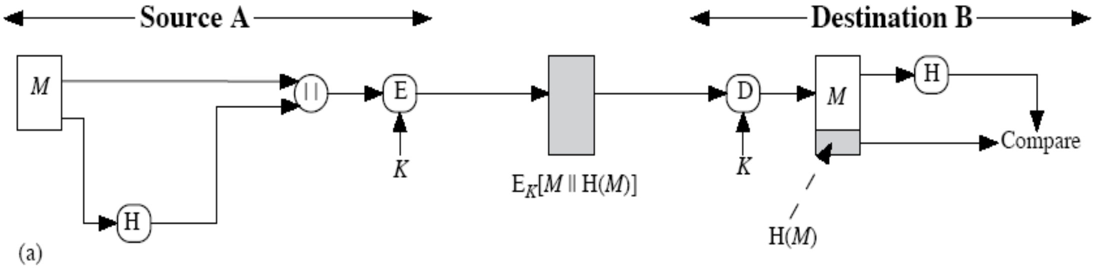
Because 9P = (4, 5) = Q, the discrete logarithm Q = (4, 5) to the base P = (16, 5) is k = 9. In a real application, k would be so large as to make the brute-force approach infeasible.

3.3 HASH FUNCTIONS

A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value.

There are varieties of ways in which a hash code can be used to provide message authentication, as follows:

□ The message plus the hash code is encrypted using symmetric encryption. This is identical to that of internal error control strategy. Because encryption is applied to the entire message plus the hash code, confidentiality is also provided.



N Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

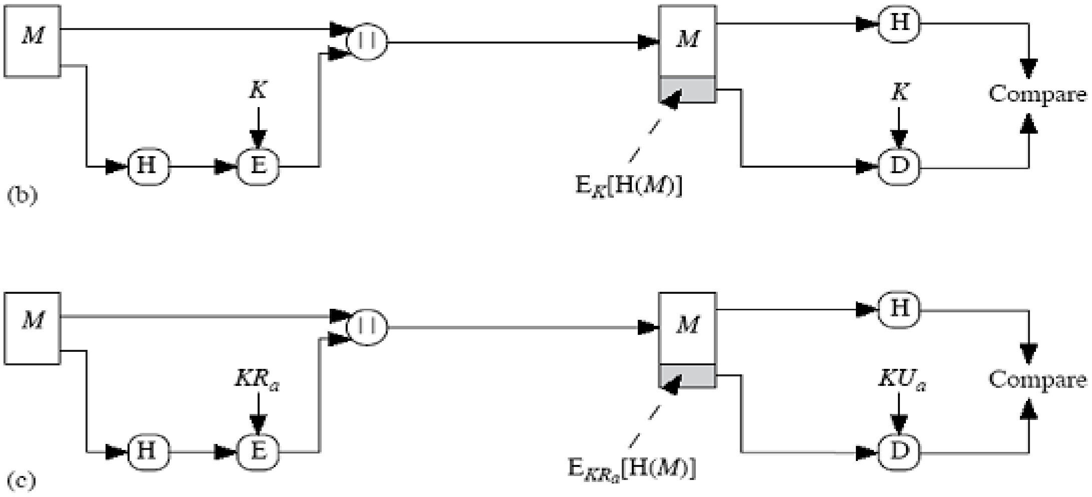
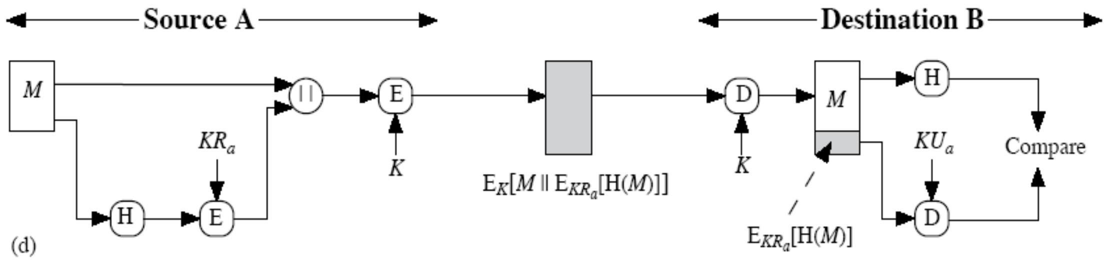
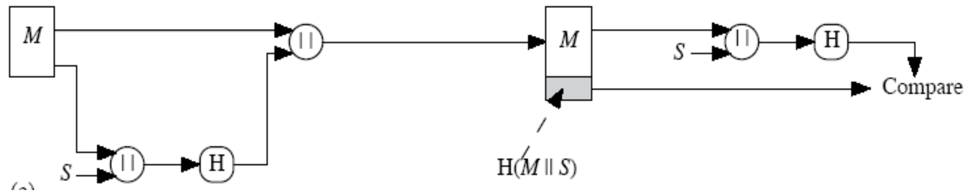


Fig.3.3.1: Basic Use of Hash Functions

c) Only the hash code is encrypted, using the public key encryption and using the sender's private key. It provides authentication plus the digital signature.

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





e) This technique uses a hash function, but no encryption for message authentication. This technique assumes that the two communicating parties share a common secret value ‘S’. The source computes the hash value over the concatenation of M and S and appends the resulting hash value to M.

f) Confidentiality can be added to the previous approach by encrypting the entire message plus the hash code.

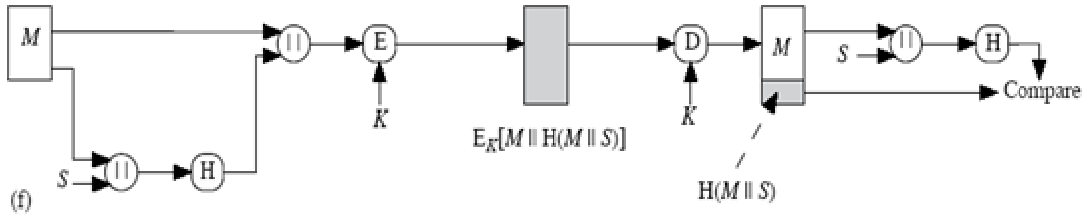


Fig3.3.2: Basic Use of Hash Functions

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

$$h = H(M)$$

where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed orknown to be correct. The receiver authenticates that message by recomputing the hashvalue.

3.3.1 Requirements for a Hash Function

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

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

The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used (Figures b and c).

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

3.4 SIMPLE HASH FUNCTIONS

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

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

$$\mathbf{C_i = b_{i1} \oplus b_{i1} \oplus ... \oplus b_{im}}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq n$ m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block $\oplus$ = XOR operation

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.

3.5 Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted message M, then an opponent would need to find an M' such that H(M') = H(M) to substitute another message and fool the receiver.

On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message. However, a different sort of attack is possible, based on **the birthday paradox** The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key.

1. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. (fraudulent message
2. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
3. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32} .

Block Chaining Techniques

Divide a message M into fixed-size blocks M_1,M_2,..., M_N and use a symmetric encryption system such as DES to compute the hash code G as follows:

H_o = initial value

Hi = EMi [Hi-1]

G = H_N

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings.

Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long:

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
2. Construct any desired message in the form Q_1, Q_2,..., Q_{N2}.
3. Compute for $H_i = EQ_i [H_{i-1}]$for $1 \leq i \leq (N-2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $E_X[H_{N-2}.]$ Generate an additional $2^{m/2}$ random blocks; for each block Y, compute $D_Y[G]$, where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $E_X [H_{N-2}] = D_Y[G]$.
6. Form the message Q_1, Q_2,..., Q_{N-2}, X, Y. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a **meet-in-the-middle attack**.

3.6 Security of Hash Functions and Macs

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

Brute-Force Attacks

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

Hash Functions

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

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

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

One way	$_{2}n$
Weak collision resistance	$_{2}n$
Strong collision resistance	$_{2}n/2$

Message Authentication Codes

A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs.. To attack a hash code, we can proceed in the following way. Given a fixed message x with n-bit hash code h = H(x), a brute-force method of finding a collision is to pick a random bit string y and check if H(y) = H(x). The attacker can do this repeatedly off line. To proceed, we need to state the desired security property of a MAC algorithm, which can be expressed as follows:

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

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

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

Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.

3.7 HASH FUNCTIONS

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, and is the structure of most hash functions in use today, including SHA and Whirlpool.

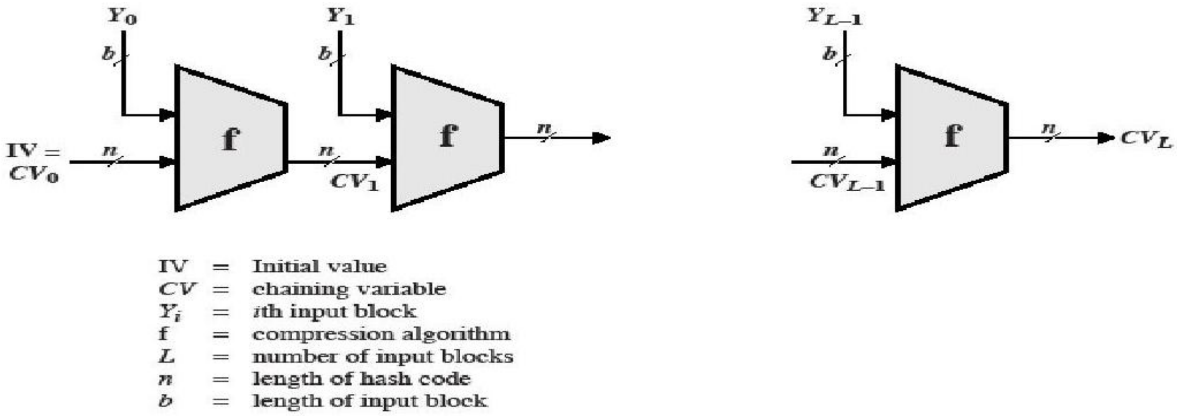

Y_0
b
Y_1
b
Y_{L-1}
b
IV = CV_0
n
f
n
CV_1
f
n
n
CV_{L-1}
f
n
CV_L
IV = Initial value
CV = chaining variable
Y_i = ith input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

Fig.3.7.1: General Structure of Secure Hash Code

The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the samevalue.

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

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

where the input to the hash function is a message M consisting of the blocks Y_o, Y_1,..., $Y_{L-1.}$ The structure can be used to produce a secure hash function to operate on a message of any length.

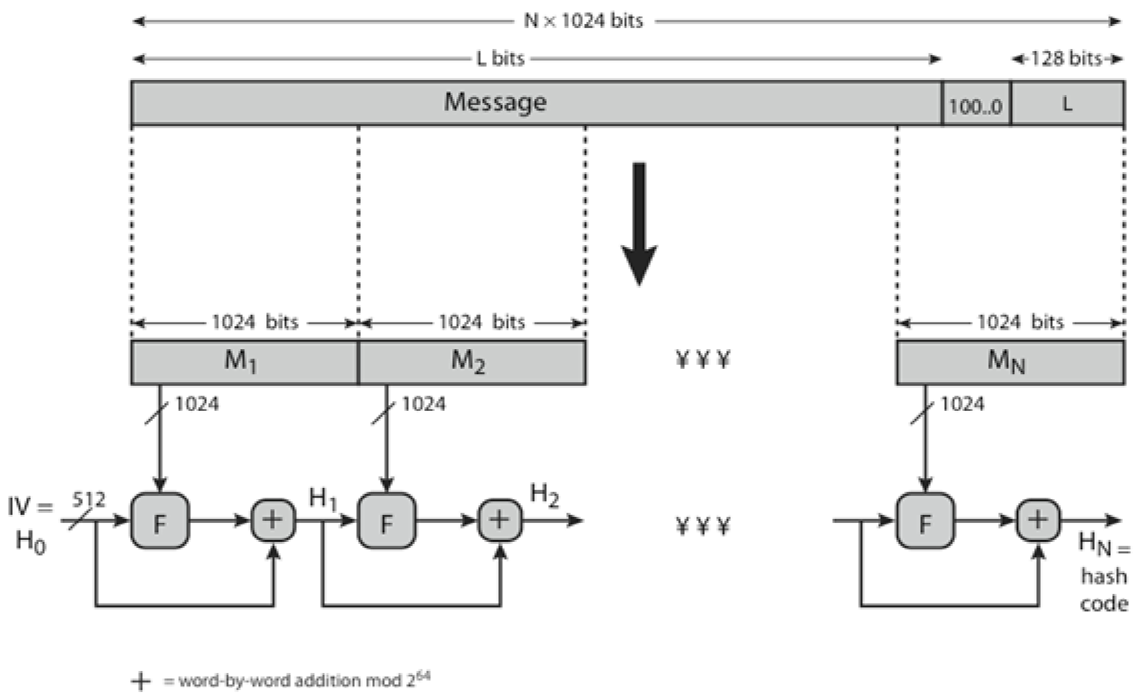
Message Authentication Codes

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs. Further, far less work has been done on developing such attacks.

3.8 SECURE HASH ALGORITHM

- SHA originally designed by NIST & NSA in 1993 was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - ✓ standard is FIPS 180-1 1995, also Internet RFC3174
  - ✓ nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

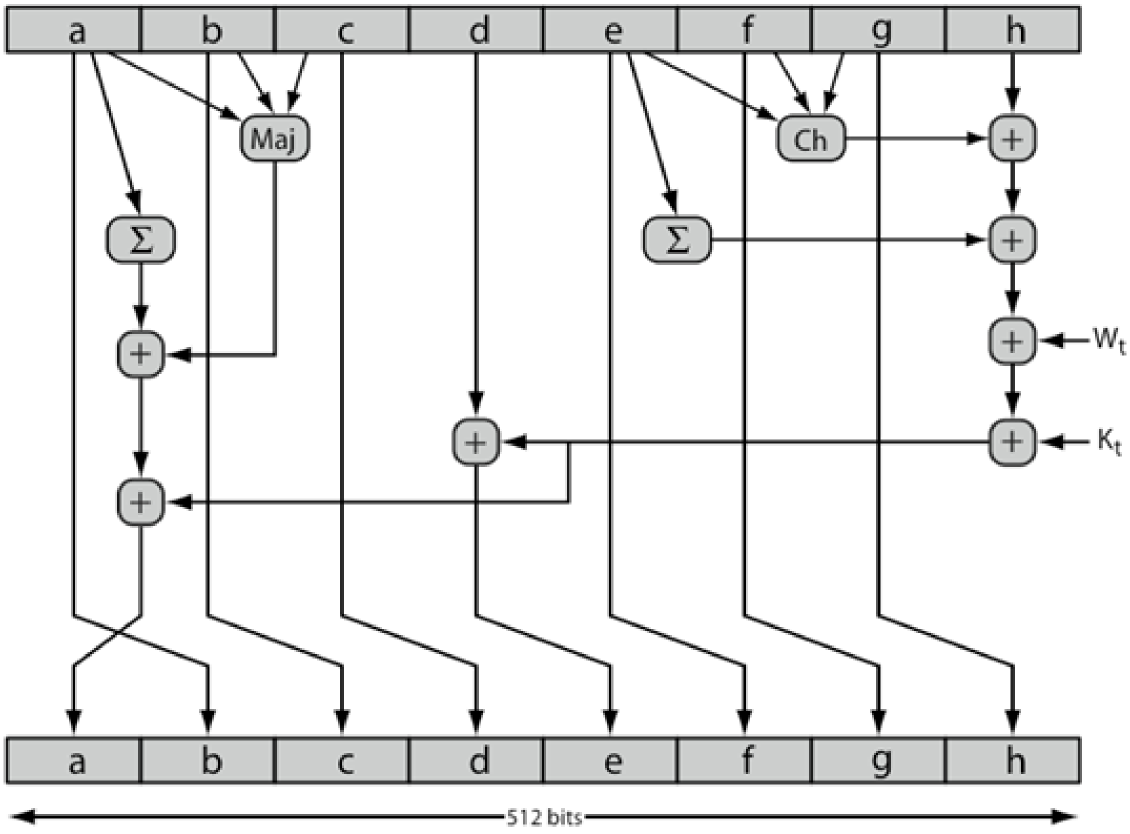
3.8.1 SHA-512 Overview



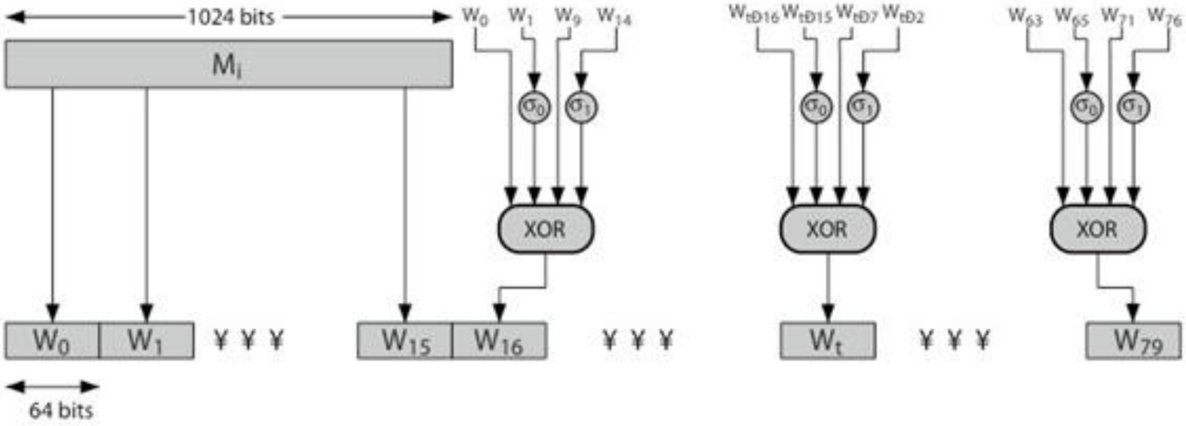
3.8.2 SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - ✓ updating a 512-bit buffer
  - ✓ using a 64-bit value Wt derived from the current message block
  - ✓ and a round constant based on cube root of first 80 prime numbers

3.8.3 SHA-512 Round Function



3.8.4 SHA-512 –Individual Round Function



3.9 DIGITAL SIGNATURES

- have looked at message authentication- but does not address issues of lack of trust
- digital signatures provide the ability to:
  - ✓ verify author, date & time of signature
  - ✓ authenticate message contents
  - ✓ be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

3.9.1 Digital Signature Properties

- must depend on the message signed
- must use information unique to sender-to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - ✓ with new message for existing digital signature
  - ✓ with fraudulent digital signature for given message
- be practical save digital signature in storage

3.10 DIRECT DIGITAL SIGNATURES

- involve only sender & receiver
- assumed receiver has sender’s public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key

3.10.1 Arbitrated Digital Signatures

- involves use of arbiter A
  - ✓ validates any signed message then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

Authentication Protocols

- used to convince parties of each others identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - ✓ confidentiality – to protect session keys
  - ✓ timeliness – to prevent replay attacks
- published protocols are often found to have flaws and need to be modified

3.10.2 Replay Attacks

- where a valid signed message is copied and later resent
  - ✓ simple replay
  - ✓ repetition that can be logged
  - ✓ repetition that cannot be detected
  - ✓ backward replay without modification
- countermeasures include
  - ✓ use of sequence numbers (generally impractical)
  - ✓ timestamps (needs synchronized clocks)
  - ✓ challenge/response (using unique nonce)

Using Symmetric Encryption

- as discussed previously can use a twolevel hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - ✓ each party shares own master key with KDC
  - ✓ KDC generates session keys used for connections between parties
  - ✓ master keys used to distribute these to them
- can refine use of KDC but can't have final exchange of nonces, vis:
  - ✓ A->KDC: IDA || IDB || N1
  - ✓ KDC -> A: EKa [Ks || IDB || N1 || EKb [Ks||IDA]]
  - ✓ A -> B: EKb [Ks||IDA] || EKs [M]
- does not protect against replays
  - ✓ could rely on timestamp in message, though email delays make this problematic

Using Public-Key Encryption

- have a range of approaches based on the use of public-key encryption
- need to ensure have correct public keys for other parties
- using a central Authentication Server (AS)
- various protocols exist using timestamps or nonces
- if confidentiality is major concern, can use:

  A->B: EPUb [Ks] || EKs [M]

  - ✓ has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:

  A->B: M || EPRa [H(M)] || EPRas [T||IDA||PUa]

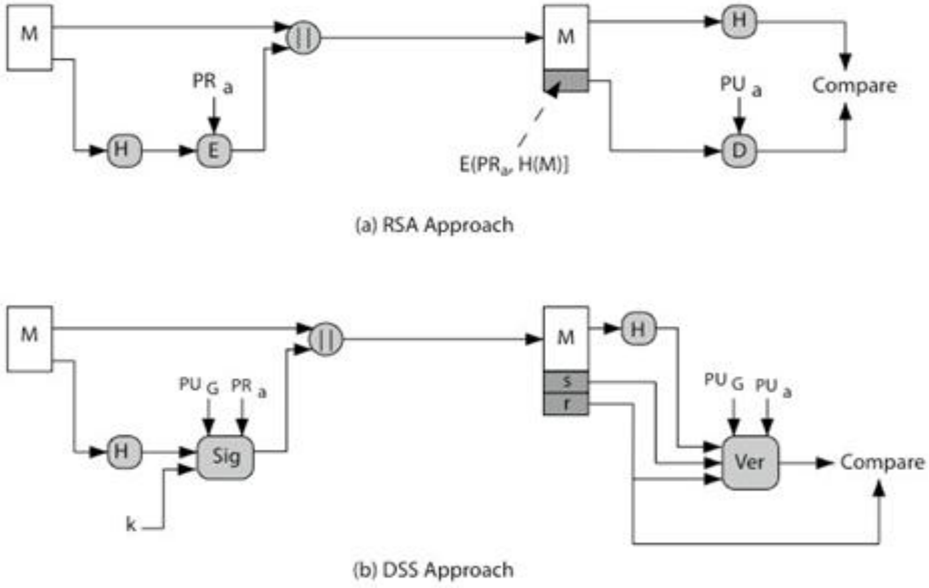
  - ✓ with message, signature, certificate

3.11 DIGITAL SIGNATURE STANDARD (DSS)

- US Govt approved signature scheme
- designed by NIST & NSA in early 90's
- published as FIPS-186 in 1991
- revised in 1993, 1996 & then 2000
- uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- FIPS 186-2 (2000) includes alternative RSA & elliptic curve signature variants

3.12 DIGITAL SIGNATURE ALGORITHM(DSA)

- creates a 320 bit signature
- with 512-1024 bit security
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms
- variant of ElGamal & Schnorr schemes



(a) RSA Approach

(b) DSS Approach

Fig. 3.12.1: Digital Signatures - Overviews

3.12.1 DSA Key Generation

- have shared global public key values (p,q,g):
- choose q, a 160 bit
- choose a large prime p = 2L
  - ✓ where L= 512 to 1024 bits and is a multiple of 64 and q is a prime factor of (p-1)
- choose g = h(p-1)/q
  - ✓ where h<p-1, h(pp-11)//q (mod p) > 1
- users choose private & compute public key:
  - ✓ choose x<q
  - ✓ compute y = gx (mod p)

3.12.2 DSA Signature Creation

- to sign a message M the sender:
  - ✓ generates a random signature key k, k<q
  - ✓ nb. k must be random, be destroyed after use, and never be reused
- then computes signature pair:

  r = (gk(mod p))(mod q)

  s = (k-1.H(M)+ x.r)(mod q)
- sends signature (r,s) with message M

3.12.3 DSA Signature Verification

- having received M & signature (r,s)
- to verify a signature, recipient computes:

  w = s-1(mod q)

  u1= (H(M).w)(mod q)

  u2= (r.w)(mod q)

  v = (gu1.yu2(mod p)) (mod q)

- if v=r then signature is verified

UNIT 4

4.1 AUTHENTICATION APPLICATIONS

4.1.1 KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

The following are the requirements for Kerberos:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78] It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

4.1.1.1 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. To counter this threat, servers must be able to confirm the identities of clients who request service. 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.

The simple authentication dialogue is as follows:

□ C >> AS: IDc||Pc||IDv
□ AS >> C: Ticket
□ C >> V: IDc||Ticket Ticket=
EKv(IDc||ADc||IDv)

C: Client,
AS: Authentication Server, V: Server,
IDv : ID of the server, IDc : ID of the client,
Pc:Password of the client, ADc: Address of client,
Kv: secret key shared by AS and V, ||:
concatenation.

4.1.1.2 A More Secure Authentication Dialogue

There are two major problems associated with the previous approach:

- Plaintext transmission of the password.
- Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and anew server, known as ticket granting server (TGS). The hypothetical scenario is as follows: **Once per user logon session:**

- C >> AS: IDc||IDtgs
- AS >> C: Ek_c ($Ticket_{tgs}$)

Once per type of service:

- C >> TGS: IDc||IDv||$Ticket_{tgs}$
- TGS >> C: $ticket_v$

Once per service session:

5. C >> V: IDc||$ticket_v$

$Ticket_{tgs}$= Ek_{tgs}(IDc||ADc||IDtgs||TS1||Lifetime1) $Ticket_v$=

$Ek_V(IDc||ADc||IDv||TS2||Lifetime2)$

C: Client, AS: Authentication Server, V: Server, IDc : ID of the client, Pc:Password of the client, ADc: Address of client, IDv : ID of the server, Kv: secret key shared by AS and V, ||: concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket (Ticket$_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

□ The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

□ The AS responds with a ticket that is encrypted with a key that is derived from the user's password.
When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message.

If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

O 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.

P The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_V) known only to the TGS and the server, preventing alteration.

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

e) 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.

4.2 Kerbero V4 Authentication Dialogue Message Exchange

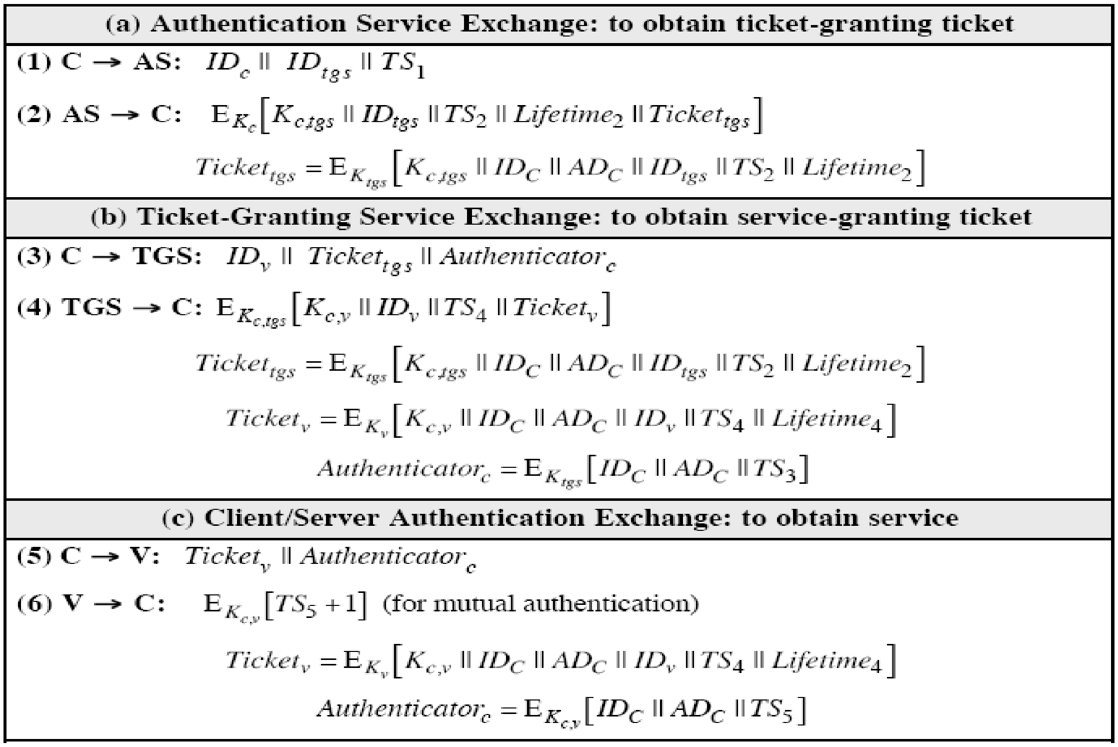
Two additional problems remain in the more secure authentication dialogue:

γ) Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

η) Requirement for the servers to authenticate themselves to users.

The actual Kerberos protocol version 4 is as follows :

5. a basic third-party authentication scheme
6. have an Authentication Server (AS)
   - users initially negotiate with AS to identify self
   - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
7. have a Ticket Granting server (TGS)
   - users subsequently request access to other services from TGS on basis of users TGT



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

(1) **C → AS:** $ID_c \parallel ID_{tgs} \parallel TS_1$

(2) **AS → C:** $E_{K_c}\left[K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$$

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

(3) **C → TGS:** $ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$

(4) **TGS → C:** $E_{K_{c,tgs}}\left[K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$$

$$Ticket_v = E_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$$

$$Authenticator_c = E_{K_{tgs}}\left[ID_C \parallel AD_C \parallel TS_3\right]$$

(c) Client/Server Authentication Exchange: to obtain service

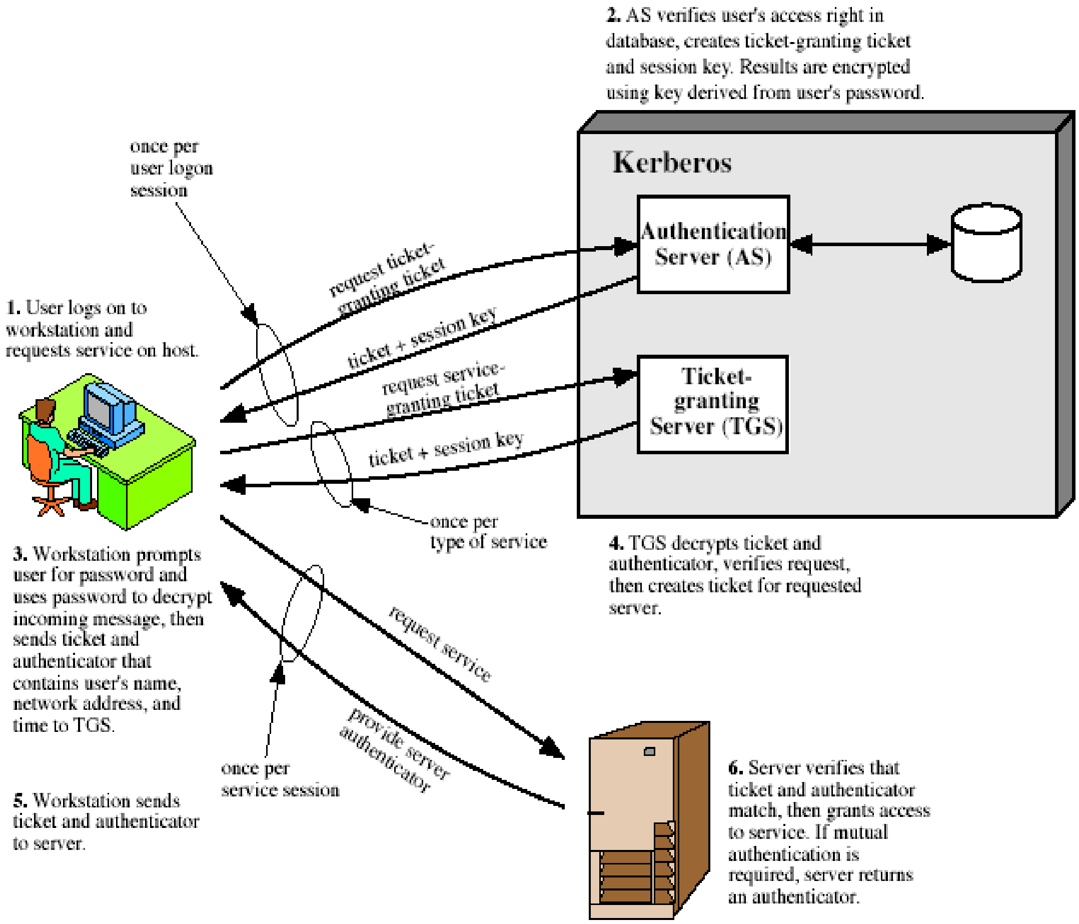
(5) **C → V:** $Ticket_v \parallel Authenticator_c$

(6) **V → C:** $E_{K_{c,v}}\left[TS_5 + 1\right]$ (for mutual authentication)

$$Ticket_v = E_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$$

$$Authenticator_c = E_{K_{c,v}}\left[ID_C \parallel AD_C \parallel TS_5\right]$$

Kerberos 4 Overview


Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
request service
provide server authenticator
once per service session

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.

1. User logs on to workstation and requests service on host.

3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.

4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.

5. Workstation sends ticket and authenticator to server.

6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

4.2.1 Kerberos Realms and Multiple Kerberi

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

- 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.
- 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.

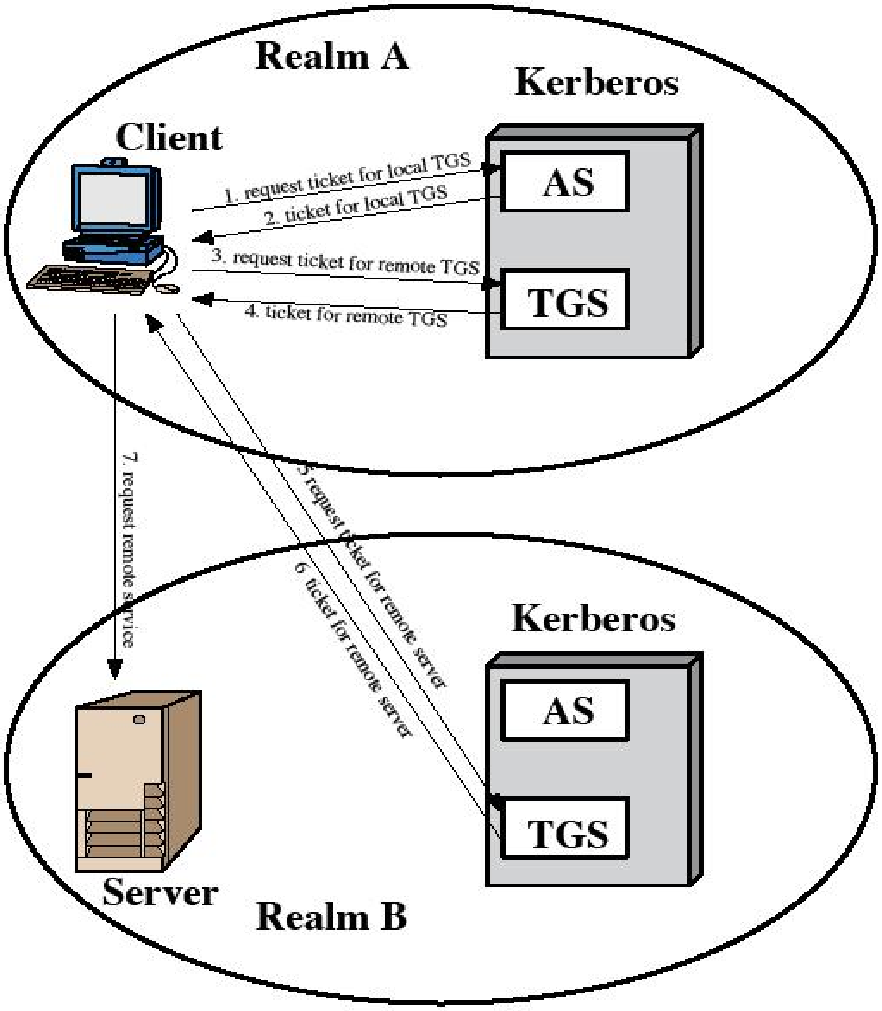

Realm A
Kerberos
Client
AS
TGS
1. request ticket for local TGS
2. ticket for local TGS
3. request ticket for remote TGS
4. ticket for remote TGS
7. request remote service
5 request ticket for remote server
6 ticket for remote server
Kerberos
AS
TGS
Server
Realm B

Fig. 4.2.1.1: Request for service in another Realm

A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room.

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

However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password.

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

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

4.2.2 Kerberos version 5

Version 5 of Kerberos provides a number of improvements over version 4.

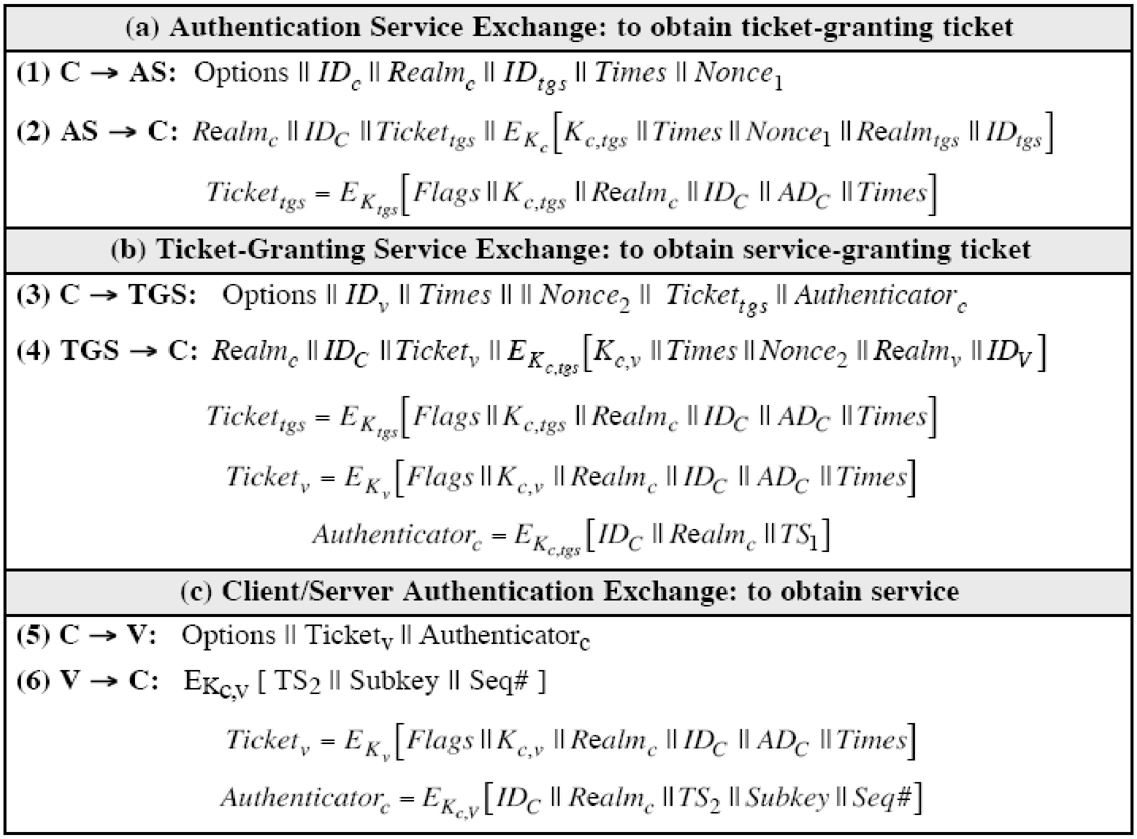
4. developed in mid 1990's
5. provides improvements over v4
   - addresses environmental shortcomings
   - and technical deficiencies
6. specified as Internet standard RFC 151

Differences between version 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas:

- **Environmental shortcomings**
  - encryption system dependence
  - internet protocol dependence
  - message byte ordering
  - ticket lifetime
  - authentication forwarding
  - inter-realm authenticaiton
- **Technical deficiencies**
  - double encryption
  - PCBC encryption
  - Session keys
  - Password attacks

The version 5 authentication dialogue



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

(1) C → AS: $\text{Options} \parallel ID_c \parallel Realm_c \parallel ID_{tgs} \parallel Times \parallel Nonce_1$

(2) AS → C: $Realm_c \parallel ID_C \parallel Ticket_{tgs} \parallel E_{K_c}\left[K_{c,tgs} \parallel Times \parallel Nonce_1 \parallel Realm_{tgs} \parallel ID_{tgs}\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \parallel K_{c,tgs} \parallel Realm_c \parallel ID_C \parallel AD_C \parallel Times\right]$$

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

(3) C → TGS: $\text{Options} \parallel ID_v \parallel Times \parallel \parallel Nonce_2 \parallel Ticket_{tgs} \parallel Authenticator_c$

(4) TGS → C: $Realm_c \parallel ID_C \parallel Ticket_v \parallel E_{K_{c,tgs}}\left[K_{c,v} \parallel Times \parallel Nonce_2 \parallel Realm_v \parallel ID_V\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \parallel K_{c,tgs} \parallel Realm_c \parallel ID_C \parallel AD_C \parallel Times\right]$$

$$Ticket_v = E_{K_v}\left[Flags \parallel K_{c,v} \parallel Realm_c \parallel ID_C \parallel AD_C \parallel Times\right]$$

$$Authenticator_c = E_{K_{c,tgs}}\left[ID_C \parallel Realm_c \parallel TS_1\right]$$

(c) Client/Server Authentication Exchange: to obtain service

(5) C → V: $\text{Options} \parallel \text{Ticket}_V \parallel \text{Authenticator}_C$

(6) V → C: $\text{E}_{\text{Kc,V}}\ [\ \text{TS}_2 \parallel \text{Subkey} \parallel \text{Seq\#}\ ]$

$$Ticket_v = E_{K_v}\left[Flags \parallel K_{c,v} \parallel Realm_c \parallel ID_C \parallel AD_C \parallel Times\right]$$

$$Authenticator_c = E_{K_{c,V}}\left[ID_C \parallel Realm_c \parallel TS_2 \parallel Subkey \parallel Seq\#\right]$$

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

- Realm: Indicates realm of user
- Options: Used to request that certain flags be set in the returned ticket
- Times: Used by the client to request the following time settings in the ticket: from:
  the desired start time for the requested ticket
  till: the requested expiration time for the requested ticket rtime:
  requested renew-till time
- **<u>Nonce</u>:** 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. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that 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).

The authenticator itself is essentially the same as the one used in version 4.

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

Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.
- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

Ticket Flags

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

4.3 X.509 Certificates

4.3.1 Overview:

- **issued by a Certification Authority (CA), containing:**
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- **notation CA<<A>> denotes certificate for A signed by CA**

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME), IP Security and SSL/TLS and SET

4.3.2 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.

- **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.

4. **Signature algorithm identifier**:

The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.

4. **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.

- **name**:

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

- **Subject's public-key information**:

The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.

- **Issuer unique identifier**:

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

- **Subject unique identifier**:

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

- **Extensions:**

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

- **Signature:**

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

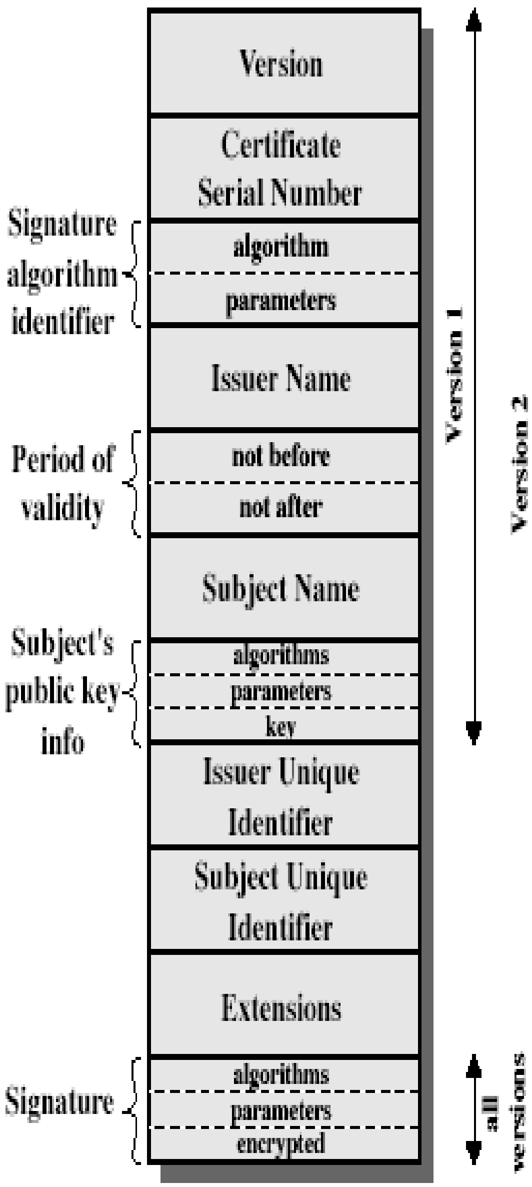
The standard uses the following notation to define a certificate:

$$\mathbf{CA<<A>> = CA\ \{V, SN, AI, CA, T_A, A, Ap\}}$$

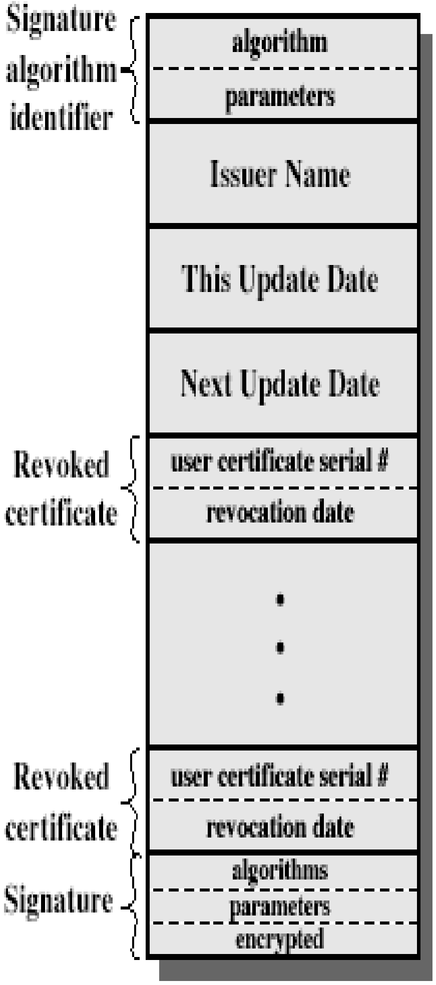
Where, Y <<X>> = the certificate of user X issued by certification authority Y

Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

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



(a) X.509 Certificate



(b) Certificate Revocation List

4.3.3 Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X $_2$, then B's certificate, issued by X_2, is useless to A.

A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2 Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1<<X_2>> X_2 <<B>>$$

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

$$X_2<<X_1>> X_1 <<A>>$$

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

$$X_1<<X_2>> X_2 <<X_3>>... X_N<<B>>$$

In this case, each pair of CAs in the chain ($X_{i,}$ X_{i+1}) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

From X.509, is an example of such a 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

4.3.4 CA Hierarchy Use

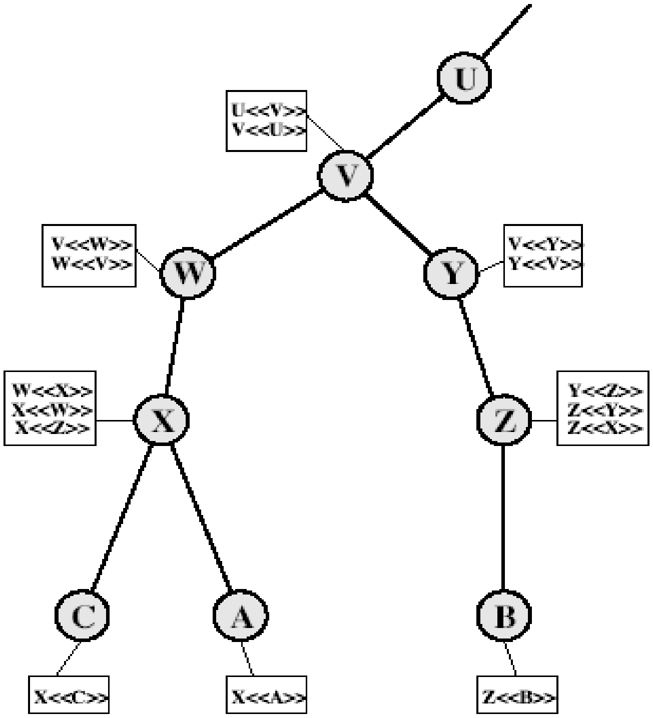
In the example given below , user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

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



Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, for the following reasons eg:
    1. user's private key is compromised
    2. user is no longer certified by this CA
    3. CA's certificate is compromised
- CA’s maintain list of revoked certificates
    1. the Certificate Revocation List (CRL)
- users should check certs with CA’s CRL

4.4 Authentication Procedures

X.509 includes three alternative authentication procedures:

- **One-Way Authentication**
- **Two-Way Authentication**
- **Three-Way Authentication**
- all use public-key signatures

One-Way Authentication

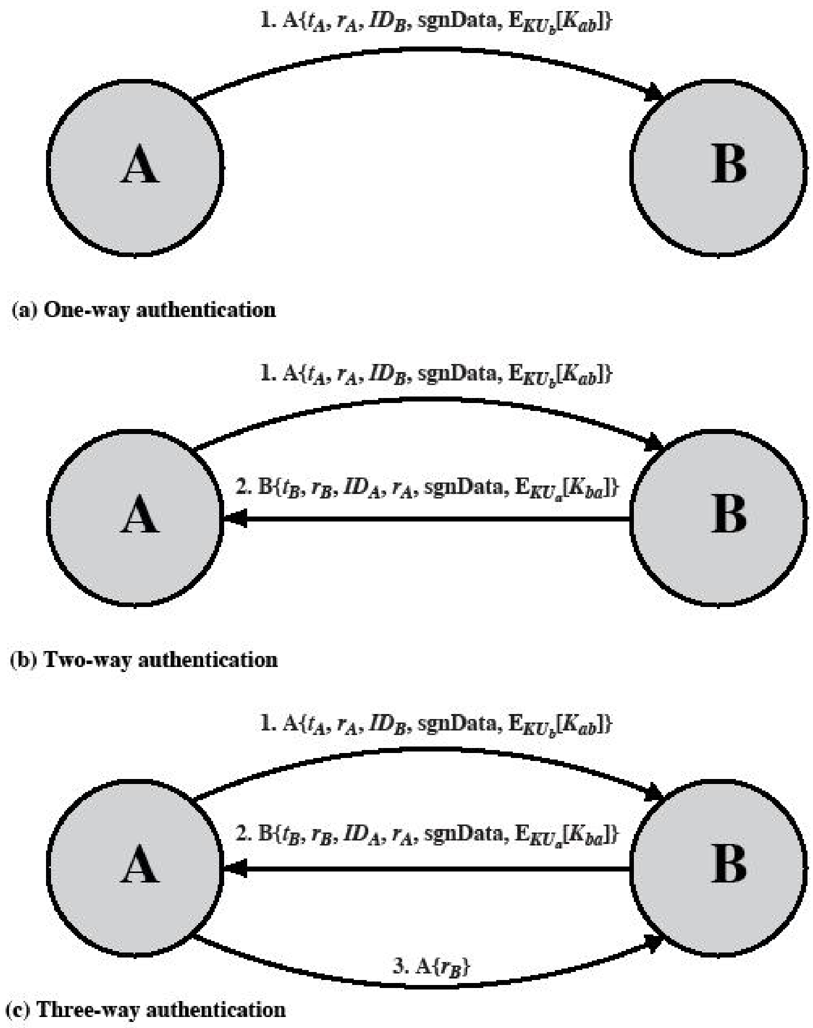
- 1 message (A->B) used to establish
    - the identity of A and that message is from A
    - message was intended for B
    - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upo


1. A{t_A, r_A, ID_B, sgnData, $E_{KU_b}[K_{ab}]$}
A
B
1. A{t_A, r_A, ID_B, sgnData, $E_{KU_b}[K_{ab}]$}
2. B{t_B, r_B, ID_A, r_A, sgnData, $E_{KU_a}[K_{ba}]$}
A
B
1. A{t_A, r_A, ID_B, sgnData, $E_{KU_b}[K_{ab}]$}
2. B{t_B, r_B, ID_A, r_A, sgnData, $E_{KU_a}[K_{ba}]$}
A
B
3. A{r_B}

(a) One-way authentication

(b) Two-way authentication

(c) Three-way authentication

Fig.4.4.1 X509 Strong Authentication Procedures

4.5 ELECTRONIC MAIL SECURITY

4.5.1 PRETTY GOOD PRIVACY (PGP)

PGP provides the confidentiality and authentication service that can be used for electronic mail and file storage applications. The steps involved in PGP are:

- Select the best available cryptographic algorithms as building blocks.
- Integrate 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.
- Make the package and its documentation, including the source code, freely available via the internet, bulletin boards and commercial networks.
- Enter into an agreement with a company to provide a fully compatible, low cost commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

- It is available free worldwide in versions that run on a variety of platform.
- It is based on algorithms that have survived extensive public review and are considered extremely secure.

  e.g., RSA, DSS and Diffie Hellman for public key encryption CAST-128, IDEA and 3DES for conventional encryption SHA-1 for hash coding.
- It has a wide range of applicability.
- It was not developed by, nor it is controlled by, any governmental or standards organization.

Operational description

The actual operation of PGP consists of five services: authentication, confidentiality, compression, e-mail compatibility and segmentation.

1. **Authentication**

   The sequence for authentication is as follows:

   - The sender creates the 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.

2. **Confidentiality**

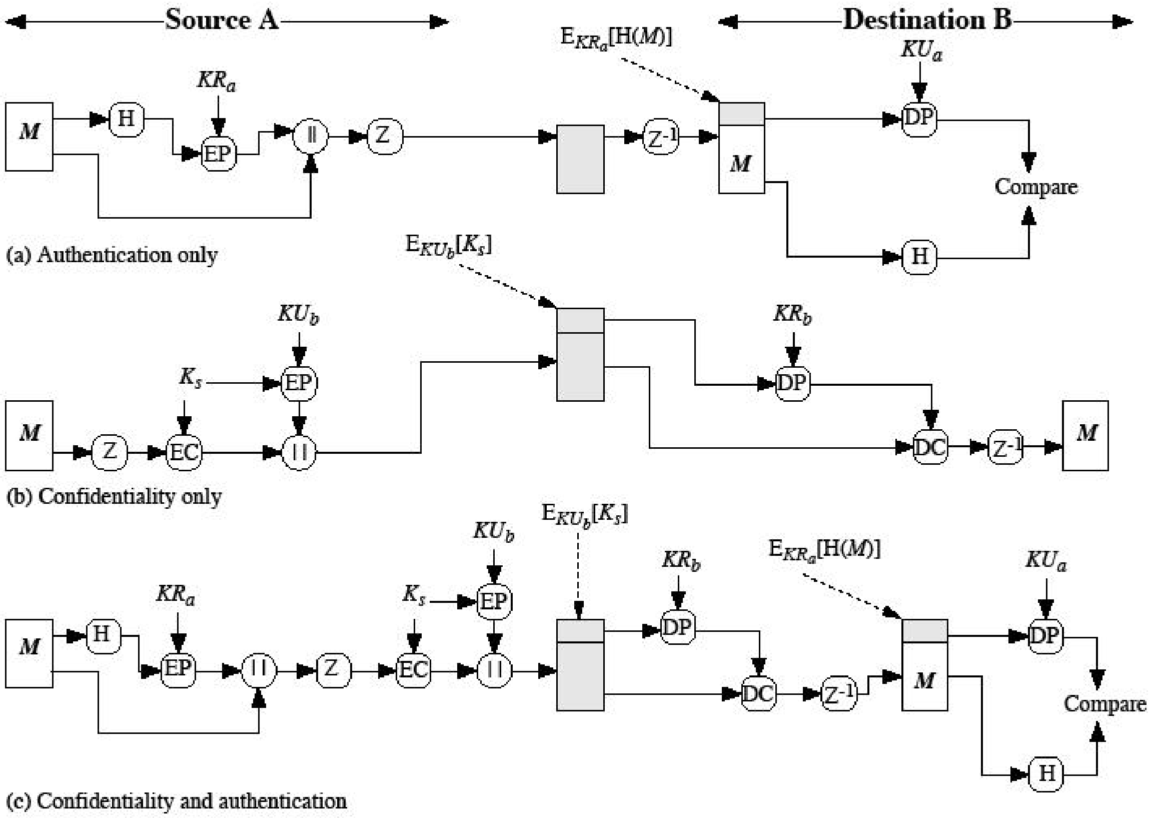
Confidentiality is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the conventional encryption algorithm CAST-128 may be used. The 64-bit cipher feedback (CFB) mode is used.

In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus although this is referred to as **a session key**, it is in reality a **one time key**. To protect the key, it is encrypted with the receiver's public key.

The sequence for confidentiality is as follows:

- 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 with the session key.
- The session key is encrypted with RSA, using the receiver'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.

Confidentiality and authentication



(a) Authentication only

(b) Confidentiality only

(c) Confidentiality and authentication

Fig.4.5.1.1:PGP Cryptographic Functions

Here both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext plus the signature is encrypted using CAST-128 and the session key is encrypted using RSA.

3. Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space for both e-mail transmission and for file storage.

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. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
- Even if one were willing to generate dynamically a recompressed message fro verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and as a result, produce different compression forms.

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult. The compression algorithm used is ZIP.

4. e-mail compatibility

Many electronic mail systems only permit the use of blocks consisting of ASCII texts. To accommodate this restriction, PGP provides the service of
converting the raw 8-bit binary stream to a stream of printable ASCII characters. The scheme used for this purpose is **radix-64 conversion**. Each group of three octets of binary data is mapped into four ASCII characters.
e.g., consider the 24-bit (3 octets) raw text sequence 00100011 01011100 10010001, we can express this input in block of 6-bits to produce 4 ASCII characters.

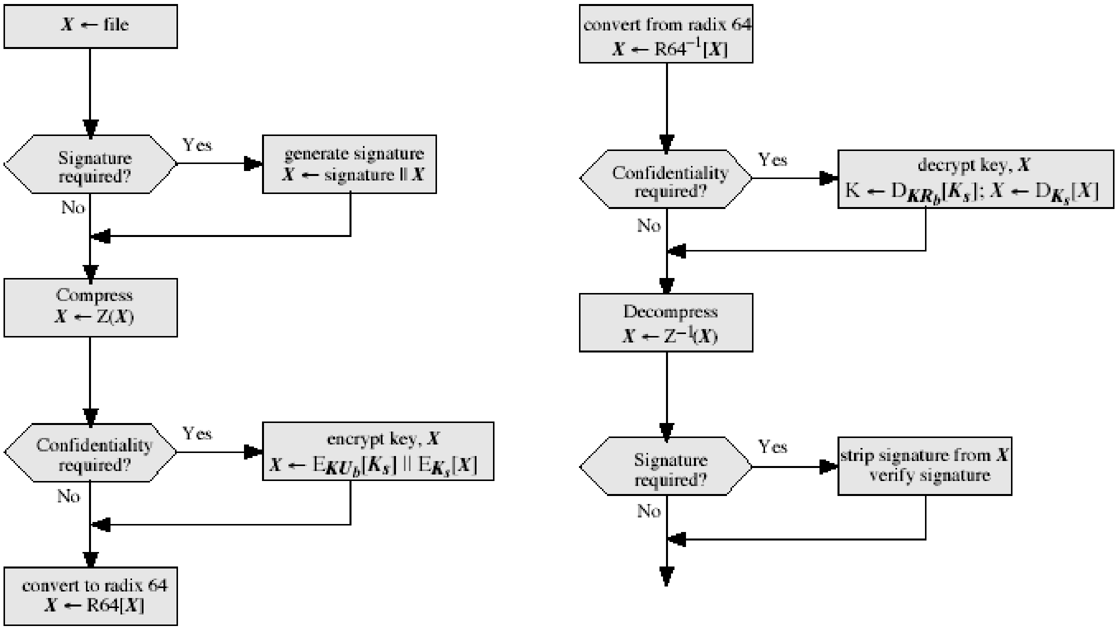
001000	110101	110010	010001
I	L	Y	R => corresponding ASCII characters

5. Segmentation and reassembly

E-mail facilities often are restricted to a maximum length. E.g., many of the facilities accessible through the internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all the other processing, including the radix-64 conversion. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing the other steps.

4.5.2 PGP Operation Summary:


$X \leftarrow$ file
Signature required?
Yes
generate signature $X \leftarrow$ signature $\|$ X
No
Compress $X \leftarrow Z(X)$
Confidentiality required?
Yes
encrypt key, X $X \leftarrow E_{KU_b}[K_s] \| E_{K_s}[X]$
No
convert to radix 64 $X \leftarrow R64[X]$
convert from radix 64 $X \leftarrow R64^{-1}[X]$
Confidentiality required?
Yes
decrypt key, X $K \leftarrow D_{KR_b}[K_s]$; $X \leftarrow D_{K_s}[X]$
No
Decompress $X \leftarrow Z^{-1}(X)$
Signature required?
Yes
strip signature from X verify signature
No

(a) Generic Transmission Diagram (from A) (b) Generic Reception Diagram (to B)

Cryptographic keys and key rings

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

- A means of generating unpredictable session keys is needed.
- It must allow a user to have multiple public key/private key pairs.
- Each PGP entity must maintain a file of its own public/private key pairs as well as a file

of public keys of correspondents.

We now examine each of the requirements in turn.

1. **Session key generation**

   Each session key is associated with a single message and is used only for the purpose of encryption and decryption of that message. Random 128-bit numbers are generated using CAST-128 itself. The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key. The plaintext input to CAST-128 is itself derived from a stream of 128-bit randomized numbers. These numbers are based on the keystroke input from the user.

2. **Key identifiers**

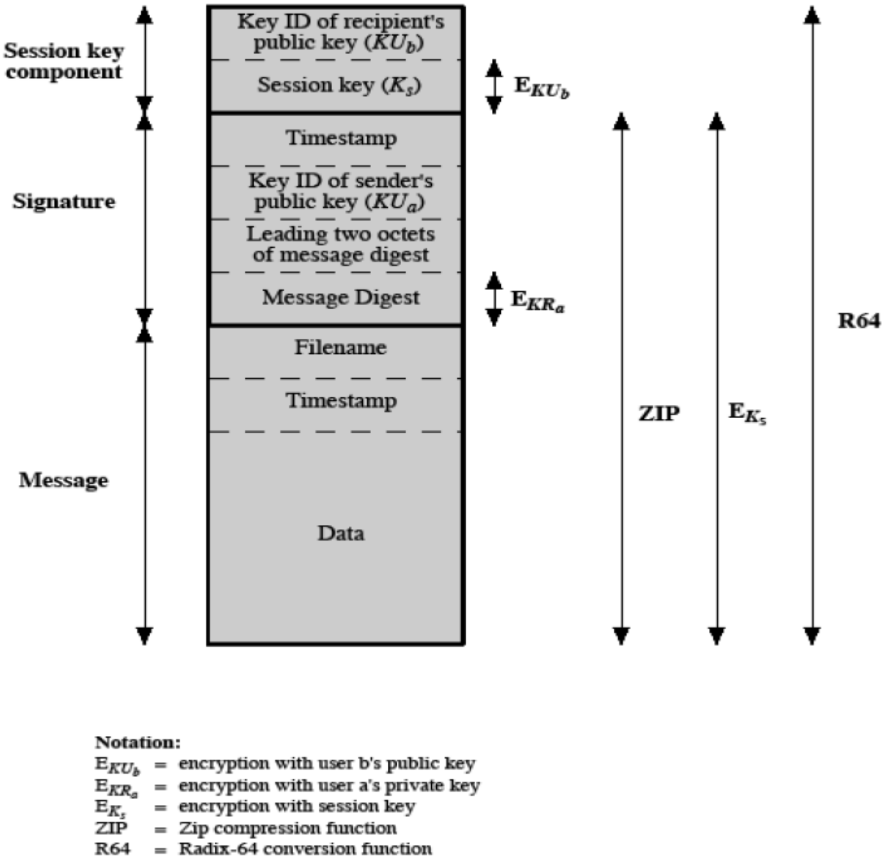
   If multiple public/private key pair are used, then how does the recipient know which of the public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message but, it is unnecessary wasteful of space. Another solution would be to associate an identifier with each public key that is unique at least within each user.

   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. i.e., the key ID of public key KU_a is

$$\mathbf{(KU_a \bmod 2^{64}).}$$

A message consists of three components.

- **Message component** – includes actual data to be transmitted, as well as the filename and a timestamp that specifies the time of creation.
- **Signature component** – includes the following
  - Timestamp – time at which the signature was made.
  - Message digest – hash code.
  - Two octets of message digest – to enable the recipient to determine if the correct public key was used to decrypt the message.
  - Key ID of sender’s public key – identifies the public key
- **Session key component** – includes session key and the identifier of the recipient public key.


Session key component
Key ID of recipient's public key (KU_b)
Session key (K_s)
E_{KU_b}
Signature
Timestamp
Key ID of sender's public key (KU_a)
Leading two octets of message digest
Message Digest
E_{KR_a}
Message
Filename
Timestamp
Data
ZIP
E_{K_s}
R64
Notation:
E_{KU_b} = encryption with user b's public key
E_{KR_a} = encryption with user a's private key
E_{K_s} = encryption with session key
ZIP = Zip compression function
R64 = Radix-64 conversion function

3. **Key rings**

PGP provides a pair of data structures at each node, one to store the public/private key pair owned by that node and one to store the public keys of the other users known at that node. These data structures are referred to as private key ring and public key ring.

4.5.3 The general structures of the private and public key rings are shown below:

Timestamp – the date/time when this entry was made.

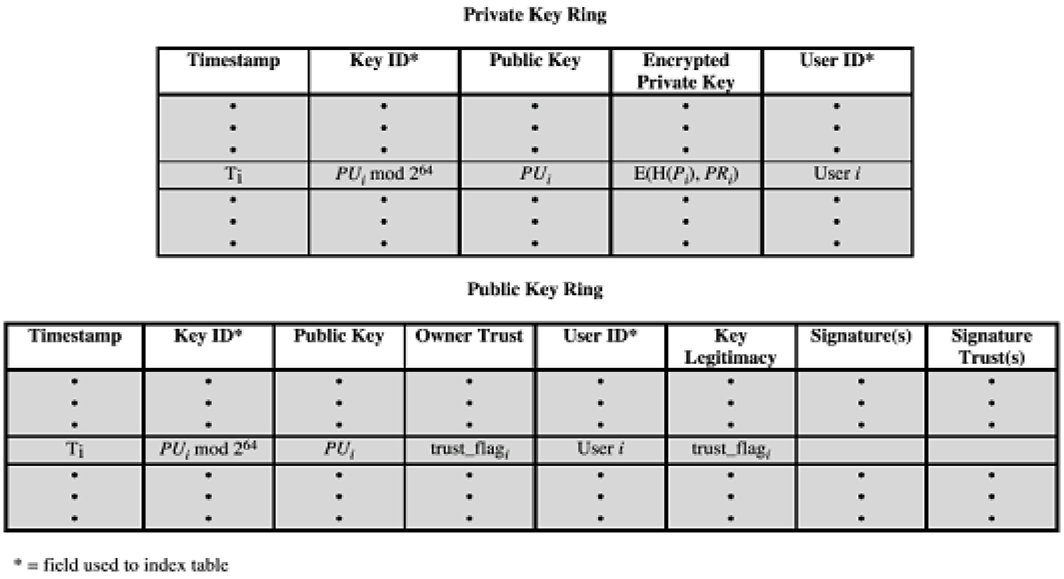
Key ID – the least significant bits of the public key.

Public key – public key portion of the pair.

Private key – private key portion of the pair.

User ID – the owner of the key.

Key legitimacy field – indicates the extent to which PGP will trust that this is a valid public key for this user.



Private Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
Ti	PU_i mod 2^{64}	PU_i	E(H(P_i), PR_i)	User *i*
• • •	• • •	• • •	• • •	• • •

Public Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
Ti	PU_i mod 2^{64}	PU_i	trust_flag$_i$	User *i*	trust_flag$_i$		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

* = field used to index table

Fig.4.5.3.1 General Structure of Private and Public Rings

Signature trust field – indicates the degree to which this PGP user trusts the signer to certify public key.

Owner trust field – indicates the degree to which this public key is trusted to sign other public key certificates.

PGP message generation

First consider message transmission and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

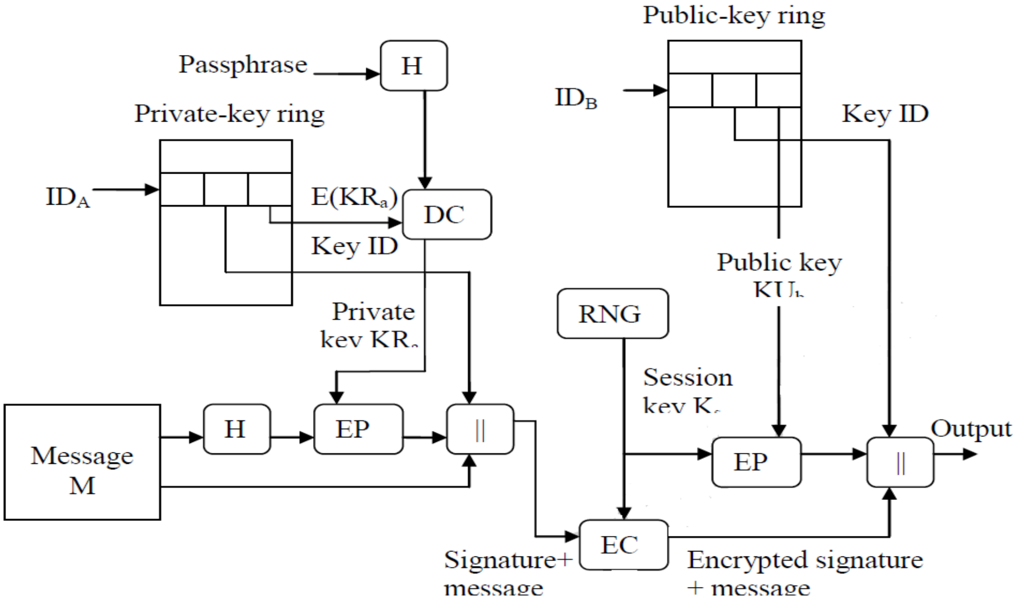


Figure 4.5.3.1: PGP message generation

1. **Signing the message**
   - PGP retrieves the sender's private key from the private key ring using user ID as an index. If user ID was not provided, the first private key from the ring is retrieved.
   - PGP prompts the user for the passpharse (password) to recover the unencrypted private key.
   - The signature component of the message is constructed.
2. **Encrypting the message**
   - PGP generates a session key and encrypts the message.
   - PGP retrieves the recipient's public key from the public key ring using user ID as index.
   - The session key component of the message is constructed.

The receiving PGP entity performs the following steps

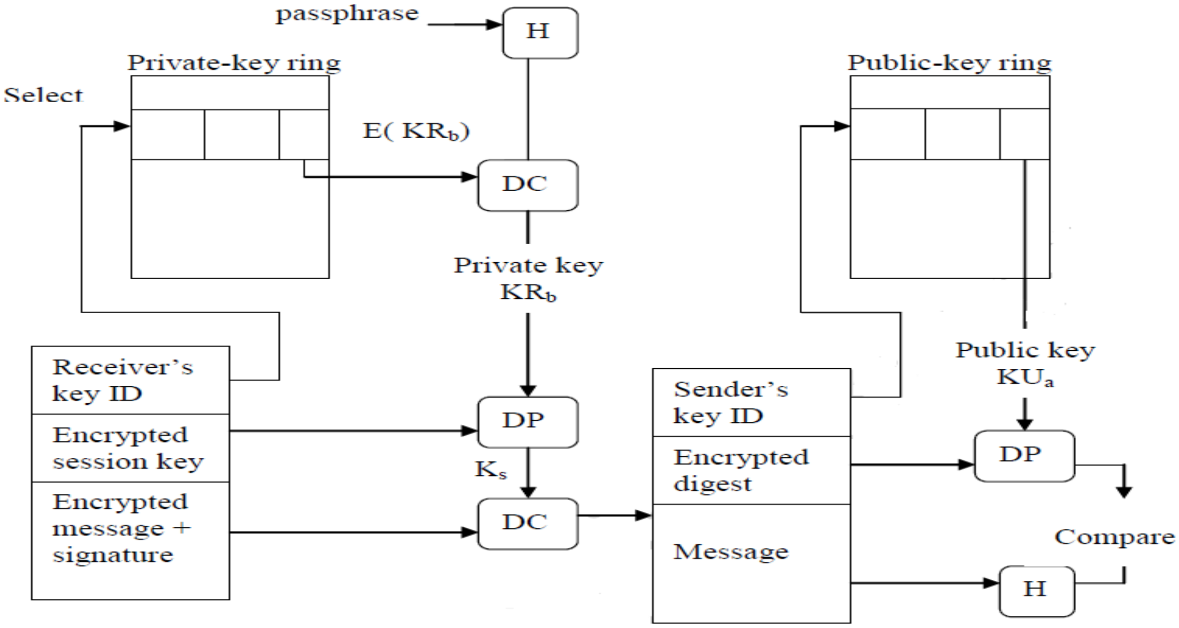


Figure: PGP message reception

1. **Decrypting the message**
   - PGP retrieves the receiver's private key from the private key ring, using the key ID field in the session key component of the message as an index.
   - PGP prompts the user for the passpharse (password) to recover the unencrypted private key.
   - 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.

4.6 Public-Key Management

This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. PGP provides a structure for solving this problem, with several suggested options that may be used.

4.6.1 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 that the key is, in fact, owned by C. This could happen if, for example, 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. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

1. Physically get the key from B. B could store her public key (PU_b) on a floppy disk and hand it to A..
2. 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.
3. 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.
4. 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.

For cases 3 and 4, A would already have to have a copy of the introducer's public key and trust that this key is valid. Ultimately, it is up to A to assign a level of trust to anyone who is to act as an introducer.

4.6.2 The Use of Trust

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. Each entry in the public-key ring is a public-key certificate.

Associated with each such entry is a 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. In turn, each signature has associated with it a signature trust field that 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.

Finally, each entry defines a public key associated with a particular owner, and an owner trust field is included that 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.

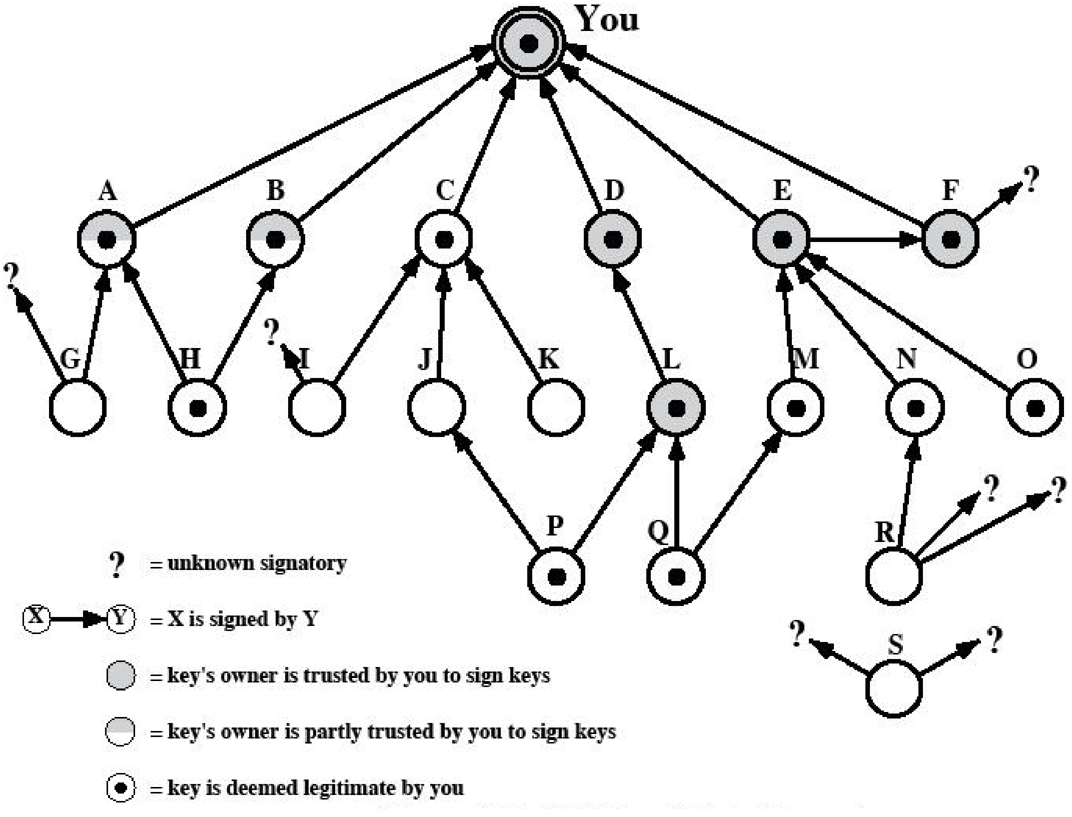
The three fields mentioned in the previous paragraph are each contained in a structure referred to as a trust flag byte.

Suppose that we are dealing with the public-key ring of user A. We can describe the operation of the trust processing as follows:

1. 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. 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.

2. 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.
3. 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.


You
A
B
C
D
E
F
?
?
G
H
?
I
J
K
L
M
N
O
P
Q
R
?
?
?
S
?
? = unknown signatory
X → Y = X is signed by Y
= key's owner is trusted by you to sign keys
= key's owner is partly trusted by you to sign keys
= key is deemed legitimate by you

The node labeled "You" refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which other users. If a key is signed by a user whose key is also in this key ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

4.7 S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

4.7.1 Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. Following are the limitations of SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
    - Deletion, addition, or reordering of carriage return and linefeed
    - Truncating or wrapping lines longer than 76 characters
    - Removal of trailing white space (tab and space characters)
    - Padding of lines in a message to the same length
    - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

4.7.2 Overview

The MIME specification includes the following elements:

1. **Five new message header** fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.

2. **A number of content formats** are defined, thus standardizing representations that support multimedia electronic mail.
3. **Transfer encodings** are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

4.7.3 The five header fields defined in MIME are as follows:

- **MIME-Version**: Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
- **Content-Type**: Describes the data contained in the body with sufficient detail
- **Content-Transfer-Encoding**: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
- **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.
- **Content-Description**: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

4.7.4 MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Below lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes

MIME Content Types (This item is displayed on page 461 in the print version)

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.
	octet-stream	General binary data consisting of 8-bit bytes.

For the text type of body, no special software is required to get the full meaning of the text, aside from support of the indicated character set. The primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility. The multipart type indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter, called boundary, that defines the delimiter between body parts.

The multipart/digest subtype is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message.

The message/partial subtype enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments.

The message/external-body subtype indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

4.7.5 MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mai transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted-printable and base64.

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.

The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

4.8 S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

4.8.1 Functions

S/MIME provides the following functions:

- **Enveloped data:** This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.
- **Signed data**: A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

4.8.2 Cryptographic Algorithms

- hash functions: SHA-1 & MD5
- digital signatures: DSS & RSA
- session key encryption: ElGamal & RSA
- message encryption: Triple-DES, RC2/40 and others
- have a procedure to decide which algorithms to use.

S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:

- Must: The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
- Should: There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

Cryptographic Algorithms Used in S/MIME

Function	**Requirement**
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.

Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

4.9 S/MIME Messages

S/MIME makes use of a number of new MIME content types. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

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	pkcs 7-mime	signedData	A signed S/MIME entity.
	pkcs 7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs 7-mime	degenerate signedData	An entity containing only public- key certificates.
	pkcs 7-mime	CompressedData	A compressed S/MIME entity
	pkcs 7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

UNIT V

5.1 INTRUDERS

One of the most publicized attacks to security is the intruder, generally referred to as hacker or cracker. Three classes of intruders are as follows:

- **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.
- **Misfeasor** – a legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuse his or her privileges.
- **Clandestine user** – an individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system. Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However there is no way in advance to know whether an intruder will be benign or malign.

An analysis of previous attack revealed that there were two levels of hackers:

- The high levels were sophisticated users with a thorough knowledge of the technology.
- The low levels were the ‘foot soldiers’ who merely use the supplied cracking programs with little understanding of how they work.

one of the results of the growing awareness of the intruder problem has been the establishment of a number of Computer Emergency Response Teams (CERT). these co-operative ventures collect information about system vulnerabilities and disseminate it to systems managers. Unfortunately, hackers can also gain access to CERT reports.

In addition to running password cracking programs, the intruders attempted to modify login software to enable them to capture passwords of users logging onto the systems.

Intrusion techniques

The objective of the intruders is to gain access to a system or to increase the range of privileges accessible on a system. Generally, this requires the intruders to acquire information that should be protected. In most cases, the information is in the form of a user password.

Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it. The password files can be protected in one of the two ways:

- **One way encryption** – the system stores only an encrypted form of user's password. 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.

The following techniques are used for learning passwords.

- Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
- Exhaustively try all short passwords.
- Try words in the system's online dictionary or a list of likely passwords.
- Collect information about users such as their full names, the name of their spouse and children, pictures in their office and books in their office that are related to hobbies.
- Try user's phone number, social security numbers and room numbers.
- Try all legitimate license plate numbers.
- Use a torjan horse to bypass restriction on access.
- Tap the line between a remote user and the host system.

Two principle countermeasures:

Θ Detection – concerned with learning of an attack, either before or after its success.

P Prevention – challenging security goal and an uphill bottle at all times.

5.2 INTRUSION DETECTION:

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

f) If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised.

g) An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.

h) Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified.

Figure 5.2.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

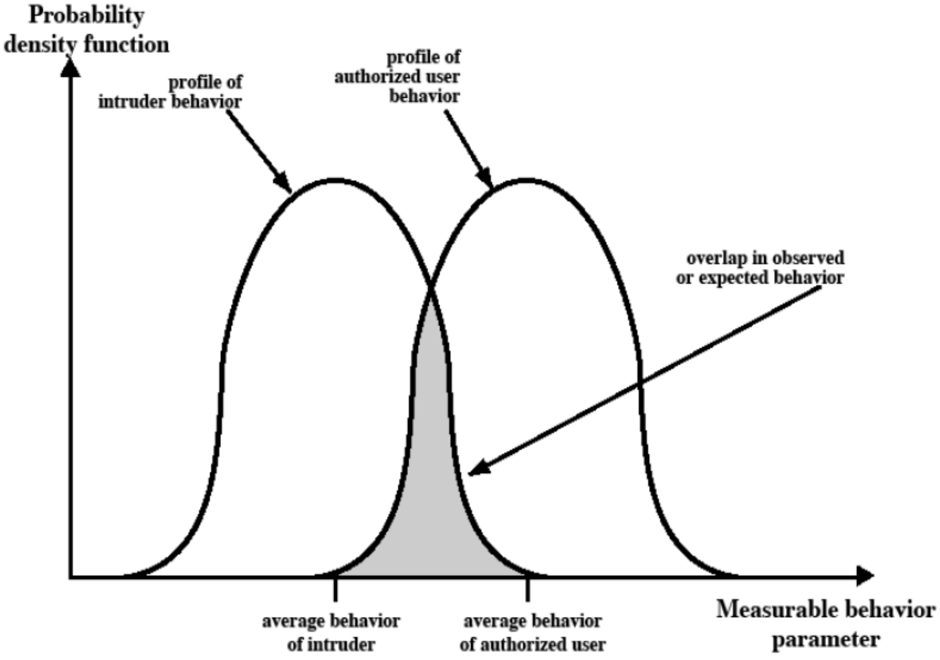

Probability density function
profile of intruder behavior
profile of authorized user behavior
overlap in observed or expected behavior
average behavior of intruder
average behavior of authorized user
Measurable behavior parameter

Fig. 5.2.1Profilesofbehavior of intruders and authorized users

5.2.1 The approaches to intrusion detection:

i)**Statistical anomaly detection**: Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.

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.

j)**Rule-based detection**: Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.

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. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

- **Native audit records**: Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.
- **Detection-specific audit records**: A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

Each audit record contains the following fields:

- **Subject:** Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users.
- **Object:** Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures

7. **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).
8. **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
```

issued by Smith to copy an executable file GAME from the current directory to the <Library> directory. The following audit records may be generated:

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

In this case, the copy is aborted because Smith does not have write permission to <Library>.

The decomposition of a user operation into elementary actions has three advantages:

2. 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
3. Single-object, single-action audit records simplify the model and the implementation.
7. 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.

5.2.1.1 Statistical Anomaly Detection:

As was mentioned, 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.

5.2.1.2 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. 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.

- 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.
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.

5.2.1.3 Rule-Based Intrusion Detection

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious.

Rule-based anomaly detection is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past

Rule-based penetration identification takes a very different approach to intrusion detection, one based on expert system technology. The key feature of such systems is the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses.

Example heuristics are the following:

1. Users should not read files in other users' personal directories.
2. Users must not write other users' files.
3. Users who log in after hours often access the same files they used earlier.
4. Users do not generally open disk devices directly but rely on higher-level operating system utilities.
5. Users should not be logged in more than once to the same system.
6. Users do not make copies of system programs.

5.2.2 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.

5.2.3 Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN 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.

- Either a centralized or decentralized architecture can be used.

Below figure 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.

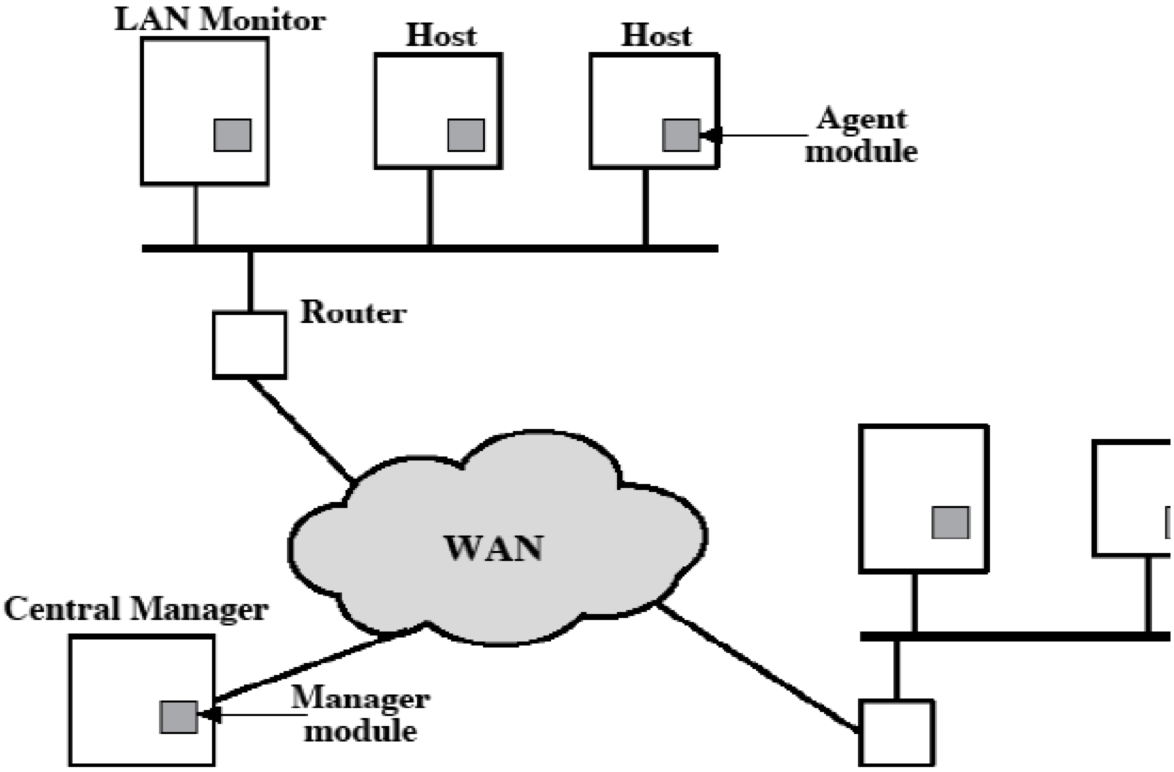


Fig. 5.2.3.1 Architecture of Distributed intrusion detection

The scheme is designed to be independent of any operating system or system auditing implementation.

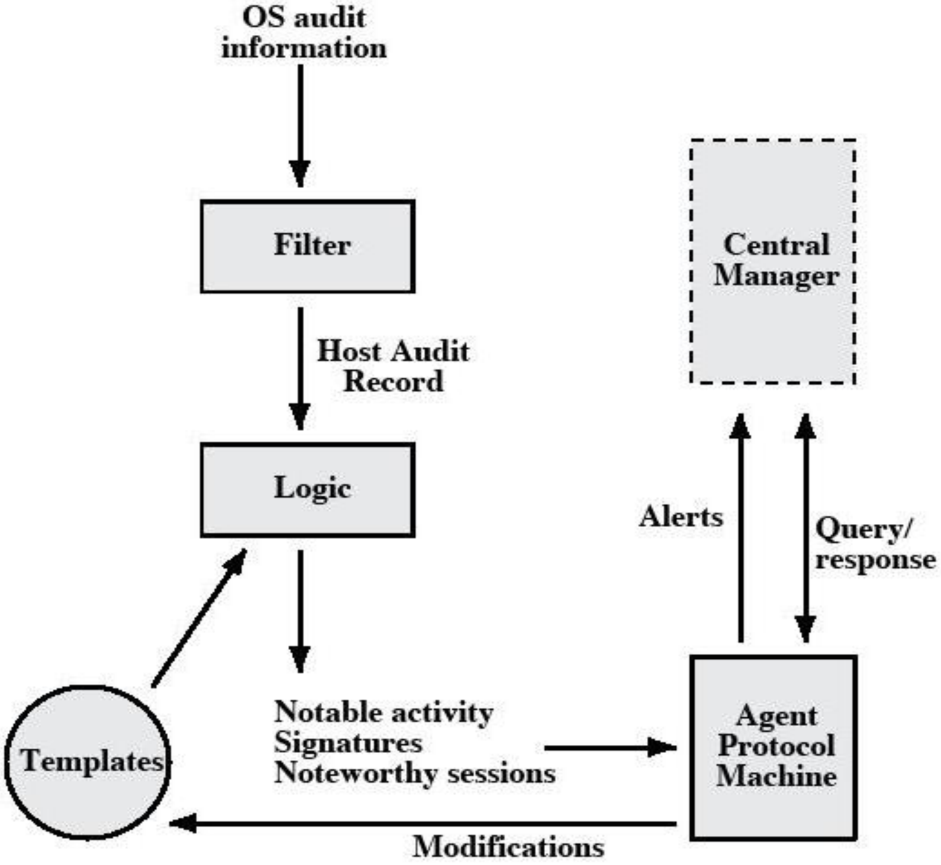
- The agent captures each audit record produced by the native audit collection system.
- A filter is applied that retains only those records that are of security interest.
- These records are then reformatted into a standardized format referred to as the host audit record (HAR).
- Next, a template-driven 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.
- When suspicious activity is detected, an alert is sent to the central manager.
- The central manager includes an expert system that can draw inferences from received data.
- The manager may also query individual systems for copies of HARs to correlate with those from other agents.
- The LAN monitor agent also supplies information to the central manager.
- The LAN monitor agent audits host-host connections, services used, and volume of traffic.
- It searches for significant events, such as sudden changes in network load, the use of security-related services, and network activities such as rlogin.

The architecture is quite general and flexible. It offers a foundation for a machine-independent approach that can expand from stand-alone intrusion detection to a system that is able to correlate activity from a number of sites and networks to detect suspicious activity that would otherwise remain undetected.

5.2.4 Honeypots

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.

5.2.5 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 outputs of this working group include the following:

1. A requirements document, which describes the high-level functional requirements for communication between intrusion detection systems and with management systems, including the rationale for those requirements.
2. A common intrusion language specification, which describes data formats that satisfy the requirements.
3. 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.

5.3 PASSWORD MANAGEMENT

5.3.1 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.
- The ID determines the privileges accorded to the user.
- 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.

5.3.2 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, the following procedure is employed.

- Each user selects a password of up to eight printable characters in length.
- This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine.
- The encryption routine, known as crypt(3), is based on DES. The DES algorithm is modified using a 12-bit "salt" value.

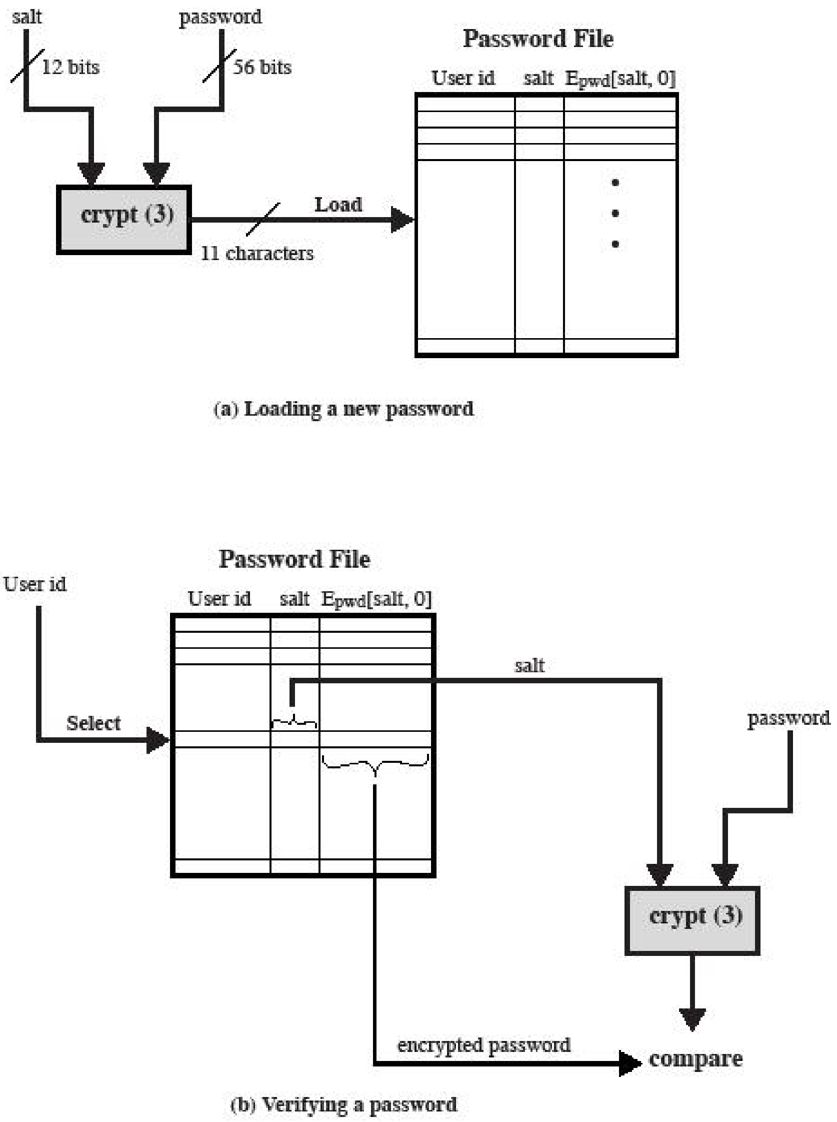
- Typically, this value is related to the time at which the password is assigned to the user.
- The modified DES algorithm is exercised with a data input consisting of a 64-bit block of zeros.
- The output of the algorithm then serves as input for a second encryption.
- This process is repeated for a total of 25 encryptions.
- The resulting 64-bit output is then translated into an 11-character sequence.
- The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID.
- This method has been shown to be secure against a variety of cryptanalytic attacks

The salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned at different times. Hence, the "extended" passwords of the two users will differ.
- It effectively increases the length of the password without requiring the user to remember two additional characters.
- 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 password are used as input to the encryption routine. If the result matches the stored value, the password is accepted.The encryption routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25.

Thus, there are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine.



(a) Loading a new password

(b) Verifying a password

Fig. 5.3.2.1 UNIX Password Scheme

As an example**, a password cracker was reported on the Internet in** August 1993. Using a Thinking Machines Corporation parallel computer, a performance of 1560 encryptions per second per vector unit was achieved. With four vector units per processing node (a standard configuration), this works out to 800,000 encryptions per second on a 128-node machine (which is a modest size) and 6.4 million encryptions per second on a 1024-node machine.

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward.

Following strategy was used:

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.
2. Try words from various dictionaries.
3. Try various permutations on the words from step 2.
4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

5.3.3 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.

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

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

A reactive password checking strategy is one in which the system periodically runs its own password cracker to find guessable passwords.

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 first approach is a simple system for rule enforcement. For example, the following rules could be enforced:

- All passwords must be at least eight characters long.
- In the first eight characters, the passwords must include at least one each of uppercase, lowercase, numeric digits, and punctuation marks. These rules could be coupled with advice to the user. Although this approach is superior to simply educating users, it may not be sufficient to thwart password crackers. This scheme alerts crackers as to which passwords not to try but may still make it possible to do password cracking.

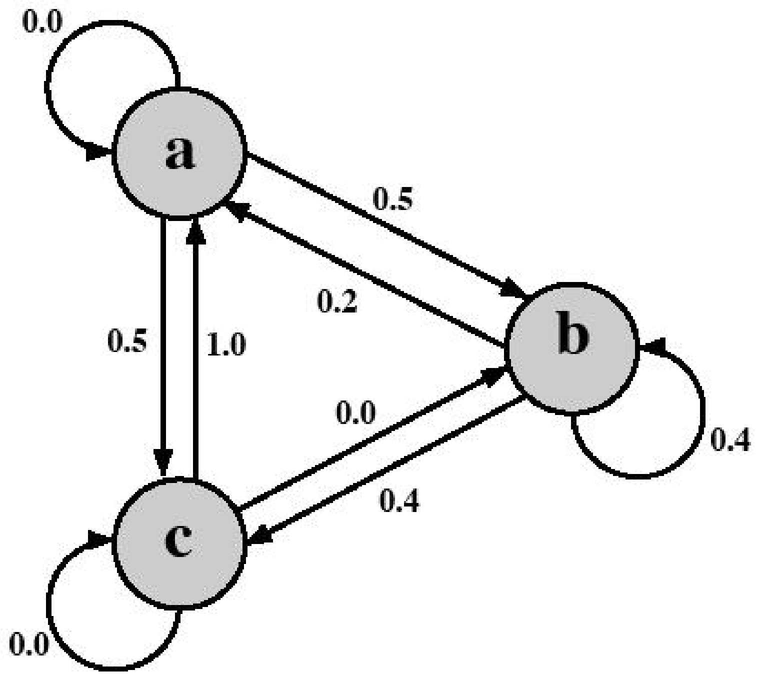
Another possible procedure is simply to compile a large dictionary of possible "bad" passwords. When a user selects a password, the system checks to make sure that it is not on the disapproved list.

There are two problems with this approach:

- Space: The dictionary must be very large to be effective..
- Time: The time required to search a large dictionary may itself be large

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. 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 kth-order model, the probability of making a transition to a particular letter depends on the previous k letters that have been generated.



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: aacccbaaa

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:

1. Determine the frequency matrix f, where f(i, j, k) is the number of occurrences of the trigram consisting of the ith, jth, and kth character. For example, the password parsnips yields the trigrams par, ars, rsn, sni, nip, and ips.
2. For each bigram ij, calculate f(i, j,∞) as the total number of trigrams beginning with ij. For example, f(a, b,∞) would be the total number of trigrams of the form aba, abb, abc, and so on.
3. Compute the entries of T as follows:

$$\mathbf{T(i,j,k) = f(i, j, k) / f(i, j,\infty)}$$

The result is a model that reflects the structure of the words in the dictionary.

A quite different approach has been reported by Spafford. It is based on the use of a Bloom filter. To begin, we explain the operation of the Bloom filter. A Bloom filter of order k consists of a set of k independent hash functions $H_1(x)$, $H_2(x)$,..., $H_k(x)$, where each function maps a password into a hash value in the range 0 to N - 1 That is,

$$\mathbf{H_i(X_j) = y \quad 1 \leq i \leq k; \quad 1 \leq j \leq D; \quad 0 \leq y \leq N- 1}$$

where
X_j = jth word in password dictionary
D = number of words in password dictionary

The following procedure is then applied to the dictionary:

1 A hash table of N bits is defined, with all bits initially set to 0.
2 For each password, its k hash values are calculated, and the corresponding bits in the hash table are set to 1. Thus, if $H_i(X_j)$ = 67 for some (i, j), then the sixty-seventh bit of the hash table is set to 1; if the bit already has the value 1, it remains at 1.

When a new password is presented to the checker, its k hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected.

5.4 FIREWALLS

5.4.1 Firewall design principles

Internet connectivity is no longer an option for most organizations. However, while internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates the threat to the organization. While it is possible to equip each workstation and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. The alternative, increasingly accepted, is the firewall.

The firewall is inserted between the premise network and internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from internet based attacks and to provide a single choke point where security and audit can be imposed. The firewall can be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

5.4.2 Firewall characteristics:

- All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible.
- 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.
- The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system. This implies that use of a trusted system with a secure operating system.

Four techniques that firewall use to control access and enforce the site's security policy is as follows:

- Service control – determines the type of internet services that can be accessed, inbound or outbound. The firewall may filter traffic on this basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as web or mail service.
- Direction control – determines the direction in which particular service request 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.
- Behavior control – controls how particular services are used.

Capabilities of 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.

= 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.

= A firewall can serve as the platform for IPsec.

5.4.3 Limitations of firewall

- 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 does not protect against internal threats. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.
- The firewall cannot protect against the transfer of virus-infected programs or files. Because of the variety of operating systems and applications supported inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and messages for viruses.

5.4.4 Types of firewalls

There are 3 common types of firewalls.

□ Packet filters

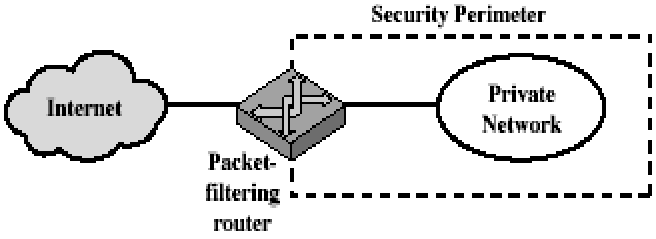
□ Application-level gateways

□ Circuit-level gateways

Packet filtering router

A packet filtering router applies a set of rules to each incoming IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions. Filtering rules are based on the information contained in a network packet:

- Source IP address – IP address of the system that originated the IP packet.
- Destination IP address – IP address of the system, the IP is trying to reach.
- Source and destination transport level address – transport level port number.
- IP protocol field – defines the transport protocol.
- Interface – for a router with three or more ports, which interface of the router the packet come from or which interface of the router the packet is destined for.



(a) Packet-filtering router

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken.

Two default policies are possible:

- Default = discard: That which is not expressly permitted is prohibited.
- Default = forward: That which is not expressly prohibited is permitted.

The default discard policy is the more conservative. Initially everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are most likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security.

5.4.5 Advantages of packet filter router

- Simple
- Transparent to users
- Very fast

Weakness 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.
- Because of the limited information available to the firewall, the logging functionality present in packet filter firewall is limited.
- It does not support advanced user authentication schemes.
- They are generally vulnerable to attacks such as layer address spoofing.

Some of the attacks that can be made on packet filtering routers and the appropriate counter measures are the following:

- IP address spoofing – the intruders transmit packets from the outside with a source IP address field containing an address of an internal host.

  Countermeasure: to discard packet with an inside source address if the packet arrives on an external interface.

- Source routing attacks – the source station specifies the route that a packet should take as it crosses the internet; i.e., it will bypass the firewall.

  Countermeasure: to discard all packets that uses this option.

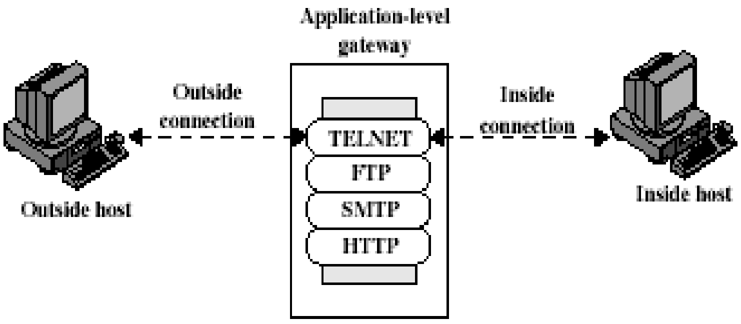
- Tiny fragment attacks – the intruder create extremely small fragments and force the TCP header information into a separate packet fragment. The attacker hopes that only the first fragment is examined and the remaining fragments are passed through.

  Countermeasure: to discard all packets where the protocol type is TCP and the IP fragment offset is equal to 1.

5.4.6 Application level gateway

An Application level gateway, also called a proxy server, acts as a relay of application level traffic. The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of 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.

Application level gateways tend to be more secure than packet filters. It is easy to log and audit all incoming traffic at the application level. A prime disadvantage is the additional processing overhead on each connection.

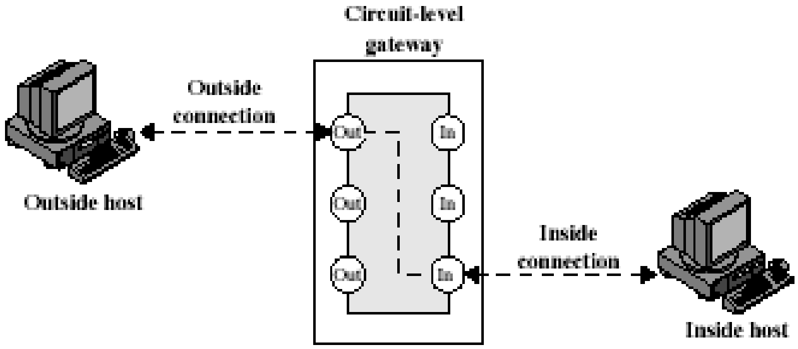


(b) Application-level gateway

5.4.7 Circuit level gateway

Circuit level gateway can be a stand-alone system or it can be a specified function performed by an application level gateway for certain applications. A Circuit level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outer host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of Circuit level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application level or proxy service on inbound connections and circuit level functions for outbound connections.



(c) Circuit-level gateway

Basiton host

It 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 and circuit level gateway.

Common characteristics of a Basiton host are as follows:

- The Bastion host hardware platform executes a secure version of its operating system, making it a trusted system.
- Only the services that the network administrator considers essential are installed on the Bastion host.
- It may require additional authentication before a user is allowed access to the proxy services.
- Each proxy is configured to support only a subset of 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.
- Each proxy is independent of other proxies on the Bastion host.
- A proxy generally performs no disk access other than to read its initial configuration file.
- Each proxy runs on a non privileged user in a private and secured directory on the Bastion host.

5.5 Firewall configurations

There are 3 common firewall configurations.

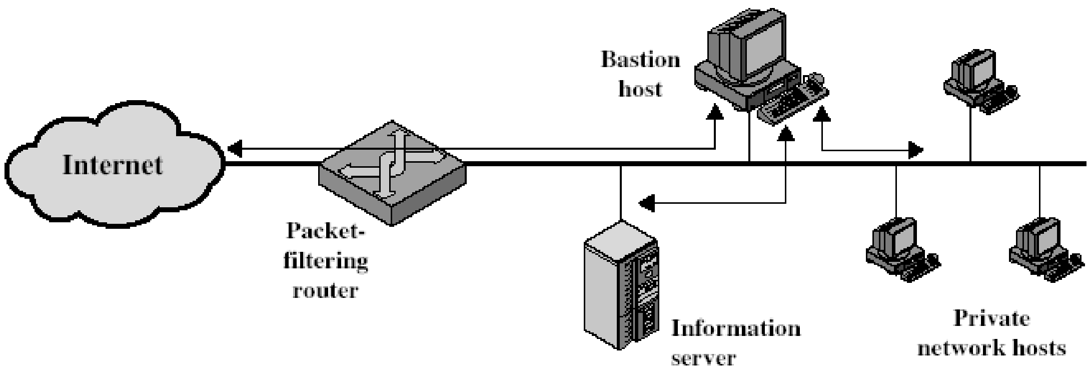
1. Screened host firewall, single-homed basiton configuration

In this configuration, the firewall consists of two systems: a packet filtering router and a bastion host. Typically, the router is configured so that

- For traffic from the internet, only IP packets destined for the basiton host are allowed in.
- For traffic from the internal network, only IP packets from the basiton host are allowed out.

The basiton host performs authentication and proxy functions. This configuration has greater security than simply a packet filtering router or an application level gateway alone, for two reasons:

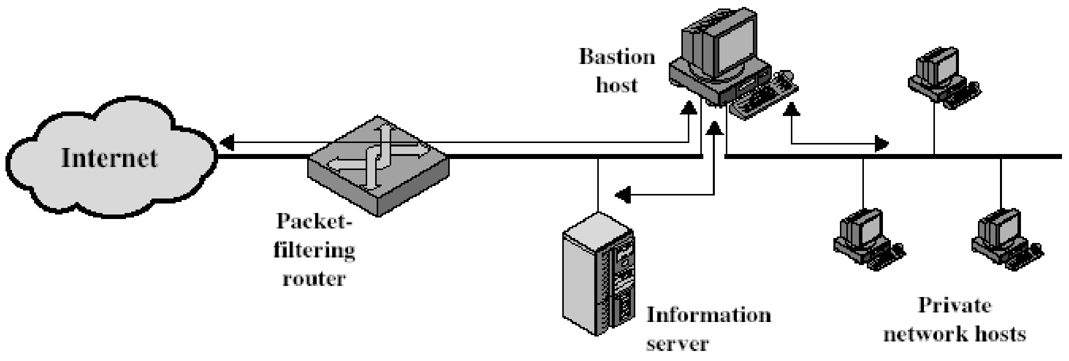
- This configuration implements both packet level and application level filtering, allowing for considerable flexibility in defining security policy.
- An intruder must generally penetrate two separate systems before the security of the internal network is compromised.



(a) Screened host firewall system (single-homed bastion host)

2. Screened host firewall, dual homed basiton configuration

In the previous configuration, if the packet filtering router is compromised, traffic could flow directly through the router between the internet and the other hosts on the private network. This configuration physically prevents such a security break.

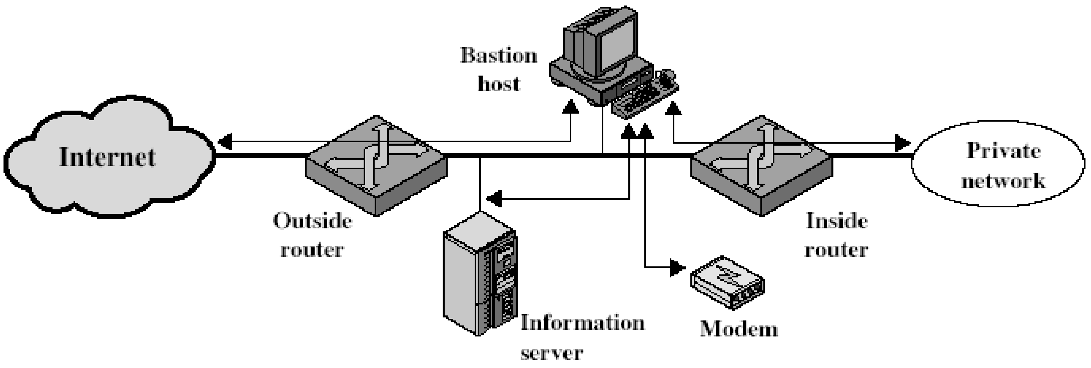


(b) Screened host firewall system (dual-homed bastion host)

3. Screened subnet firewall configuration

In this configuration, two packet filtering routers are used, one between the basiton host and internet and one between the basiton host and the internal network. This configuration creates an isolated subnetwork, which may consist of simply the basiton host but may also include one or more information servers and modems for dial-in capability. Typically both the internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. This configuration offers several advantages:

- There are now three levels of defense to thwart intruders.
- The outside router advertises only the existence of the screened subnet to the internet; therefore the internal network is invisible to the internet.
- Similarly, the inside router advertises only the existence of the screened subnet to the internal network; therefore the systems on the internal network cannot construct direct routes to the internet.



(c) Screened-subnet firewall system

5.6 Trusted systems

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology.

5.6.1 Data access control

Following successful logon, the user has been granted access to one or set of hosts and applications. This is generally not sufficient for a system that includes sensitive data in its database. Through the user access control procedure, 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 operating system can then enforce rules based on the user profile. The database management system, however, must control access to specific records or even portions of records. The operating system may grant a user permission to access a file or use an application, following which there are no further security checks, the database management system must make a decision on each individual access attempt. That decision will depend not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user.

A general model of access control as exercised by an file or database management system is that of an access matrix. The basic elements of the model are as follows:

- **Subject**: An entity capable of accessing objects. Generally, the concept of subject equates with that of process.
- **Object**: Anything to which access is controlled. Examples include files, portion of files, programs, and segments of memory.
- **Access right:** The way in which the object is accessed by a subject. Examples are read, write and 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. The other axis lists the objects that may be accessed. Objects may be individual data fields. Each entry in the matrix indicates the access rights of that subject for that object. The matrix may be decomposed by columns, yielding **access control lists.** 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.

	Program1	• • •	SegmentA	SegmentB
Process1	Read Execute		Read Write	
Process2				Read
• • •				

a. Access matrix

Access control list for Program1: Process1 (Read, Execute)
Access control list for Segment A: Process1 (Read, Write)
Access control list for Segment B: Process2 (Read)

b. Access control list

Capability list for Process1: Program1 (Read, Execute) Segment A (Read)
Capability list for Process2: Segment B (Read)

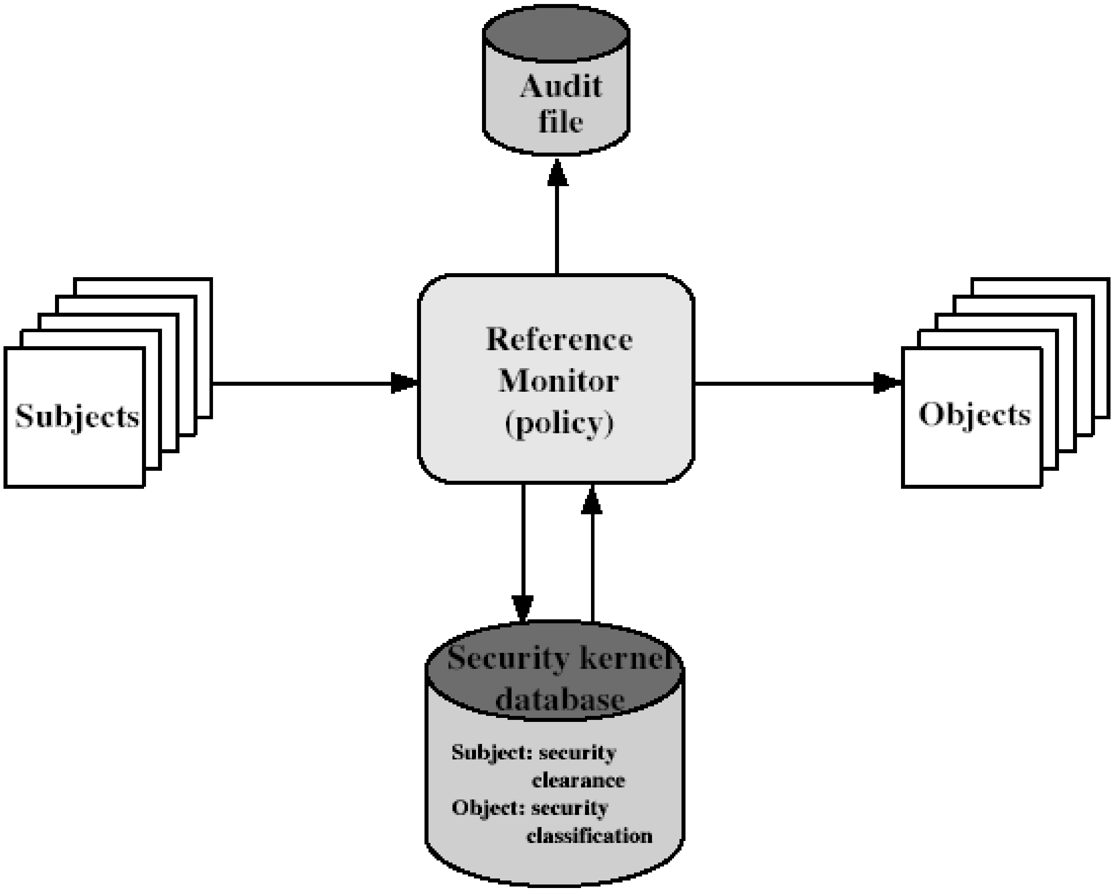
Fig: Access Control Structure

Decomposition by rows yields **capability tickets**. 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.

5.6.2 The concept of Trusted Systems

When multiple categories or levels of data are defined, the requirement is referred to as multilevel security. The general statement of the requirement for multilevel security is that a subject at a high level may not convey information to a subject at a lower or noncomparable level unless that flow accurately reflects the will of an authorized user. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system must enforce:

- **No read up:** A subject can only read an object of less or equal security level. This is referred to as **simple security property.**
- **No write down:** A subject can only write into an object of greater or equal security level.


Audit file
Reference Monitor (policy)
Subjects
Objects
Security kernel database
Subject: security clearance
Object: security classification

a
he
el
on
es
or
ed

authorized changes to the security kernel database, are stored in the audit file.

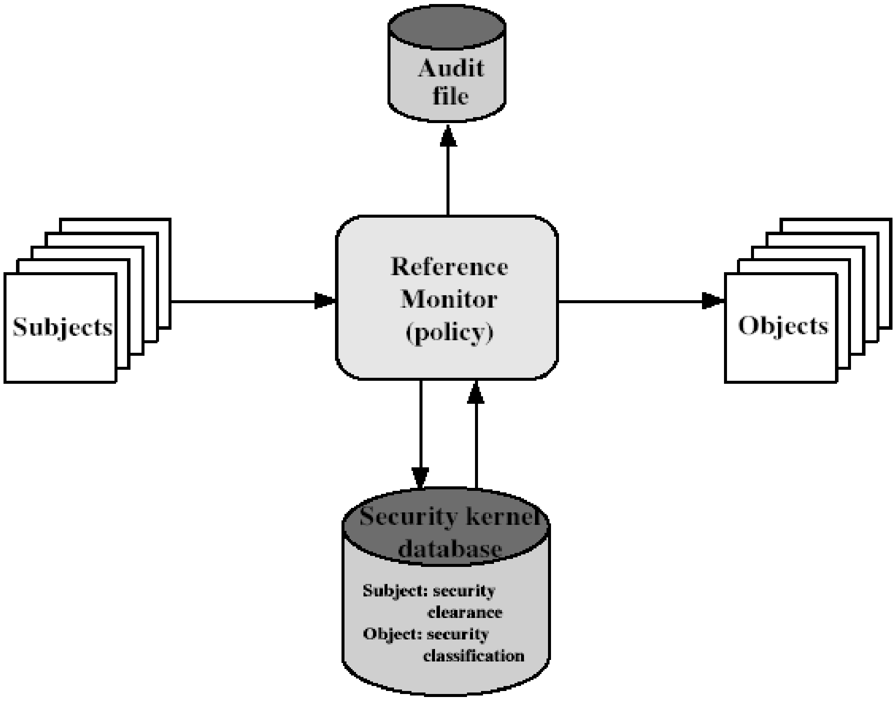

Audit file
Reference Monitor (policy)
Subjects
Objects
Security kernel database
Subject: security clearance
Object: security classification

Fig:Reference Monitor Concept

5.7 VIRUSES AND RELATED THREATS

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems.

5.7.1 Malicious Programs

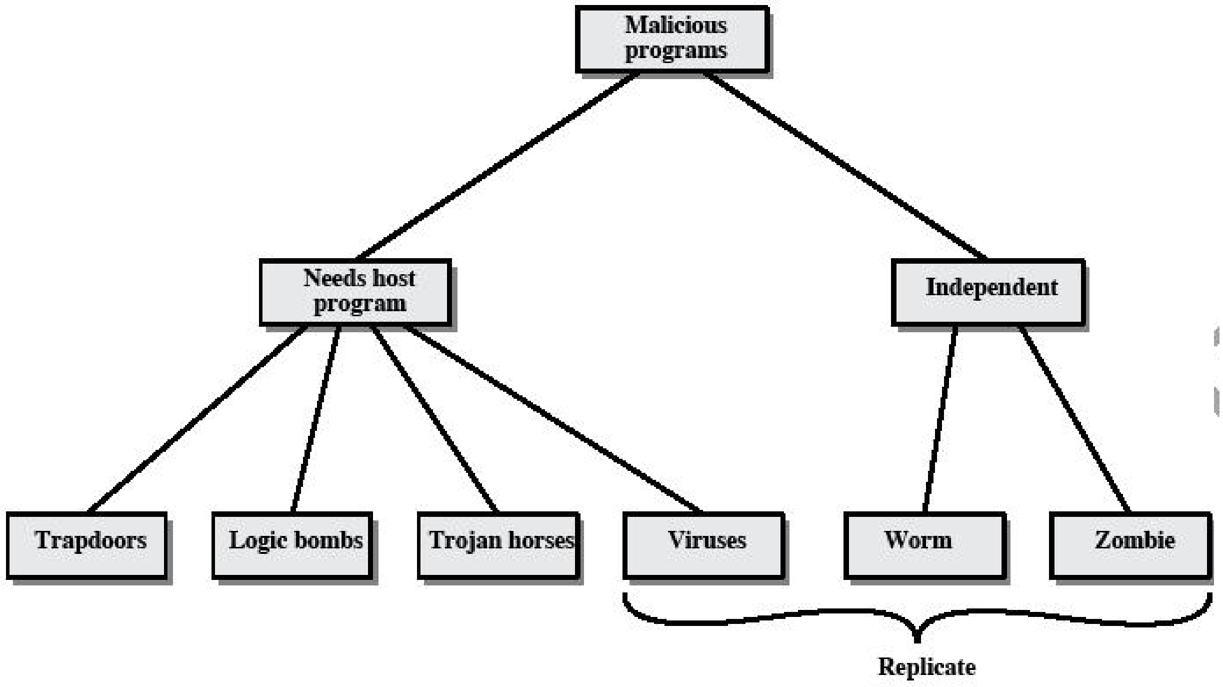


Figure 19.1 Taxonomy of Malicious Programs

Malicious software can be divided into two categories:

those that need a host program, and those that are independent.

The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples.

Name	**Description**
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs
Trojan horse	Program that contains unexpected additional functionality
Backdoor (trapdoor)	Program modification that allows unauthorized access to functionality
Exploits	Code specific to a single vulnerability or set of vulnerabilities
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access
Zombie	Program activated on an infected machine that is activated to launch attacks on other machines

5.7.2 The Nature of Viruses

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four phases:

- **Dormant phase**: The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.
- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.
- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

5.7.3 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.

An infected program begins with the virus code and works as follows.

The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus.

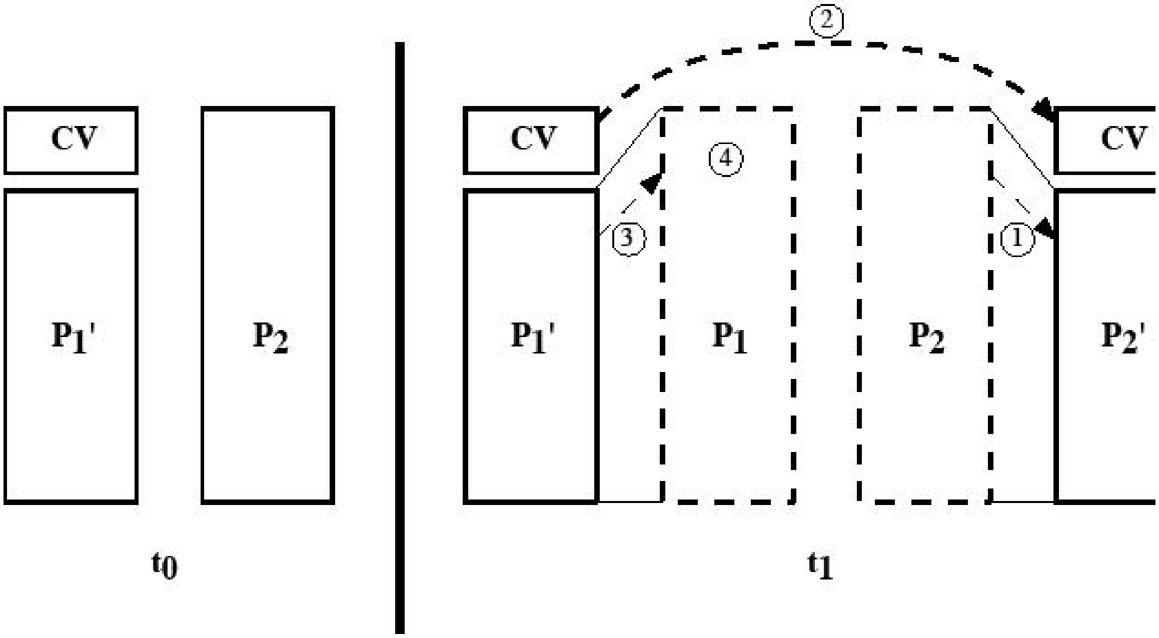
When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system.

This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions.

Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length.. The key lines in this virus are numbered. We assume that program P_1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file P_2 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.



In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.

5.7.4 Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

5.8 Types of Viruses

following categories as being among the most significant types of viruses:

- **Parasitic virus**: The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.
- **Memory-resident virus**: Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.
- **Boot sector virus**: Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.
- **Stealth virus**: A form of virus explicitly designed to hide itself from detection by antivirus software.
- **Polymorphic virus**: A virus that mutates with every infection, making detection by the "signature" of the virus impossible.
- **Metamorphic virus**: As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.
2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.
3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input.

Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.
2. The virus does local damage.

Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- Electronic mail facility: A worm mails a copy of itself to other systems.
- Remote execution capability: A worm executes a copy of itself on another system.
- Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.

2. Establish a connection with a remote system.

3. Copy itself to the remote system and cause the copy to be run.

As with viruses, network worms are difficult to counter.

The Morris Worm

The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation.

5. It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried

   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

6. It exploited a bug in the finger protocol, which reports the whereabouts of a remote user.
7. It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter.

Recent Worm Attacks

In late 2001, a more versatile worm appeared, known as Nimda. Nimda spreads by multiple mechanisms:

- from client to client via e-mail
- from client to client via open network shares
- from Web server to client via browsing of compromised Web sites
- from client to Web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities
- from client to Web server via scanning for the back doors left behind by the "Code Red II" worms

The worm modifies Web documents (e.g., .htm, .html, and .asp files) and certain executable files found on the systems it infects and creates numerous copies of itself under various filenames.

In early 2003, the SQL Slammer worm appeared. This worm exploited a buffer overflow vulnerability in Microsoft SQL server.

Mydoom is a mass-mailing e-mail worm that appeared in 2004

5.9 VIRUS COUNTERMEASURES

Antivirus Approaches

The ideal solution to the threat of viruses is prevention: The next best approach is to be able to do the following:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.
- **Identification**: Once detection has been achieved, identify the specific virus that has infected a program.
- **Removal**: Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

There are four generations of antivirus software:

- First generation: simple scanners
- Second generation: heuristic scanners
- Third generation: activity traps
- Fourth generation: full-featured protection

A first-generation scanner requires a virus signature to identify a virus.. Such signature-specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A second-generation scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures.

Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

Generic Decryption

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds . In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

- **CPU emulator:** A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.

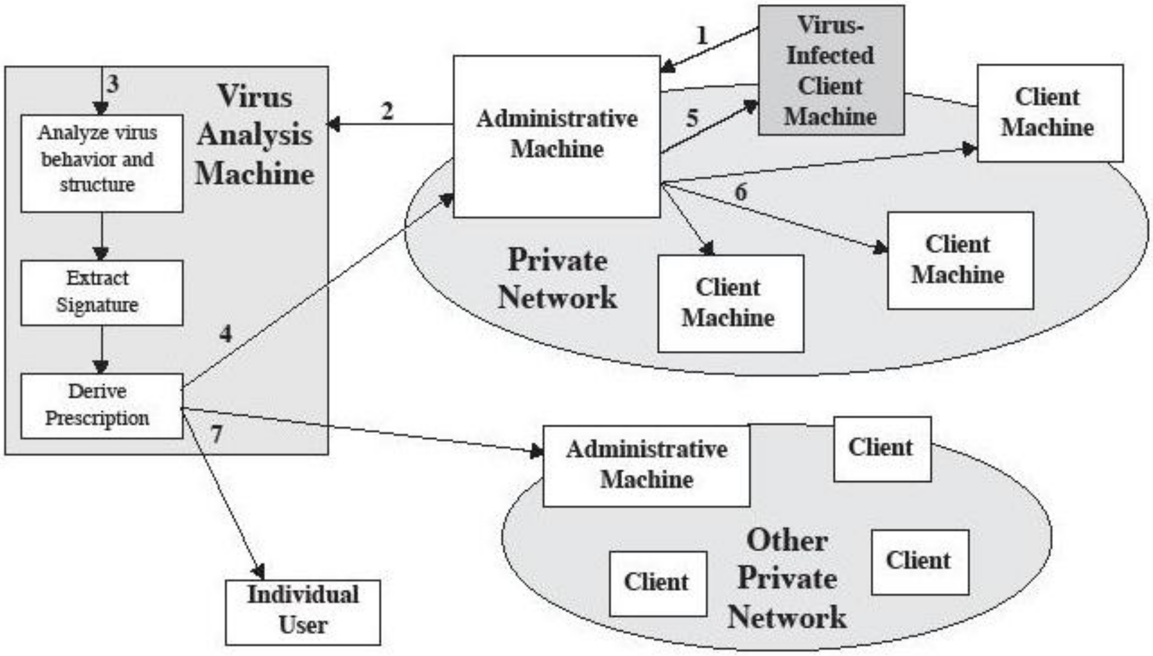
- **Virus signature scanner:** A module that scans the target code looking for known virus signatures.

- **Emulation control module:** Controls the execution of the target code.

Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM]. The motivation for this development has been the rising threat of Internet-based virus propagation.Two major trends in Internet technology have had an increasing impact on the rate of virus propagation in recent years:

- Integrated mail systems: Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.
- Mobile-program systems: Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.



1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.
2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.
3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.
4. The resulting prescription is sent back to the administrative machine.
5. The administrative machine forwards the prescription to the infected client.
6. The prescription is also forwarded to other clients in the organization.
7. Subscribers around the world receive regular antivirus updates that protect them from the new virus.

The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

Behavior-Blocking Software

Unlike heuristics or fingerprint-based scanners, behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real-time for malicious actions. Monitored behaviors can include the following:

- Attempts to open, view, delete, and/or modify files;
- Attempts to format disk drives and other unrecoverable disk operations;
- Modifications to the logic of executable files or macros;
- Modification of critical system settings, such as start-up settings;
- Scripting of e-mail and instant messaging clients to send executable content; and
- Initiation of network communications.

If the behavior blocker detects that a program is initiating would-be malicious behaviors as it runs, it can block these behaviors in real-time and/or terminate the offending software. This gives it a fundamental advantage over such established antivirus detection techniques as fingerprinting or heuristics.