اسم التدريسي: شهد عبدالرحمن حسو

Introduction to data security

With the introduction of the computer, the need for automated tools for protecting files and other information stored on the computer became evident. This is especially the case for a shared system, such as a time-sharing system, and the need is even more acute for systems that can be accessed over a public telephone network, data network, or the Internet. The generic name for the collection of tools designed to protect data and to thwart hackers is **computer security**.

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

Cryptology: This is the study of techniques for ensuring the secrecy and/or authenticity of information. The two main branches of cryptology are **cryptography**, which is the study of the design of such techniques; and **cryptanalysis**, which deals with the defeating such techniques, to recover information, or forging information that will be accepted as authentic.

Network security: This area covers the use of cryptographic algorithms in network protocols and network applications.

Computer security: we use this term to refer to the security of computers against intruders (e.g., hackers) and malicious software (e.g., viruses). Typically, the computer to be secured is attached to a network and the bulk of the threats arise from the network.

Cryptanalysis

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

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

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

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

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

● **Security attack:** Any action that compromises the security of information owned by an organization.

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

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

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

A useful means of classifying security attacks, is in terms of *passive attacks* and *active attacks*. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.

Passive Attacks

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

The **release of message contents** is easily understood (Figure 1.a). A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

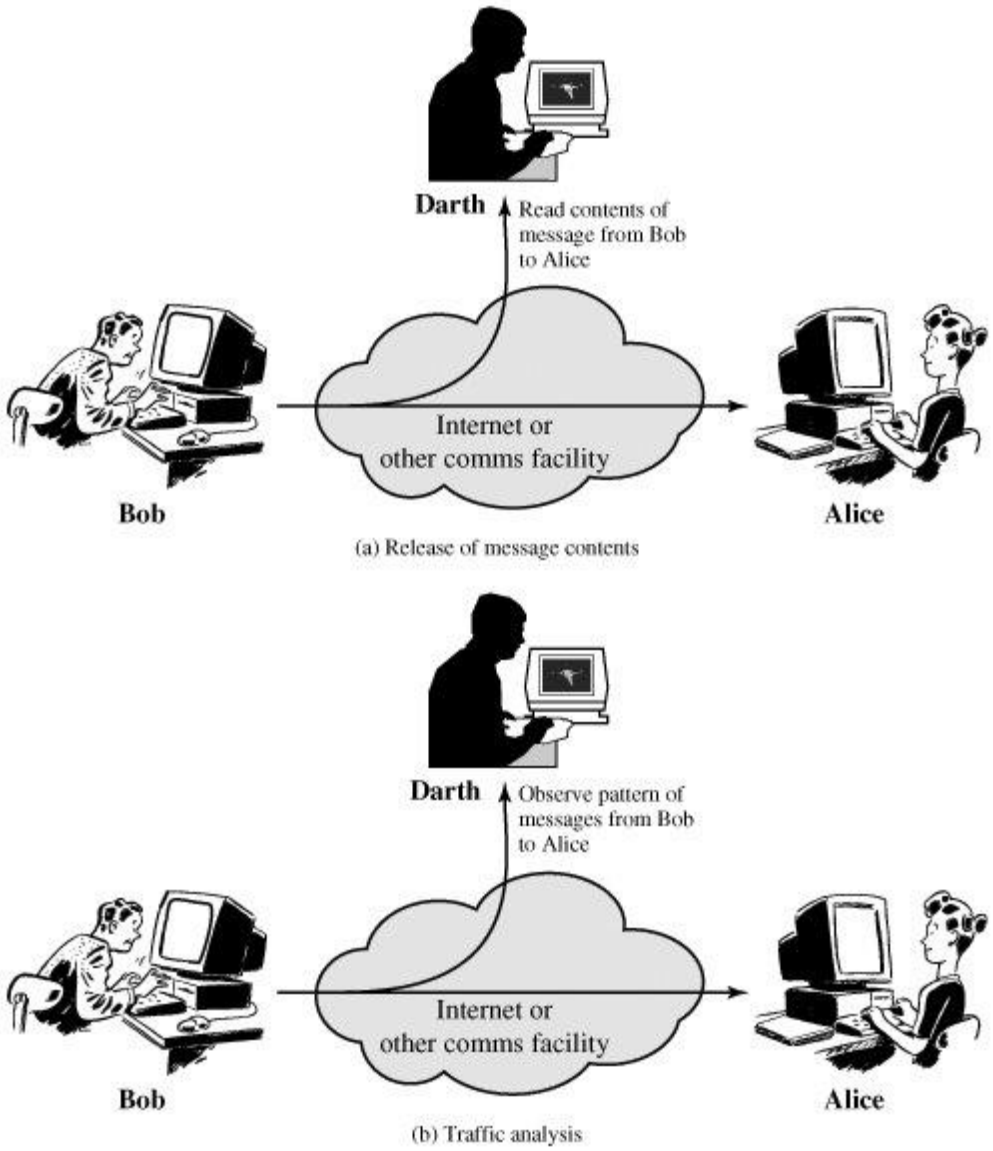


Figure (1) Passive Attacks

A second type of passive attack, **traffic analysis**, is subtler (Figure 1.b). Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being

exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. Typically, the message traffic is sent and received in an apparently normal fashion and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern.

However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

Active Attacks

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

A **masquerade** takes place when one entity pretends to be a different entity (Figure 2. a). A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect (Figure 2.b).

Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect (Figure 2.c). For example, a message meaning "Allow John Smith to read confidential file *accounts*" is modified to mean "Allow Fred Brown to read confidential file *accounts.*"

The **denial of service** prevents or inhibits the normal use or management of communications facilities (Figure 2.d). This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages.

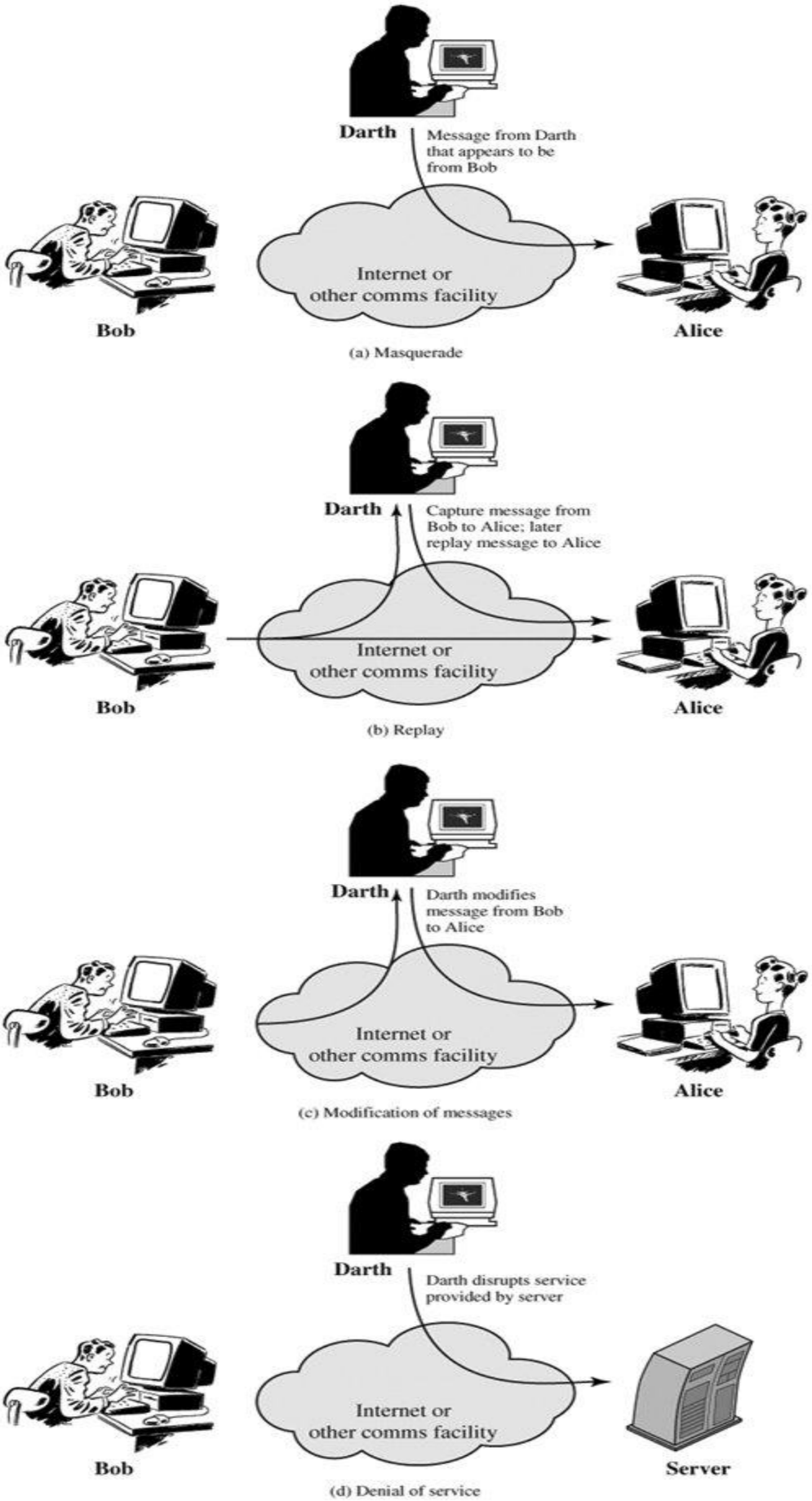


Figure (2) active attack

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

Authentication

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from. In the case of an on going interaction, such as the connection of a terminal to a host, two aspects are involved. First, at the time of connection initiation, the service assures that the two entities are authentic, that is, that each is the entity that it claims to be. Second, the service must assure that the connection is not interfered with in such a way that a third party can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception.

Access Control

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

Data Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. With respect to the content of a data transmission, several levels of protection can be identified. The broadest service protects all user data transmitted between two users over a period of time.

Data Integrity

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

A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion,

modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service, one that deals with individual messages without regard to any larger context, generally provides protection against message modification only.

Nonrepudiation

Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

Symmetric Cipher Model

A symmetric encryption scheme has five ingredients (Figure 3):

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

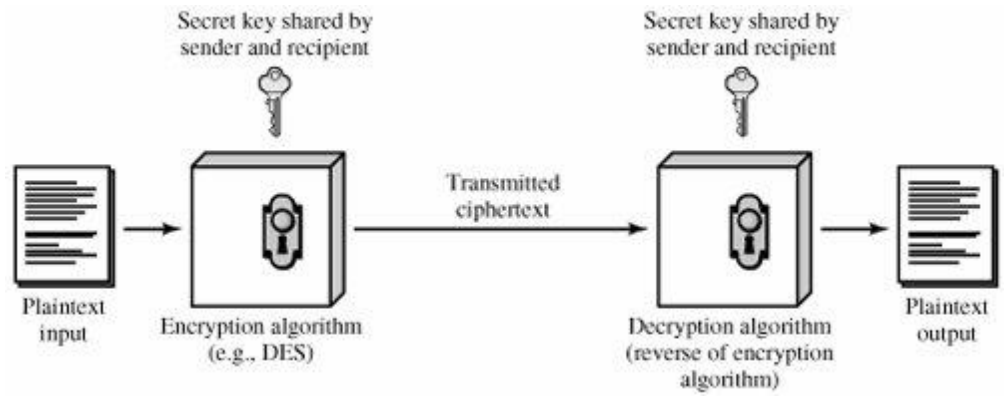


Figure (3) Simplified Model of Conventional Encryption

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

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 4.

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

With the message X and the encryption key K as input, the encryption algorithm forms the ciphertext $Y= [Y1, Y2, ..., YN]$. We can write this as

$Y = E(K, X)$

This notation indicates that Y is produced by using encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key K. The intended receiver, in possession of the key, is able to invert the transformation:

$X = D(K, Y)$

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

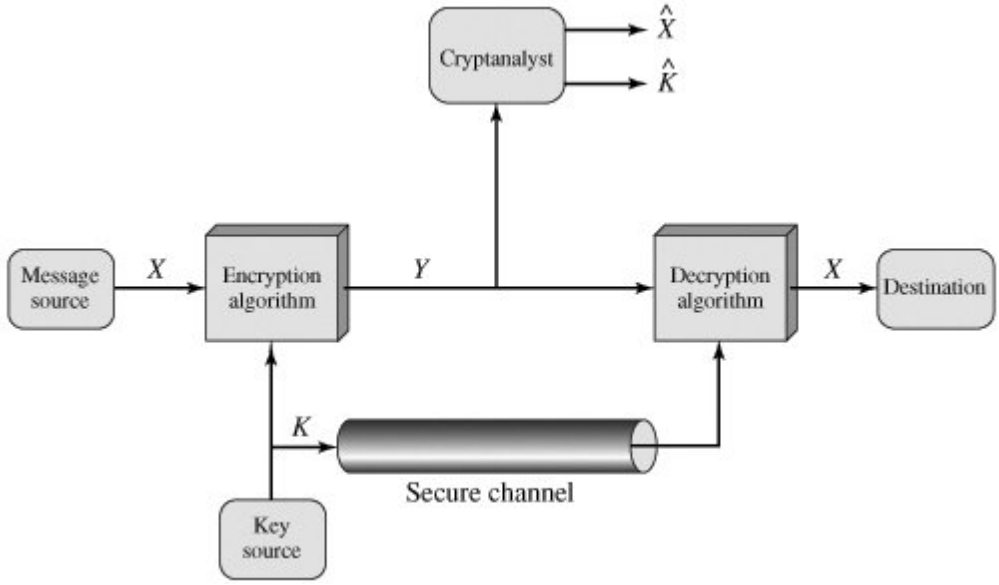


Figure (4) Model of Conventional Cryptosystem

Cryptographic systems are characterized along three independent dimensions:

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

2.The number of keys used. If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.

3.The way in which the plaintext is processed. A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

Transposition Techniques

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.

The simplest such cipher is the rail fence technique, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, to encipher the message "meet me after the toga party" with a rail fence of depth 2, we write the following:

m e m a t r h t g p r y
e t e f e t e o a a t

The encrypted message is

MEMATRHTGPRYETEFETEOAAT

This sort of thing would be trivial to cryptanalyze. 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 the columns then becomes the key to the algorithm. For example,

Key: **4 3 1 2 5 6 7**
Plaintext: a t t a c k p
o s t p o n e
d u n t i l t
w o a m x y z
Ciphertext: **TTNAAPTMTSUOAODWCOIXKNLYPETZ**

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with column positions. Digram and trigram frequency tables can be useful.

The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed. Thus, if the foregoing message is re-encrypted using the same algorithm,

Key: **4 3 1 2 5 6 7**
Input: t t n a a p t
m t s u o a o
d w c o i x k

n l y p e t z

Output: **NSCYAUOPTTWLTMDNAOIEPAXTTOKZ**

Substitution technique

we examine a sampling of what might be called classical encryption techniques. A study of these techniques enables us to illustrate the basic approaches to symmetric encryption used today and the types of cryptanalytic attacks that must be anticipated.

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

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols.1 If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

Caesar cipher

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

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

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

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

Let us assign a numerical equivalent to each letter:

a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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

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

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

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

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

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

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: simply try all the 25 possible keys. Figure (3) shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

Three important characteristics of this problem enabled us to use a bruteforce cryptanalysis:

1. The encryption and decryption algorithms are known.

2. There are only 25 keys to try.

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

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Figure 3 .Brute-Force Cryptanalysis of Caesar Cipher

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 ciphertext digrams.

The Playfair algorithm is based on the use of a 5 × 5 matrix of letters constructed using a keyword. Here is an example,

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

In this case, the keyword is *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 alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.

2. Two 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 circularly following the last. For example, ar is encrypted as RM.

3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.

4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are 26 × 26 = 676 digrams, so that identification of individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable.

Hill Cipher

Another interesting multiletter cipher is the Hill cipher, this encryption algorithm takes successive plaintext letters and substitutes for them ciphertext letters. The substitution is determined by

linear equations in which each character is assigned a numerical value . For , the system can be described as

$c1 = (k11P1 + k12P2 + k13P3) \bmod 26$

$c2 = (k21P1 + k22P2 + k23P3) \bmod 26$

$c3 = (k31P1 + k32P2 + k33P3) \bmod 26$

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

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

Or

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

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

For example, consider the plaintext "paymoremoney" and use the encryption key

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

The first three letters of the plaintext are represented by the vector $\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$. Then $\mathbf{K}\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix}$ = LNS. Continuing in this fashion,

the ciphertext for the entire plaintext is LNSHDLEWMTRW.

Decryption requires using the inverse of the matrix **K**. The inverse **K**1 of a matrix **K** is defined by the equation **KK**1 = **K**1**K** = **I**, where **I** is the matrix that is all zeros except for ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation. In this case, the inverse is:

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

This is demonstrated as follows:

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

Polyalphabetic Ciphers

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

1. A set of related monoalphabetic substitution rules is used.

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

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

To aid in understanding the scheme and to aid in its use, a matrix known as the Vigenère tableau is constructed (Table 1). 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 ciphertext letter is at the intersection of the row labeled *x* and the column labeled y; in this case the ciphertext is V.

Table (1) The Modern Vigenère Tableau

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

encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

key: ***deceptivedeceptivedeceptive***
plaintext: wearediscoveredsaveyourself
ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

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

The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 2.6 shows the frequency distribution for a Vigenère cipher with a keyword of length 9. An improvement is achieved over the Playfair cipher, but considerable frequency information remains. It is instructive to sketch a method of breaking this cipher, because the method reveals some of the mathematical principles that apply in cryptanalysis.

Transposition Techniques

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.

The simplest such cipher is the rail fence technique, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, to encipher the message "meet me after the toga party" with a rail fence of depth 2, we write the following:

m e m a t r h t g p r y
e t e f e t e o a a t

The encrypted message is

MEMATRHTGPRYETEFETEOAAT

This sort of thing would be trivial to cryptanalyze. 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 the columns then becomes the key to the algorithm. For example,

Key: **4 3 1 2 5 6 7**
Plaintext: a t t a c k p
o s t p o n e
d u n t i l t
w o a m x y z
Ciphertext: **TTNAAPTMTSUOAODWCOIXKNLYPETZ**

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with column positions. Digram and trigram frequency tables can be useful.

The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed. Thus, if the foregoing message is re-encrypted using the same algorithm,

Key: **4 3 1 2 5 6 7**
Input: t t n a a p t
m t s u o a o
d w c o i x k
n l y p e t z
Output: **NSCYAUOPTTWLTMDNAOIEPAXTTOKZ**

Block cipher

Stream Ciphers and Block Ciphers

A stream cipher is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher. A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

Motivation for the Feistel Cipher Structure

A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits. There are2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular.

The Feistel Cipher

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

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations. In fact, this is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions.

Diffusion and Confusion

The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system. Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or

phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the ciphertext, the cryptanalyst may be able to deduce the encryption key, or part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the ciphertext are independent of the particular key used.

Other than recourse to ideal systems, Shannon suggests two methods for frustrating statistical cryptanalysis: diffusion and confusion. In diffusion, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally this is equivalent to having each ciphertext digit be affected by many plaintext digits. An example of diffusion is to encrypt a message $M = m1, m2, m3,...$ of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter yn. One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of ciphertext.

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

Feistel Cipher Structure

Figure (8) depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K. The plaintext block is divided into two halves, $L0$ and $R0$. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs $Li-1$ and $Ri-1$, derived from the previous round, as well as a subkey Ki, derived from the overall K. In general, the subkeys Ki are different from K and from each other.

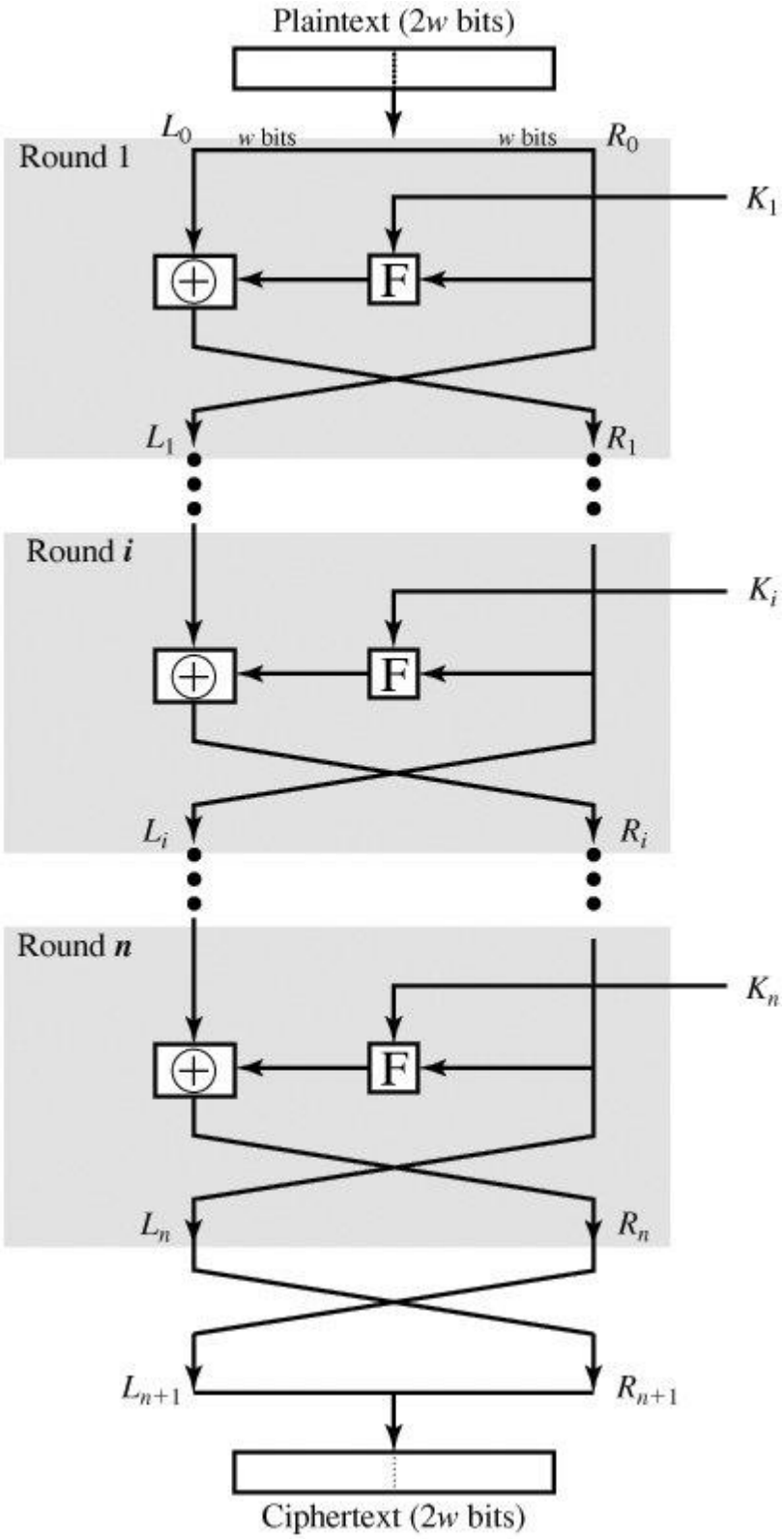


Figure (8) Classical Feistel Network

All rounds have the same structure. A substitution is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is

parameterized by the round subkey *Ki*. Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data.

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

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

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

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

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

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

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

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

● **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength.

The Data Encryption Standard (DES)

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA). For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

DES Encryption

The overall scheme for DES encryption is illustrated in Figure (9). As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length. Actually, the function expects a 64-bit key as input. However,

only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily.

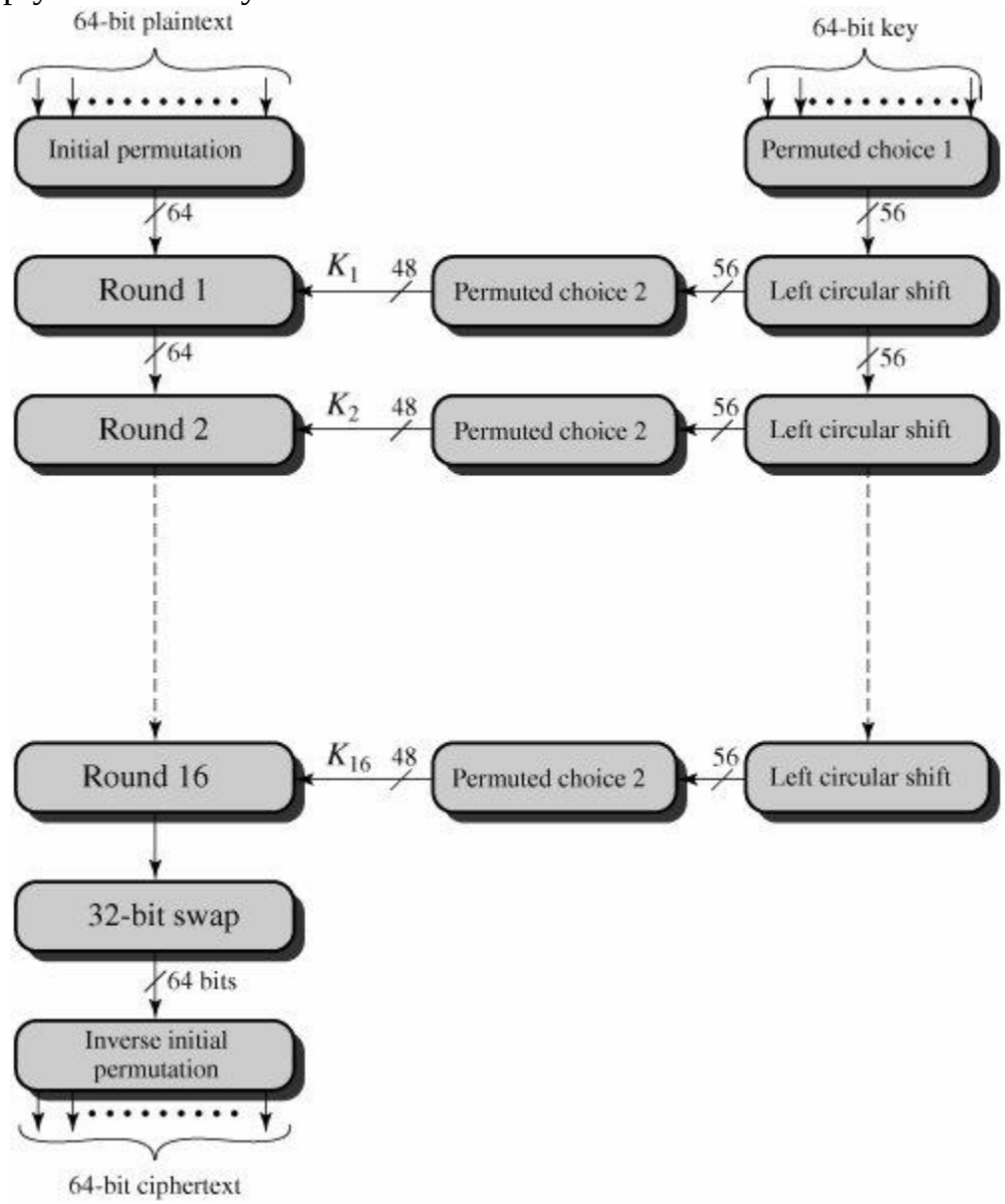


Figure (9) General Depiction of DES Encryption Algorithm

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput. Finally, the preoutput is passed through a permutation (IP-1) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure (8).

The right-hand portion of Figure (9) shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a *subkey* (*Ki*) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

Initial Permutation

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

Table 1. Permutation Tables for DES

(a) Initial Permutation (IP)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7
(b) Inverse Initial Permutation (IP1)							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25
(c) Expansion Permutation (E)							
	32	1	2	3	4	5	
	4	5	6	7	8	9	
	8	9	10	11	12	13	
	12	13	14	15	16	17	
	16	17	18	19	20	21	
	20	21	22	23	24	25	
	24	25	26	27	28	29	
	28	29	30	31	32	1	
(d) Permutation Function (P)							
16	7	20	21	29	12	28	17

1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input *M*:

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

where *Mi* is a binary digit. Then the permutation $X = \text{IP}(M)$ is as follows:

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

If we then take the inverse permutation $Y = \text{IP}^{-1}(X) = \text{IP}^{-1}(\text{IP}(M))$, it can be seen that the original ordering of the bits is restored.

Details of Single Round

Figure (10) shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$Li = Ri\text{-}1$

$Ri = \text{L}i\text{-}1 \times \text{F}(Ri\text{-}1, Ki)$

The round key *Ki* is 48 bits. The *R* input is 32 bits. This *R* input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the *R* bits (Table1.c). The resulting 48 bits are XORed with *Ki*. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 1.d.

The role of the S-boxes in the function F is illustrated in Figure 3.6. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box S*i* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S*i*. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to

produce the output. For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

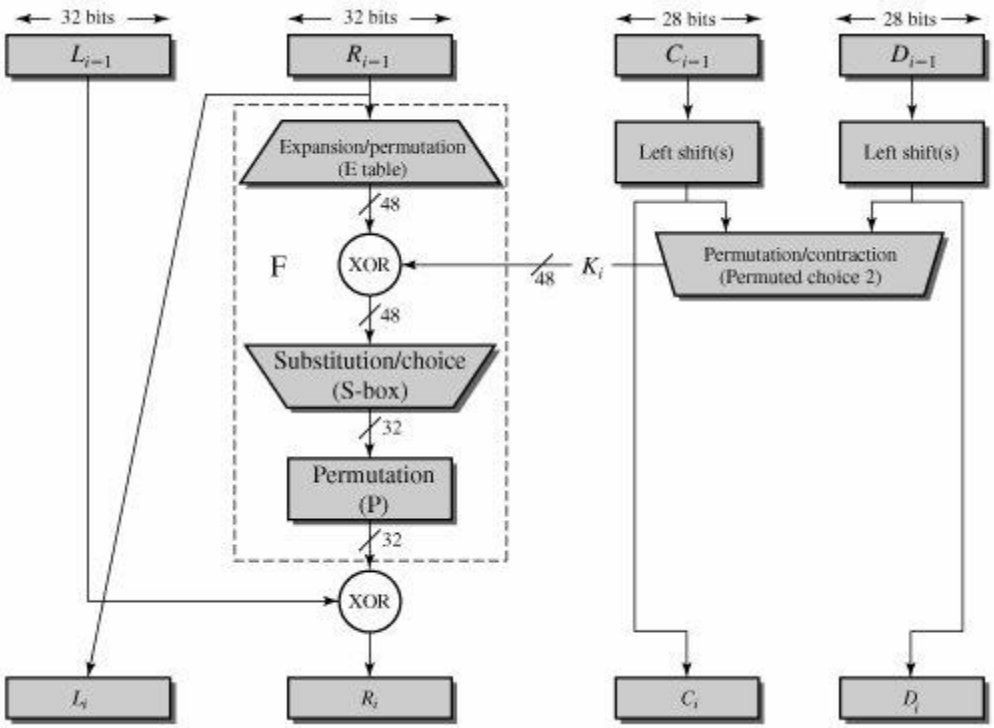


Figure (10) Single Round of DES Algorithm

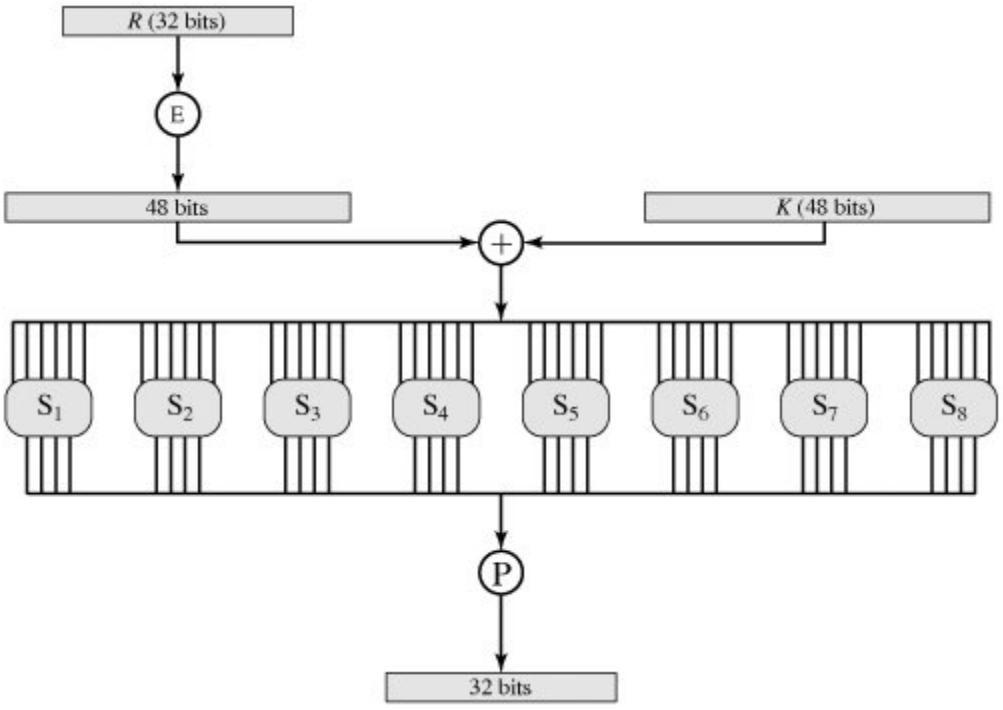


Figure (11) Calculation of F(R, K)

Table (2). Definition of DES S-Boxes

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

The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (*Ki*). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits by taking the outer bits from the two adjacent groups. The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-

bit output from the eight S-boxes is then permuted, so that on the next round the output from each S-box immediately affects as many others as possible.

Key Generation

Returning to Figures (8) and (9), we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled *C*0 and *D*0. At

each round, *Ci*-1 and *Di*-1 are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 3d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 3c), which produces a 48-bit output that serves as input to the function F(*Ri*-1, *Ki*).

Table 3 DES Key Schedule Calculation

(a) Input Key
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
(b) Permuted Choice One (PC-1)
57 49 41 33 25 17 9
1 58 50 42 34 26 18
10 2 59 51 43 35 27
19 11 3 60 52 44 36
63 55 47 39 31 23 15
7 62 54 46 38 30 22
14 6 61 53 45 37 29
21 13 5 28 20 12 4
(c) Permuted Choice Two (PC-2)
14 17 11 24 1 5 3 28
15 6 21 10 23 19 12 4
26 8 16 7 27 20 13 2
41 52 31 37 47 55 30 40
51 45 33 48 44 49 39 56
34 53 46 42 50 36 29 32
(d) Schedule of Left Shifts
Round number 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Bits rotated 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

DES Decryption

Decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.

The Strength of Des

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

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2 x 1016. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered.

Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

Timing Attacks

In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

Public key cryptosystem

Asymmetric Keys: Two related keys, a public key and a private key, that are used to perform complementary operations, such as encryption and decryption or signature generation and signature verification.

Public Key Certificate: A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.

Public Key (Asymmetric) Cryptographic Algorithm: A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible. The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. The first problem is that of **key distribution.** The second problem that Diffie pondered, and one that was apparently unrelated to the first, was that of **digital signatures.** If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the equivalent of signatures used in paper documents.

Public-Key Cryptosystems

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

• It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key. In addition, some algorithms, such as RSA, also exhibit the following characteristic.

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

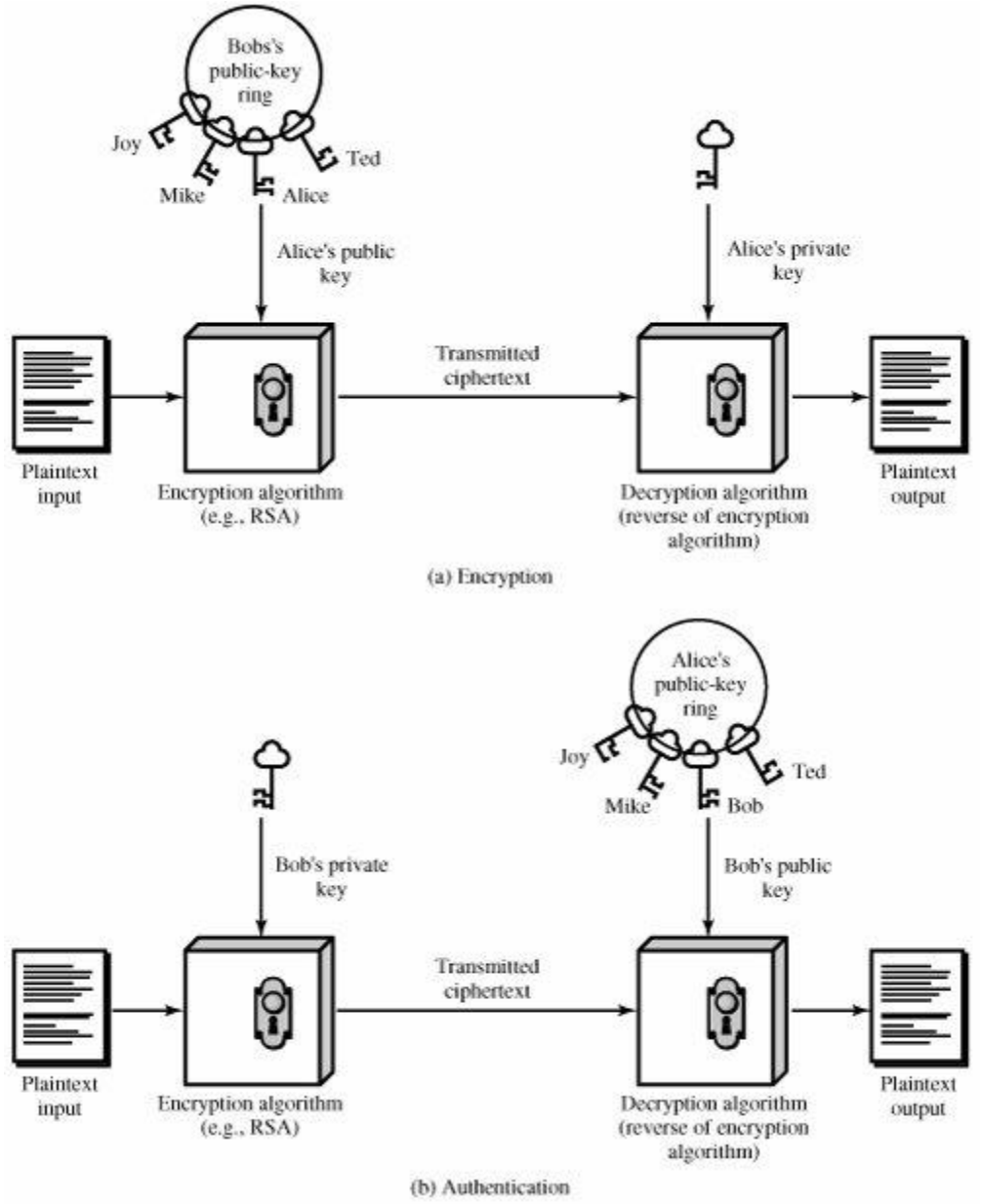


Figure (12) Public-Key Cryptography

The essential steps are the following:

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

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure (12).a suggests, each user maintains a collection of public keys obtained from others.

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

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

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As

long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key. Table 4 summarizes some of the important aspects of symmetric and public-key encryption.

Table 4. Conventional and Public-Key Encryption

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

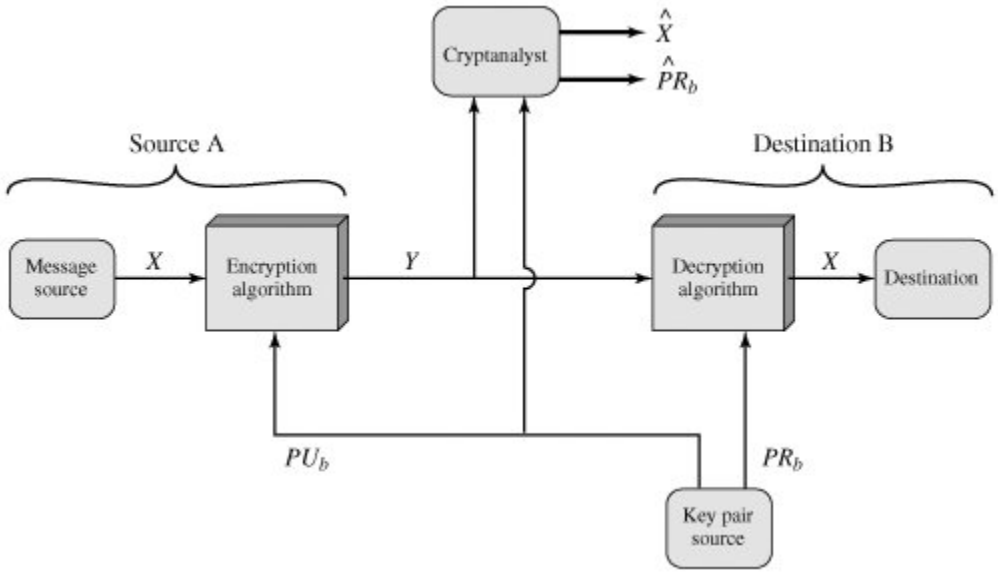


Figure (13) Public-Key Cryptosystem: Secrecy

With the message *X* and the encryption key *PUb* as input, A forms the ciphertext *Y* = [*Y*1, *Y*2,..., *YN*]: *Y* = E(*PUb*, *X*)

The intended receiver, in possession of the matching private key, is able to invert the transformation: *X* = D(*PRb*, *Y*)

An adversary, observing *Y* and having access to *PUb* but not having access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover *X*, by generating a plaintext estimate Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover *PRb* by generating an estimate.

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Figure (13) provides confidentiality, Figures (12).b and 14 show the use of public-key encryption to provide authentication:

Y = E(*PRa*, *X*)

Y = E(*PUa*, *Y*)

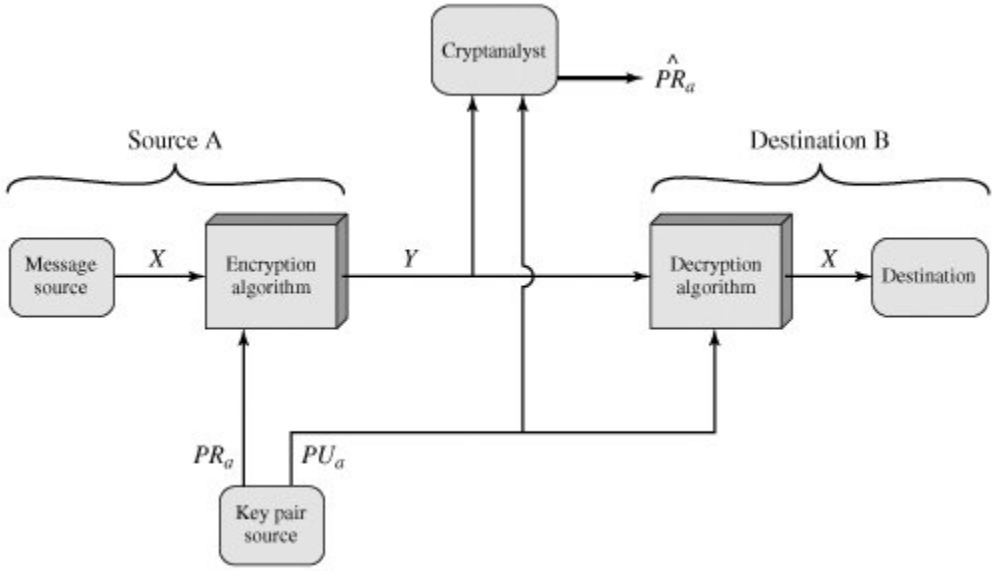


Figure (14) Public-Key Cryptosystem: Authentication

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

the message is authenticated both in terms of source and in terms of data integrity.

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

It is important to emphasize that the encryption process depicted in Figures (12).b and 14 does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in

Figure (14), 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 function and confidentiality by a double use of the public-key scheme (Figure (15)):

$Z = E(PUb, E(PRa, X))$

$X = D(PUa, E(PRb, Z))$

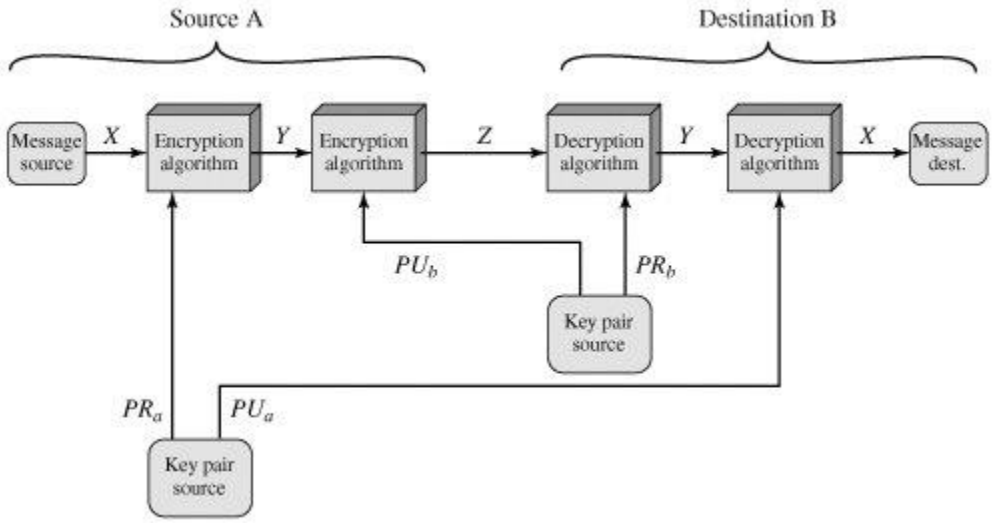


Figure (15) Public-Key Cryptosystem: Authentication and Secrecy

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

Applications for Public-Key Cryptosystems

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

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

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

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

Requirements for Public-Key Cryptography

1. It is computationally easy for a party B to generate a pair (public key *PUb*, private key *PRb*).

2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding ciphertext:

$C = E(PUb, M)$

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

$M = D(PRb, C) = D[PRb, E(PUb, M)]$

4. It is computationally infeasible for an adversary, knowing the public key, *PUb*, to determine the private key, *PRb*.

5. It is computationally infeasible for an adversary, knowing the public key, *PUb*, and a ciphertext, *C*, to recover the original message, *M*.

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

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

$M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)]$

Before elaborating on why the requirements are so formidable, let us first recast them. The requirements boil down to the need for a trap-door one-way function. A *one-way function* is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$Y = f(X)$ easy

$X = f1(X)$ infeasible

The RSA Algorithm

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and n 1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024}. We

examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of RSA.

Description of the Algorithm

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials.

Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to log2(n); in practice, the block size is i bits, where $2i < n < 2i+1$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$C = Me \bmod n$

$M = Cd \bmod n = (Me)d \bmod n = Med \bmod n$

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

the following requirements must be met:

1.It is possible to find values of e, d, n such that $Med \bmod n = M$ for all $M < n$.

2.It is relatively easy to calculate mod Me mod n and Cd for all values of $M < n$.

3.It is infeasible to determine d given e and n.

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

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

The preceding relationship holds if e and d are multiplicative inverses modulo f(n), where $\phi(n)$ is the Euler totient function.

$\phi(pq) = (p-1)(q-1)$ The relationship between e and d can be expressed as

$ed \bmod \phi(n)=1$

This is equivalent to saying

$ed \equiv 1 \bmod \phi(n)$

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

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

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

p,q, two prime numbers (private, chosen)

$n = pq$ (public, calculated)

e, with $\gcd(\phi(n),e) = 1; 1 < e < \phi(n)$ (public, chosen)

$d \equiv e^{-1} (\bmod \phi(n))$ (private, calculated)

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

the keys were generated as follows:

1.Select two prime numbers, $p = 17$ and $q = 11$.

2.Calculate $n = pq = 17 \times 11 = 187$.

3.Calculate f(n) = (p 1)(q 1) = 16 x 10 = 160.

4.Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$ we choose $e = 7$.

5.Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 * 7 = 161 = 10 \times 160 + 1$; d can be calculated using the extended Euclid's algorithm.

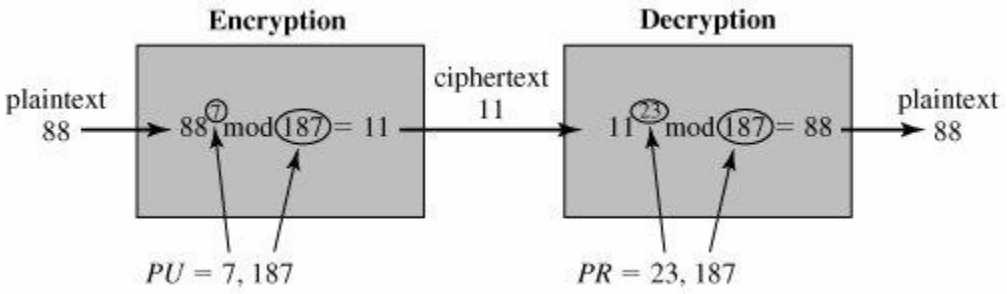


Figure (16) Example of RSA Algorithm

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

Exploiting the properties of modular arithmetic, we can do this as follows:

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

$88^1 \bmod 187 = 88$

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

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

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

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

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

$11^1 \bmod 187 = 11$

$11^2 \bmod 187 = 121$

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

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

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

Key Generation

Before the application of the public-key cryptosystem, each participant must generate a pair of keys. This involves the following tasks:

- Determining two prime numbers, p and q
- Selecting either e or d and calculating the other

First, consider the selection of p and q. Because the value of $n = pq$ will be known to any potential adversary, to prevent the discovery of p and q by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e., p and q must be large numbers). On the other hand, the method used for finding large primes must be reasonably efficient.

Steganography techniques

Steganography is the act of covert communications, which means that only the sender, Alice, and receiver, Bob, are aware of the secret communication. To accomplish this, the secret message is hidden within benign-looking communications known as covertexts or cover Works. To an adversary, Eve, it is clear that Alice and Bob are communicating, but the combined covertext and hidden message, referred to as a stegotext or stego Work, appears to be innocuous (i.e.,Eve is unaware that the innocuous content hides a message).

The main requirement of steganography is *undetectability*, which, loosely defined, means that no algorithm exists that can determine whether a Work contains a hidden message. Steganalysis is the process of detection of steganographic communications. And since steganography and steganalysis are closely intertwined.

Steganography and watermarking are both forms of data hiding and share some common foundations. Nevertheless, it is worth reiterating the goals of these two data-hiding applications in order to highlight the key differences.

We define steganography as the practice of undetectably altering a Work to embed a message.

We define watermarking as the practice of imperceptibly altering a Work to embed a message about that Work.

When designing a steganographic scheme, we need to consider issues such as the properties of the communication channel, the source of cover Works, and the embedding/extraction function.

The central concept in steganography is statistical undetectability. Without a precise definition, the field of steganography would lack the criterion to evaluate how secure steganographic schemes really are.

STEGANOGRAPHIC COMMUNICATION

Two prisoners, Alice and Bob, are under the surveillance of a warden, Eve. The warden will permit Alice and Bob to communicate, but all communications must go through the warden. If the warden thinks that Alice's message to Bob is innocuous, she may simply forward it to Bob. Alternatively, she may intentionally distort the content (e.g., apply lossy compression) in the hope that such a distortion will remove any secret message that just might be present. If the warden thinks Alice's message to Bob hides a covert communication, then she may block the communication entirely.

This framework, which models the applications depicted in Figure (1). A number of different assumptions can be made regarding the channel, the source of cover Works, and the embedding and extraction functions.

. **Figure (1)Steganographic embedding scheme**

The Channel

In steganography, the physical channel used for communication is generally assumed noise free, as this can be ensured using error correction and standard Internet protocols. Instead, the channel's properties are defined by the warden. The warden is considered a part of the channel because she may, or may not, interfere with the communication. As such, there are three types of warden:

passive, active, and malicious.

The warden is called **passive** if she is restricted from modifying the content sent by Alice prior to receipt by Bob (i.e., the warden can only prevent or permit delivery of Alice's message). In this scenario, the warden tests each communication from Alice for the presence of a covert message. If the warden's test is negative, the communication is relayed to Bob. Otherwise it is blocked.

This is the most commonly assumed scenario and why most steganographic algorithms are not designed to be robust.

The warden is called **active** if she intentionally modifies the content sent by Alice prior to receipt by Bob. In this scenario, the warden may not be entirely confident of her steganalysis program. Thus, even though her tests are negative, the warden may alter the content, hoping that the modification will destroy any steganographic message that *might* be present. If the steganographic algorithm assumes a passive warden, then there is a good chance that alterations to the content will severely degrade or remove the hidden message. The types of modification an active warden might apply include lossy recompression of images and audio clips, low-pass filtering, and other procedures that slightly degrade the content.

The warden is called **malicious** if her actions are based on the specifics of the steganographic scheme and are aimed at catching the prisoners communicating

secretly. This may include the warden trying to impersonate Alice or Bob or otherwise tricking them. A malicious warden is usually considered in public-key steganography. In this scenario, the *stego* key is known and anyone can extract the secret message. However, the message is encrypted using a public-key cryptosystem. Only those who possess Bob's private key can decipher Alice's message. Even though the stego key is known, it is difficult to distinguish between an encrypted message and a random bit sequence extracted from a cover Work. Nevertheless, since the Warden also knows the stego key, she has more options to attack the stego system.

The Building Blocks

The main building blocks of any steganographic algorithm are:

1. The choice of the cover Work.

2. The embedding and extracting algorithms, which might include

a. Symbol assignment function.

b. The embedding modification.

c. The selection rule.

3. Stego key management.

We now discuss each of these design elements in more detail.

Unlike a watermark, a steganographic message says nothing about the cover Work in which it is hidden. Consequently, the steganographer is free to choose a particular cover Work from his or her source of covers. The main restriction is the source of cover Works, which is determined by the resources available to Alice and Bob, by the warden herself, and the context in which the communication takes place. For example, an oppressive regime can specify allowable forms of messages and Alice must comply with them to avoid being caught. Or, if Alice and Bob communicate by posting images to a discussion newsgroup, they must choose the covers among those that are typically posted. But even with these restrictions, there are still numerous cover Works in which to hide the covert message. Alice is therefore at liberty to choose the cover Work which, after embedding, has the least likelihood of being detected.

For example, it is intuitively clear that noisy or highly textured images will better mask any embedding changes than high-quality images with little content (e.g., blue sky images). Alternatively, Alice can think ahead and attempt to guess what tests the warden is going to use and embed the same message into many different

covers, run known steganalysis attacks on each stego Work, and then simply send the cover that passes the tests.

Fundamentally, an embedding function can be based on three different principles, namely:

1. The cover Works are preexisting and the embedder does *not* modify the

cover Works. This is referred to as steganography by cover lookup.

2. The cover Works are generated based on the hidden message and the embedder does *not* modify the cover Works. This is referred to as cover synthesis.

3. The cover Works are preexisting and the embedder modifies the cover Works. This is referred to as steganography by cover modification.

Steganography by cover modification describes methods where Alice alters an existing cover Work to create a stego Work that conveys the desired message.

This approach is both the most common and the most advanced. we will only focus on this class of steganographic algorithms.

The type of changes introduced by the embedder, together with the location of these changes within the cover Work, have a major influence on how inconspicuous the embedded message will be. Intuitively, changes of large magnitude will be more obvious than changes of smaller magnitude. Consequently,

most steganographic schemes try to modify the cover Work as little as possible.

The location of the changes is controlled by the *selection rule*. There are three types of selection rules: sequential, (pseudo) random, and adaptive.

A sequential selection rule embeds the message bits in individual elements of the cover Work in a sequential manner, for example, starting in the upper left corner of an image and proceeding in a row-wise manner to the lower right corner. Although the sequential selection rule is the easiest one to implement, it provides poor security, since steganalysis algorithms can inspect the statistical properties of pixels in the same order, looking for a sudden change in behavior.

A pseudo-random selection rule embeds the message bits in a pseudo randomly

selected subset of the cover Work. The sender might first use a secret stego key, *Ks*, to initialize a pseudo-random number generator (PRNG) that in turn generates a pseudo-random walk through the cover Work. The message bits are then embedded into the elements constituting this walk. Pseudo-random selection rules typically offer better security than sequential rules.

An adaptive selection rule embeds the message bits at locations that are determined based on the content of the cover Work. The motivation for this is that statistical detectability is likely to depend on the content of the cover Work as well The process of embedding is controlled by a secret key shared between Alice and Bob. The key can be used for several different purposes. As previously mentioned, the key may seed a pseudo-random number generator to generate a random walk through the cover Work. It can also be used to generate other pseudo-random entities needed for embedding.

The primary goal of steganography is to design embedding functions that are statistically undetectable and capable of communicating practical (i.e., large) payloads.

NOTATION AND TERMINOLOGY

We now mathematically define a steganographic scheme. Let *Ks* denote a stego key drawn from a set, K, of all secret stego keys, M the set of all embeddable messages, and C the set of all cover Works. A steganographic scheme is formed by two mappings, the embedding mapping, *Emb*, and the extraction mapping, *Ext*:

Emb: C ×K×M→C

Ext: C →M,

such that *Ext*(*Emb*(**c**, *Ks* ,**m**)) = **m** for all **c** ∈ C, *Ks*

∈ K, and **m**∈M. The Work

s = *Emb*(**c**, *Ks* ,**m**) is called the stego Work.

The embedding algorithm *Emb* takes the cover Work, the secret key, and the message as its input and produces the modified stego Work.

Cover Work may be (text, image, audio, video).

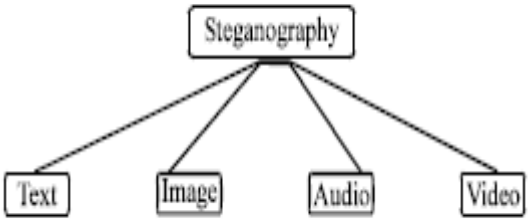


IMAGE STEGANOGRAPHY

Image steganography techniques can be divided into two groups: those in the Image Domain and those in the Transform Domain. Image - also known as spatial - domain techniques embed messages in the intensity of the pixels directly, while for transform - also known as frequency - domain, images are first transformed and then the message is embedded in the image.

Image domain techniques encompass bit-wise methods that apply bit insertion and noise manipulation and are sometimes characterised as simple systems. The image formats that are most suitable for image domain steganography are lossless and the techniques are typically dependent on the image format.

Steganography in the transform domain involves the manipulation of algorithms and image transforms. These methods hide messages in more significant areas of the cover image, making it more robust. Many transform domain methods are independent of the image format and the embedded message may survive conversion between lossy and lossless compression. In the next sections steganographic algorithms will be explained in categories according to image file formats and the domain in which they are performed. Figure(2) indicate the possibility of using images as a cover carrier.

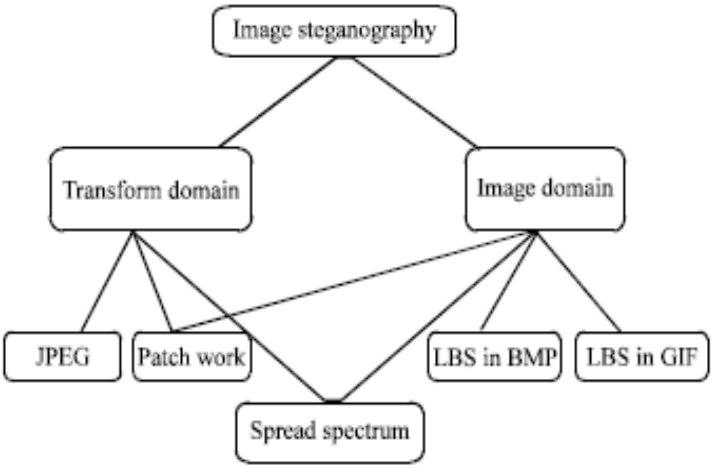


Figure (2) Image based steganography

Substitution Systems

Basic substitution systems try to encode secret information by substituting insignificant parts of the cover by secret message bits. The receiver can extract the information if he has knowledge of the positions where secret information has been embedded. Since only minor modifications are made in the embedding process, the

sender assumes that they will not be noticed by an attacker. It consists of several techniques that will be discussed in more detail, in the following subsection:

Least Significant Bit Substitution (LSB)

The embedding process consists of choosing a subset {j1… jl(m)} of cover elements and performing the substitution operation cji _ mi on them, which exchange the LSB of cji by mi (mi can be either 1 or 0). In the extraction process, the LSB of the selected cover-element is extracted and lined up to reconstruct the secret message.

In the case of a 24-bit bitmap each pixel is represented by 4 bytes. Of those, 3 bytes, or 24 bits, are used to store the red, green and blue values for the pixel. The fourth byte is reserved and should be zero. To store each character in the low order bit plane of the raster data, it is necessary to obtain an 8 bit representation of the character. For example, the character A is represented by the number 65. The equivalent binary representation is 0100 0001. Each of the 8 bits used to represent the letter A is then stored in the low order bit of one byte of raster data. Thus, to store a single letter, 8 bytes of raster data are consumed. This leads to a limit of embeddable information of size lengthOfRasterData/8. Consider hiding the letter A in the first 8 bits of raster data of an image. The first 8 bytes could possibly be (from left to right, top to bottom):

'1001 1001' '1110 0011' '0110 1001' '0001 1100'
'0001 1100' '0110 0100' '1011 0000' '1010 1001'

And the character A is:

'0100 0001'

Therefore, we need to set bits 7, 5, 4, 3, 2 and 1 to zero,

Although the resulting bit has not changed, we have ensured that the least significant bit has been set to '1'. Because the byte values for the red, green and blue pixels will only change by at most 1, the change in the resulting image will be imperceptible to the human eye. The resulting image will not, however, be well protected against statistical attack.

Pseudo Random Permutation

If all cover bits are accessed in the embedding process, the cover is a random access cover and the secret message bits can be distributed randomly over the whole cover. This technique further increases the complexity for the attacker, since it is not guaranteed that the subsequent message bits are embedded in the same order.

Image Downgrading and Cover Channels

Image downgrading is a special case of a substitution system in which image acts both as a secret message and a cover. Given cover-image and secret image of equal dimensions, the sender exchanges the four least significant bits of the cover grayscale (or colour) values with the four most significant bits of the secret image. The receiver extracts the four least significant bits out of the stego-image, thereby gaining access to the most significant bits of the stego-image. Whereas, the

degradation of the cover is not visually noticeable in many cases, four bits are sufficient to transmit a rough approximation of the secret image.

Cover Regions and Parity Bits

Any nonempty subset of {c1,……..,cI(c)} is called a cover-region. By dividing the cover into several disjoint regions, it is possible to store one bit of information in a whole cover-region rather than in a single element. A parity bit of a region I can be calculated by:

B (I) ==_LSB (cj) mod 2 j_I

We will call any nonempty subset of {*c*1, . . . ,*cl(c)}* a cover-region. By dividing the cover in several disjoint regions, it is possible to store one bit of information in a whole cover-region rather than in a single element. A *parity bit* of a region *I* can be calculated by

$$p(I) = \sum_{j \in I} LSB(c_j) \bmod 2 \qquad (3.3)$$

In the embedding step, *l(m)* disjoint cover-regions *Ii* ($1 \leq i \leq l(m)$) are selected, each encodes one secret bit *mi* in the parity bit *p(Ii).* If the parity bit of one cover-region *Ii* does not match with the secret bit *mi* to encode, one LSB of the values in *Ii* is flipped. This will result in *p(Ii) = mi.* In the decoding process, the parity bits of all selected regions are calculated and lined up to reconstruct the message. Again, the cover-regions can be constructed pseudo-randomly using the stego-key as a seed.

Transform Domain Techniques

We have seen that LSB modification techniques are easy ways to embed information, but they are highly vulnerable to even small cover modifications. An attacker can simply apply signal processing techniques in order to destroy the secret information entirely. In many cases even the small changes resulting out of lossy compression systems yield to total information loss.

It has been noted early in the development of steganographic systems that embedding information in the frequency domain of a signal can be much more robust than embedding rules operating in the time domain. Most robust steganographic systems known today actually operate in some sort of transform domain.

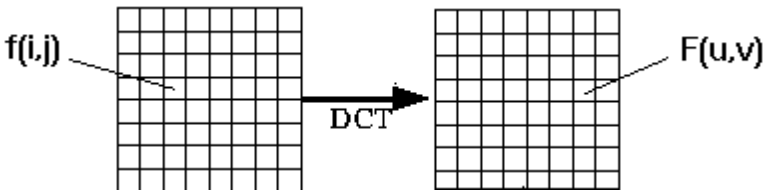
Transform domain methods hide messages in significant areas of the cover image which makes them more robust to attacks, such as compression, cropping, and some image processing, than the LSB approach. However, while they are more robust to various kinds of signal processing, they remain imperceptible to the human sensory system.

Many transform domain variations exist. One method is to use the discrete cosine transformation (DCT) as a vehicle to embed information in images; another would be the use of wavelet transforms. Transformations can be applied over the entire image, to blocks throughout the image, or other variations. However, a trade-off exists between the amount of information added to the image and the robustness obtained.

Many transform domain methods are independent to image format and may survive conversion between lossless and lossy formats.

Discrete Cosine Transform:

DCT commonly used for multimedia image/video compression. The discrete cosine transform (DCT) helps separate the image into parts with respect to the image's visual qualities. high, low & middle frequency components. The DCT transforms a signal or image from the spatial domain to the frequency domain.



A DCT is a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even proportion and computationally quite simpler than FFT.Much of the signal energy in image lies at low frequencies which appear in the upper left corner of the DCT. The lower right values represent higher frequencies which are small enough to be neglected with little visible distortion

2D DCT:

For 2D N by M image 2D DCT is defined:

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}}\left(\frac{2}{M}\right)^{\frac{1}{2}}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\Lambda(u)\cdot\Lambda(v).$$
$$cos\left[\frac{\pi\cdot u}{2\cdot N}(2i+1)\right]cos\left[\frac{\pi\cdot v}{2\cdot M}(2j+1)\right]\cdot f(i,j)$$

And the corresponding „inverse“ 2D DCT transform is simple F-1(u,v), i.e:

$$f(i,j) = F^{-1}(u,v)$$
$$= \left(\frac{2}{N}\right)^{\frac{1}{2}}\sum_{u=0}^{N-1}\sum_{v=0}^{M-1}\Lambda(u)\cdot\Lambda(v).$$
$$cos\left[\frac{\pi.u}{2.N}(2i+1)\right]\cdot cos\left[\frac{\pi\cdot v}{2\cdot M}(2j+1)\right]\cdot F(u,v)$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for}\xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

Steganography in the DCT Domain

One popular method of encoding secret information in the frequency domain is modulating the relative size of two (or more) DCT coefficients within one image block.

Algorithm *DCT-Steg encoding process*
for *i = 1,...,l(M)* ***do***
choose one cover-block bi
Bi = D{bi}
if *mi = 0* ***then***
if *Bi(u1, v1) > Bi(u2, v2* ***then***
swap Bi(u1, v1) and Bi(u2, v2)
end if
else
if *Bi(u1, v1) < Bi(u2, v2)* ***then***
swap Bi(u1, v1) and Bi(u2, v2)
end if
end if
adjust both values so that |Bi(u1, v1) - Bi(u2, v2)| > x
b 'i = D-1{Bi}
end for
create stego-image out of all b 'i

During the encoding process, the sender splits the cover-image in 8×8 pixel blocks; each block encodes exactly one secret message bit. The embedding process starts with selecting a pseudorandom block *bi* which will be used to code the *i*th message bit. Let *Bi = D{bi}* be the DCT-transformed image block.

Before the communication starts, both sender and receiver have to agree on the location of two DCT coefficients, which will be used in the embedding process; let us denote these two indices by ($u1$, $v1$) and ($u2$, $v2$). The two coefficients should correspond to cosine functions with middle frequencies; this ensures that the information is stored in significant parts of the signal (hence the embedded information will not be completely damaged by JPEG compression). Furthermore, we can assume that the embedding process will not degenerate the cover heavily, because it is widely believed that DCT coefficients of middle frequencies have similar magnitudes. Since the constructed system should be robust against JPEG compression, we choose the DCT coefficients in such a way that the quantization values associated with them in the JPEG compression algorithm are equal. The coefficients (4,1) and (3,2) or (1,2) and (3,0) are good candidates. One block encodes a "1," if $Bi(u1, v1) > Bi(u2, v2)$, otherwise a "0." In the encoding step, the two coefficients are swapped if their relative size does not match with the bit to be encoded. Since the JPEG compression can (in the quantization step)

Algorithm DCT-Steg decoding process
for $i = 1,...,l(M)$ **do**
get cover-block *bi* associated with bit *i*
$Bi = D\{bi\}$
if $Bi(u1, v1) \leq Bi(u2, v2)$ **then**
$mi = 0$
else
$mi = 1$
end if
end for

affect the relative sizes of the coefficients, the algorithm ensures that $|Bi(u1, v1) - Bi(u2, v2)| > x$ for some $x > 0$, by adding random values to both coefficients. The higher x is, the more robust the algorithm will be against JPEG compression, however, at the expense of image quality. The sender then performs an inverse DCT to map the coefficients back into the space domain. To decode the picture, all available blocks are DCT-transformed. By comparing the two coefficients of every block, the information can be restored.

If the constant x and the location of the used DCT coefficients are chosen properly, the embedding process will not degenerate the cover visibly. We can expect this method to be robust against JPEG compression, since in the quantization process both coefficients are divided by the same quantization values. Their relative size will therefore only be affected in the rounding step.

Performing DCT for Steganography: To embed the data in image consider a cover image of MxN& the data is to be

concealed. First divide the image in 8x8 blocks. & perform 2D DCT on each block. This will generate a matrix of DCT

Coefficients. The lower right coefficients represent higher frequencies which are negligible. So in next step

quantization is applied on each 8x8 block to compress the block. Then LSB of DCT coefficients are replaced with data

bits (that are to be hidden). Perform Inverse DCT on each resulting block. Then combine all blocks to form a stego

image.

Intruders

One of the two most publicized threats to security is the intruder (the other is viruses), generally referred to as a hacker or cracker. There are three classes of intruders:

- **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 misuses 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.

Intrusion Techniques

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Generally, this requires the intruder to acquire information that should have been protected. In some cases, this information is in the form of a user password. With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user.

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

- One-way function: The system stores only the value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.

- Access control: Access to the password file is limited to one or a very few accounts.

If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password crackers, the following techniques for learning passwords are reported:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Tap the line between a remote user and the host system.

Intrusion Detection

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

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.
2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Figure 16 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an

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

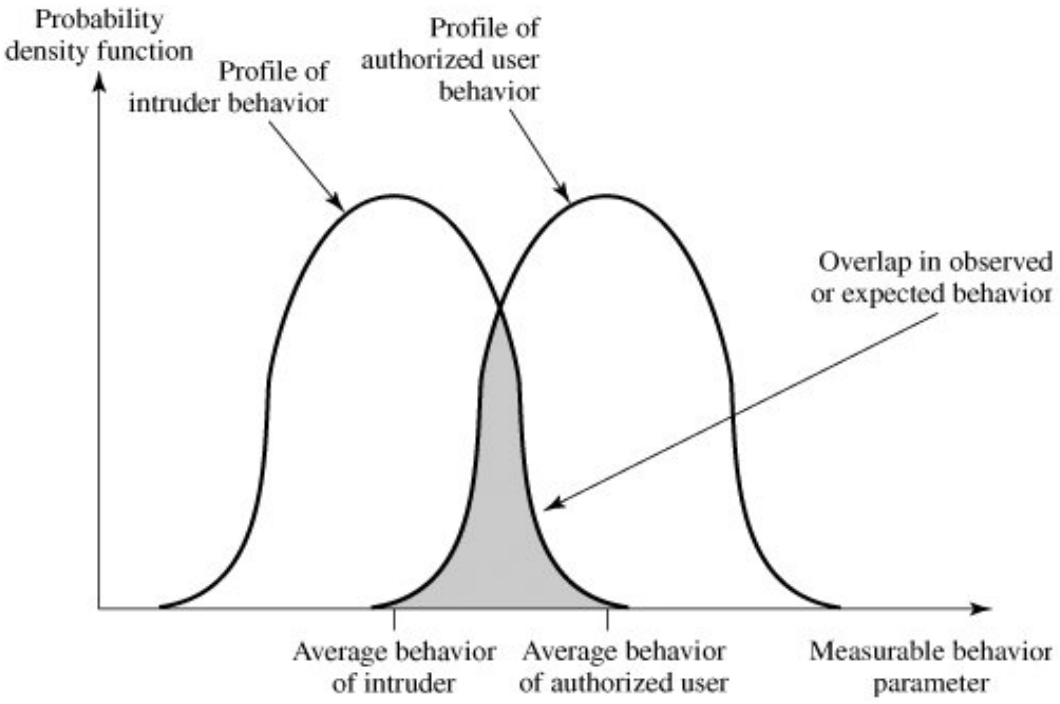


Figure 16. Profiles of Behavior of Intruders and Authorized Users

The following approaches to intrusion detection has been identified:

1. **Statistical anomaly detection**: Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.
    a. Threshold detection: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.
    b. Profile based: A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.
2. **Rule-based detection**: Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.
    a. Anomaly detection: Rules are developed to detect deviation from previous usage patterns.

b. Penetration identification: An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal, or expected, behavior, whereas rule-based approaches attempt to define proper behavior.

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.

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. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior.

An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.

Examples of metrics that are useful for profile-based intrusion detection are the following:

- **Counter:** A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.
- **Gauge:** A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.
- **Interval timer:** The length of time between two related events. An example is the length of time between successive logins to an account.
- **Resource utilization:** Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. The following approaches may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series
- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period. This gives a reflection of the average behavior and its variability. The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures. But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such

correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

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. In very general terms, we can characterize all approaches as focusing on either anomaly detection or penetration identification, although there is some overlap in these approaches.

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. In order for this approach to be effective, a rather large database of rules will be needed.

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. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage. Typically, the rules used in these systems are specific to the machine and operating system. Also, such rules are generated by "experts" rather than by means of an automated analysis of audit records. The normal procedure is to interview system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system. Thus, the strength of the approach depends on the skill of those involved in setting up the rules.

Early system used heuristic rules that can be used to assign degrees of suspicion to activities. 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.

Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

the following are the major issues in the design of a distributed intrusion detection system:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw

audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.

- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

A good example of a distributed intrusion detection system is one developed at the University of California at Davis. Figure 17 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.

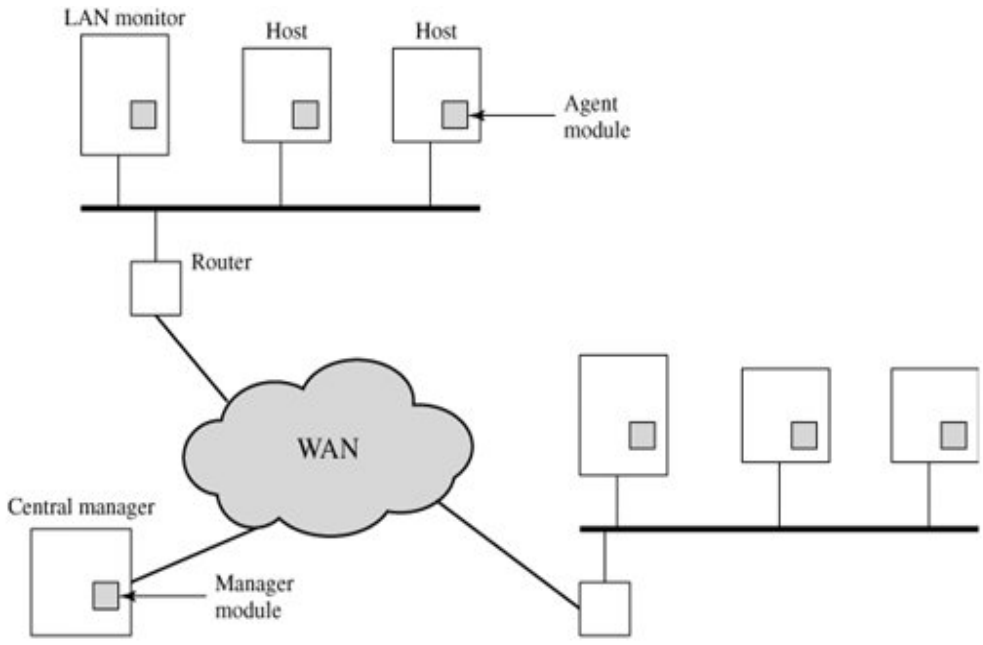


Figure 17. Architecture for Distributed Intrusion Detection

Password Management

Password Protection

The front line of defense against intruders is the password system. Virtually all multiuser systems require that a user provide not only a name or identifier (ID) but also a password. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

Selection Strategies

Many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords
- Reactive password checking
- Proactive password checking

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many

users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

Computer-generated passwords also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them.

The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The 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. For example, the dictionary used in the Purdue study occupies more than 30 megabytes of storage.
- Time: The time required to search a large dictionary may itself be large. In addition, to check for likely permutations of dictionary words, either those words most be included in the dictionary, making it truly huge, or each search must also involve considerable processing.

Message Authentication

Perhaps the most confusing area of network security is that of message authentication and the related topic of digital signatures. Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness.

Authentication Requirements:

In the context of communications across a network, the following attacks can be identified:

1. **Disclosure**: Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis**: Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
3. **Masquerade**: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or non receipt by someone other than the message recipient.
4. **Content modification**: Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification**: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
6. **Timing modification**: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
7. **Source repudiation**: Denial of transmission of message by source.
8. **Destination repudiation**: Denial of receipt of message by destination.

Authentication Functions

Any message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

Types of functions that may be used to produce an authenticator may be grouped into three classes, as follows:

- **Message encryption:** The ciphertext of the entire message serves as its authenticator
- **Message authentication code (MAC)**: A function of the message and a secret key that produces a fixed-length value that serves as the authenticator
- **Hash function:** A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator

Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Symmetric Encryption

Consider the straightforward use of symmetric encryption (Figure13a). A message M transmitted from source A to destination B is encrypted using a secret key K shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

In addition, we may say that B is assured that the message was generated by A. Why? The message must have come from A because A is the only other party that possesses K and therefore the only other party with the information necessary to construct ciphertext that can be decrypted with K. Furthermore, if M is recovered, B knows that none of the bits of M have been altered, because an opponent that does not know K would not know how to alter bits in the ciphertext to produce desired changes in the plaintext.

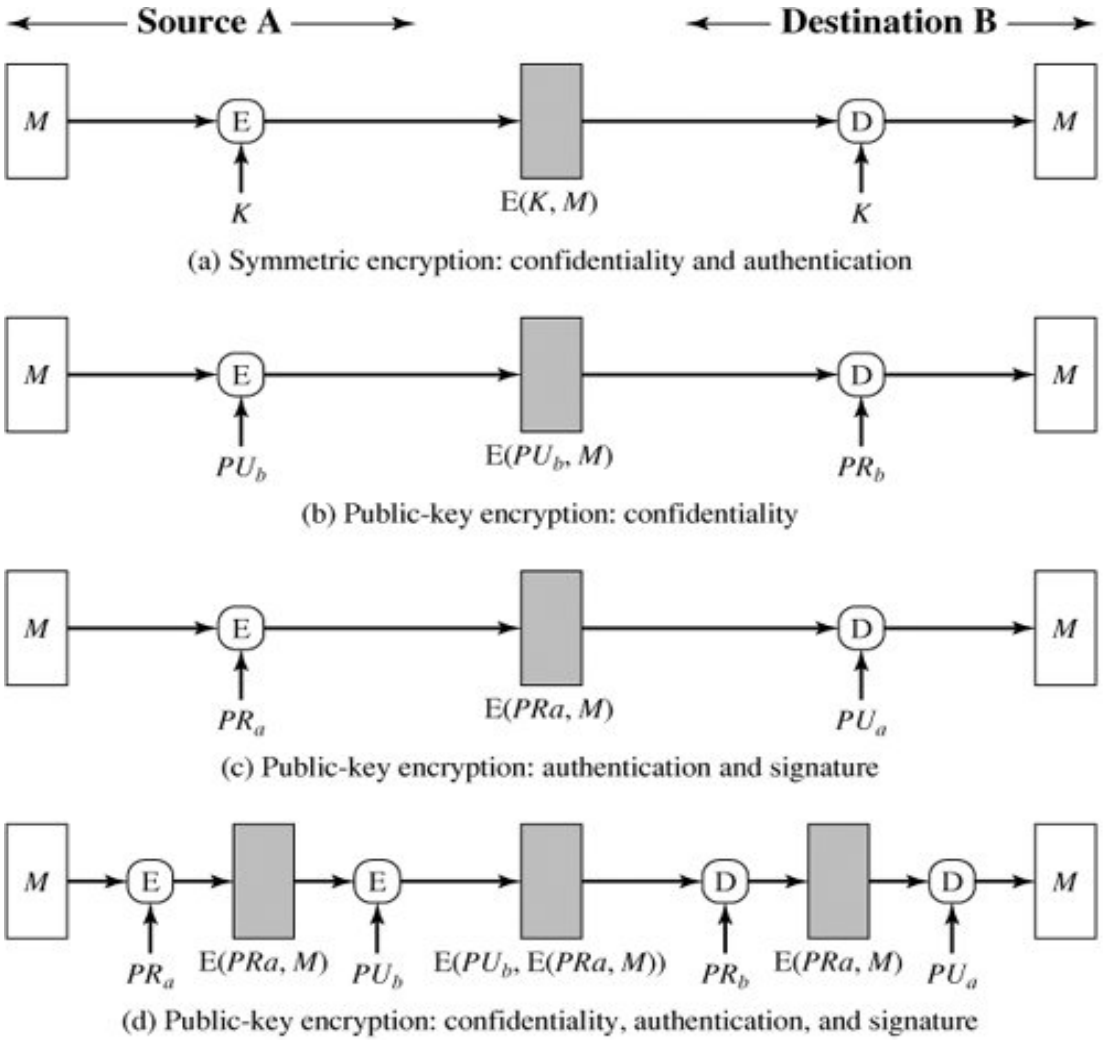


Figure 13. Basic Uses of Message Encryption

So we may say that symmetric encryption provides authentication as well as confidentiality.

Public-Key Encryption

The straightforward use of public-key encryption (Figure 13b) provides confidentiality but not authentication. The source (A) uses the public key PU_b of the destination (B) to encrypt M. Because only B has the corresponding private key PR_b, only B can decrypt the message. This scheme provides no authentication because any opponent could also use B's public key to encrypt a message, claiming to be A.

To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure 13c). This provides authentication using the same

type of reasoning as in the symmetric encryption case: The message must have come from A because A is the only party that possesses PR_a and therefore the only party with the information necessary to construct ciphertext that can be decrypted with PU_a. Again, the same reasoning as before applies: There must be some internal structure to the plaintext so that the receiver can distinguish between well-formed plaintext and random bits.

Assuming there is such structure, then the scheme of Figure 13c does provide authentication. It also provides what is known as digital signature.[1] Only A could have constructed the ciphertext because only A possesses PR_a. Not even B, the recipient, could have constructed the ciphertext. Therefore, if B is in possession of the ciphertext, B has the means to prove that the message must have come from A. In effect, A has "signed" the message by using its private key to encrypt. Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the ciphertext.

To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure 13d). The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Message Authentication Code

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key:MAC = C(K, M), where

M = input message

C = MAC function

K = shared secret key

MAC = message authentication code

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 14). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number, then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

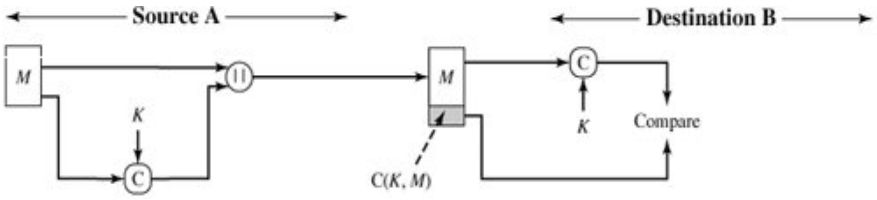


Figure 14. Basic Uses of Message Authentication Code (MAC)

Hash Function

A variation on the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message M as input and produces a fixed-size output, referred to as a **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. The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

Figure 15 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

a. The message plus concatenated hash code is encrypted using symmetric encryption. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.
b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

   Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 14). That is, E(K, H(M)) is a function

of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.

c. Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.
d. If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.
e. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.
f. Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

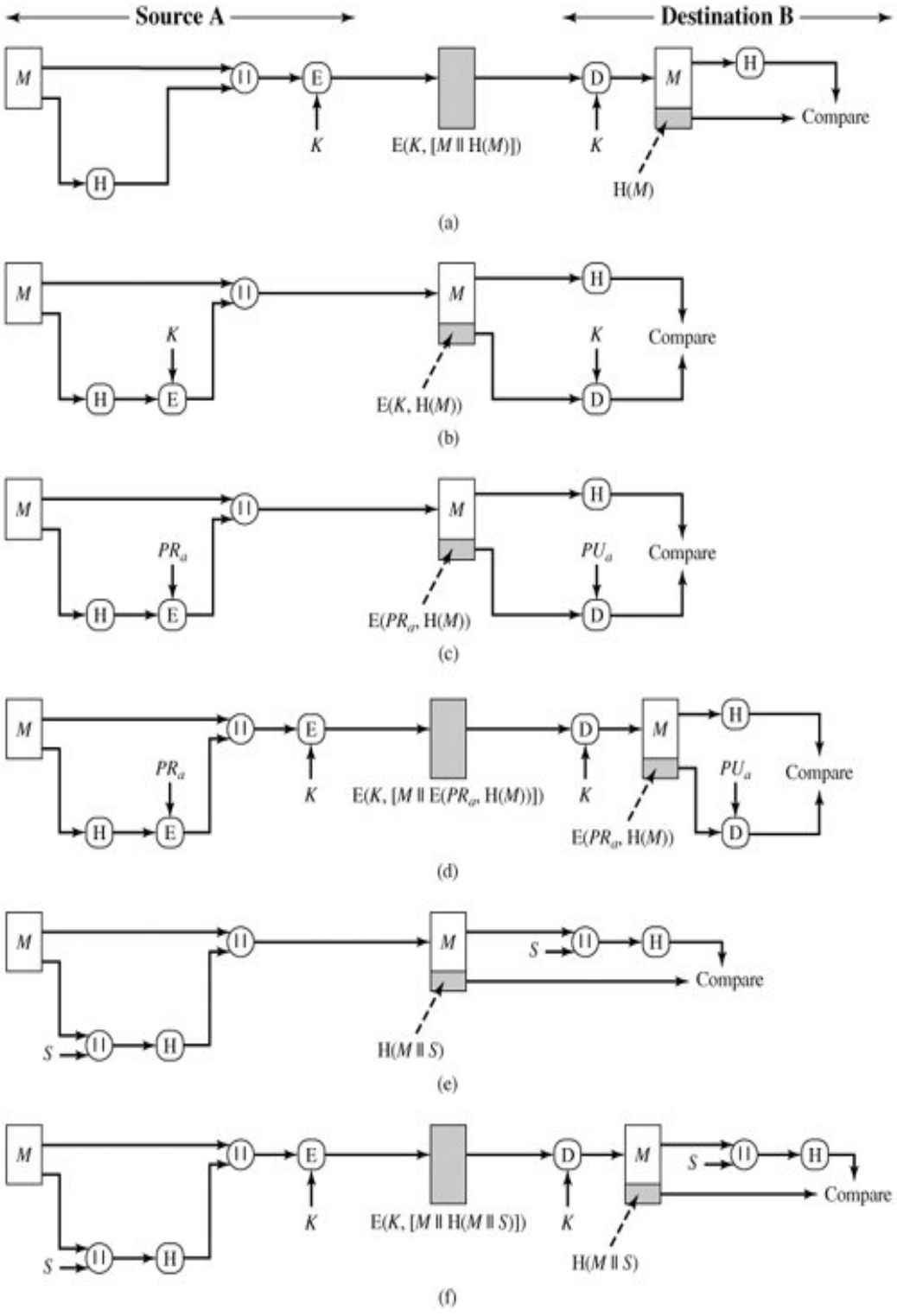


Figure 15. Basic Uses of Hash Function

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

h = H(M)

where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value (Figure 15).

Requirements for a Hash Function

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

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

Simple Hash Functions

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

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

$$C_i = b_{i1} \oplus b_{i1} \oplus \ldots \oplus b_{im}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check.

Computer Viruses

What Are Computer Viruses?

Computer viruses are small software programs that are designed to spread from one computer to another and to interfere with computer operation.

This definition is rather broad, because it contains everything from small viruses that duplicate your files and are just a mere annoyance to the dangerous ones that put a lock on your files and refuse to give you access to them until you pay a certain amount of money to its creator.

Virus attaches itself to files stored on floppy disks, USBs, email attachments and hard disks. A file containing a virus is called infected file. If this file is copied to a computer, virus is also copied to the computer.

Besides stealing information, computer viruses oftentimes:

- delete information off of your computer,
- corrupt files and make your computer act wonky and weird,
- use your e-mail address to spread itself to other users,
- take your files hostage until you pay a certain amount of money to the creator of the virus in order to release them.

In short: Computer viruses are small software programs created for malicious purposes that involve stealing information. They can easily hide themselves into small files such as images and attachments and infect your computer, after which they can multiply and send themselves as attachments using your e-mail address to other people in your list.
The damages each virus can do to your computer varies according to the category it fits in. While some may corrupt your files, multiply them or delete them, others go as far as deleting everything from your hard disk or taking your data hostage until you pay a certain fee.

Activation of Viruses

When the computer virus starts working, it is called the activation of virus. A virus normally runs all the time in the computer. Different viruses are activated in different ways. Many viruses are activated on a certain date. For example, a popular virus "Friday, the 13th" is activated only if the date is 13 and the day is Friday.

Causes of Computer Viruses:

The following are the main causes of a Computer Virus.

Infected Flash Drives or Disks

Flash drives and disks are the main cause of spreading viruses. Flash drives and disks are used to transfer data from one computer to other. A virus can also be copied from one computer to other when the user copies infected files using flash drives and disks.

Email Attachments

Most of the viruses spread through emails. Email attachment is a file that is sent along with an email. An email may contain an infected file attachment. Virus can spread if the users opens and downloads an email attachment. It may harm the computer when it is activated. It may destroy files on the hard disk or may send the virus automatically to all email addresses saved in the address book.

Infected websites

Thousands of insecure websites can infect computer with viruses. Most of the websites with suspicion materials are infected, so by visiting these websites the user's computer also gets infected by virus. These websites are developed to spread viruses. The virus is transferred to the user's computer when this material is downloaded. These websites may access the computer automatically when the users visit them.

Networks

Virus can spread if an infected computer is connected to a network. The internet is an example of such network. When a user downloads a file infected with virus from the internet, the virus is copied to the computer. It may infect the files stored on the computer.

Pirated Software

An illegal copy of software is called pirated software. Virus can spread if user installs pirated software that contains a virus. A variety of pirated software is available in CDs and from the internet. Some companies intentionally add virus in the software. The virus is automatically activated if the user uses the software without purchasing license.

Types of computer viruses

Basic types of viruses:

Boot viruses: Boot viruses attack the boot sectors on your hard drive and interfere with your computer's basic operation, making your operating system run strangely or even corrupt it all together .

Macro viruses: Macro viruses tend to attack data files, like word documents and spreadsheets, causing you to loose files or cause your word or excel software to not work properly .

Trojan viruses: Trojan viruses pretend to be other software, hence their name as in the Trojan horse. Trojan viruses pretend to be a legitimate piece of software, but in reality can attack your hard drives, deleting files and re-writing system files, causing your computer to become unstable, particular when operating system files are deleted .

As a general rule, computer viruses only attack files in your computer. They do not attack your computer's hardware, like the monitor, mouse or keyboard .

However, some viruses will attack the files that operate your computer's hardware, causing hard drives to reformat, video drivers to be deleted or your operating system to stop running. While this may cause your monitor to stop working properly, it doesn't mean you need to get a new monitor .

Structure of a Virus

A computer virus has three parts:

Infection mechanism: How a virus spreads, by modifying other code to contain a (possibly altered) copy of the virus. The exact means through which a virus spreads is referred to as its infection vector. This doesn't have to be unique - a virus that infects in multiple ways is called multipartite.
Trigger: The means of deciding whether to deliver the payload or not.
Payload: What the virus does, besides spread. The payload may involve damage, either intentional or accidental. Accidental damage may result from bugs in the virus, encountering an unknown type of system, or perhaps unanticipated multiple viral infections.

In pseudocode, a virus would have the structure below. The trigger function would return a Boolean, whose value would indicate whether or not the trigger conditions were met. The payload could be anything, of course.

```
def virus():
    infect()
    if trigger() is true:
        payload()
```

Infection is done by selecting some target code and infecting it, as shown below. The target code is locally accessible to the machine where the virus runs. Locally accessible targets may include code in shared network directories, though, as these directories are made to appear locally accessible. Generally, k targets may be infected each time the infection code below is run. The exact method used to select targets varies, and may be trivial, as in the case of the boot-sector infectors. The tricky part of select_target is that the virus doesn't want to repeatedly re-infect the same code; that would be a waste of effort, and may reveal the presence of the virus. Select_target has to have some way to detect whether or not some potential target code is already infected, which is a double-edged sword. If the virus can detect itself, then so can anti-virus software. The infect _code routine performs the actual infection by placing some version of the virus' code in the target.

```
def infect():
    repeat k times:
        target = select_target()
        if no target:
            return
        infect_code(target)
```

File Infectors

Operating systems have a notion of files that are executable. A file infector is a virus that infects files which the operating system considers to be executable; Where is the virus placed?

Beginning of a File

Very simple executable file formats like the .COM MS-DOS format would treat the entire file as a combination of code and data. When executed, the entire file would be loaded into memory, and execution would start by jumping to the beginning

of the loaded file. In this case, a virus that places itself at the start of the file gets control
first when the infected file is run, as illustrated in Figure 1. This is called a prepending virus.

End of a File

Appending code onto the end of a file is extremely easy. A virus that places itself at the end of a file is called an appending virus. How does the virus get control? There are two basic possibilities:
• The original instruction(s) in the code can be saved, and replaced by a jump to the viral code. Later, the virus will transfer control back to the code it infected. The virus may try to run the original instructions directly in their saved location, or the virus may restore the infected code back to its original state and run it.

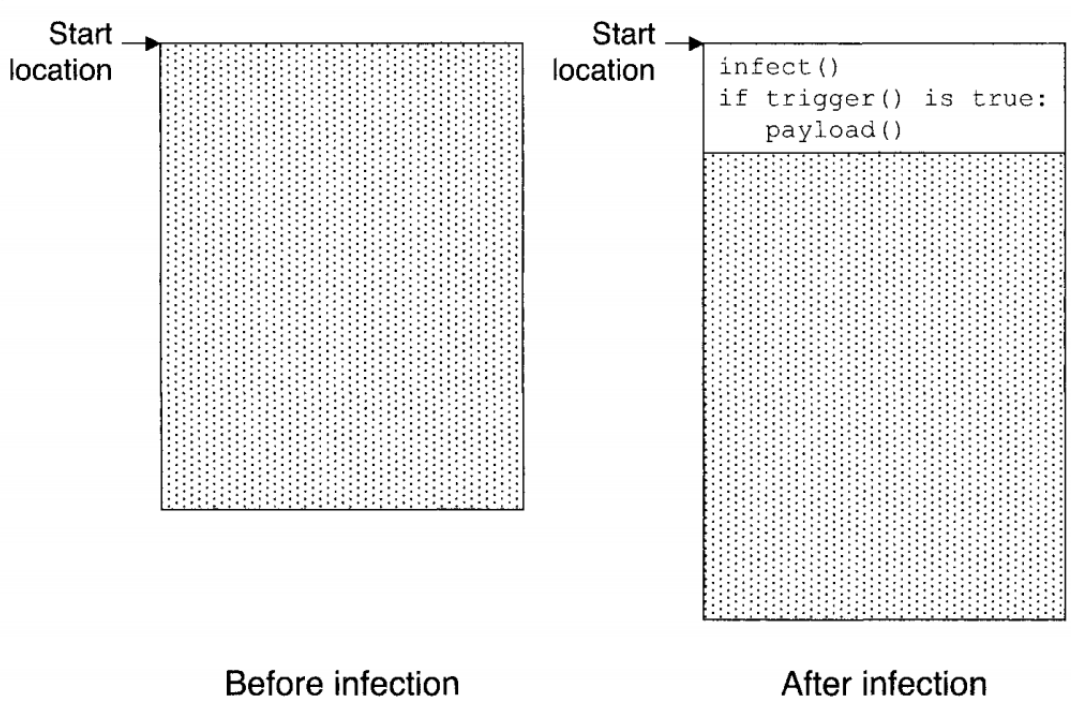


Figure1. Prepending Virus

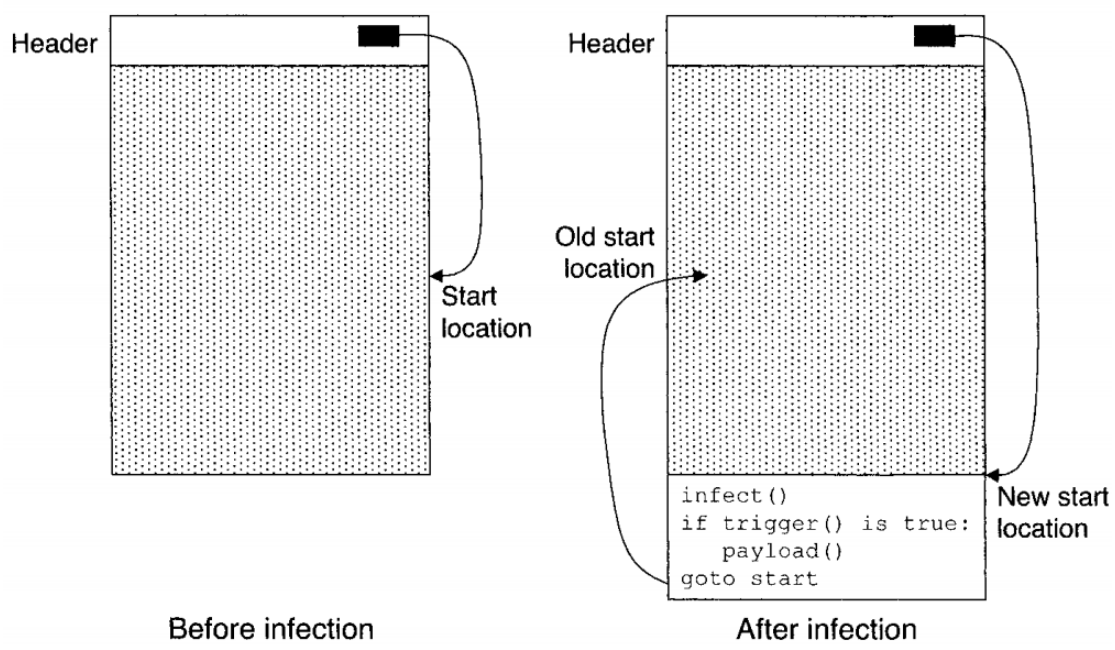


Figure2. Appending Virus

• Many executable file formats specify the start location in a file header. The virus can change this start location to point to its own code, then jump to the original start location when done. Figure 2 shows an appending virus using the latter scheme.

Well known Computer Viruses:

The following are some well-known viruses:

CodeRed

It is a worm that infects a computer running Microsoft IIS server. This virus launched DOS attack on White House's website. It allows the hacker to access the infected computer remotely.

Nimba

It is spreads itself using different methods. IT damages computer in different ways. It modified files, alters security settings and degrades performance.

SirCam

It is distributed as an email attachment. It may delete files, degrade performance and send the files to anyone.

Melisa

It is a virus that is distributed as an email attachment. IT disables different safeguards in MS Word. It sends itself to 50 people if Microsoft Outlook is installed.

Ripper. It corrupts data from the hard disk.

MDMA. It is transferred from one MS Word file to other if both files are in memory.

Concept

It is also transferred as an email attachment. It saves the file in template directory instead of its original location.

One_Half

It encrypts hard disk so only the virus may read the data. It displays One_Half on the screen when the encryption is half completed.

Anti-virus programs:

The broad definition for anti-virus programs is that they are one or multiple programs that have been designed with the specific purpose of preventing and destroying the malicious software (also known as the computer viruses) they detect on the user's computer. These programs are very smart and they use all sorts of tricks in order to catch and destroy the malicious software on the user's computer. They go as far as creating 'bait files' that the viruses will attack and thus get destroyed or quarantined.

Things to remember when purchasing an anti-virus program:

- **Make sure it's a full paid version** – the free ones just search your computer for patterns of malicious software they are already familiar with and they don't do anything about the viruses they find on your computer. The full and paid version of the anti-virus program will use a variety of techniques to catch, destroy or quarantine the viruses on your computer.
- **Make sure you have an Internet connection** –the anti-virus programs need a constant internet connection in order to stay up to date on the virus definitions and make sure that they catch all of the malicious software / programs that can be found in your computer.
- **Don't download anything that you don't trust to be virus free**. No matter how pretty that wallpaper looks like or how much you want that new song,

unless you know and trust the website, don't ever download anything off of the internet. Oftentimes, those that create such malicious software hide it in plain sight – in .mp3 or .jpg files and once it reaches your computer, it's very difficult to get rid of it.

- **Do some research before purchasing a certain anti-virus program**. As you probably know by now, the competition is fierce in the anti-virus industry, do some research, see what other people experienced with the same program and then decide whether to buy it or not.

How do anti-virus programs work?

The science behind anti-virus programs is not that difficult to understand. As mentioned previously, they were created with the sole purpose of stopping the malicious software from damaging the computers of the users. They employ a variety of techniques in order to detect, quarantine and eventually destroy the viruses, some of which include:

- **Scanning the downloaded files;** This means that the anti-virus program starts to scan the documents you are downloading into your computer as soon as they appear in your computer or as soon as you click the "Download" button on the website you want to download the said file from.
- **Scanning the programs before you execute them;** When you double click a program in order to start it up, even if you don't notice it because it happens so fast, the anti-virus program on your computer will scan it very quick in order to make sure that there is no malicious software attached to it and that you can safely open it. If it is, then the anti-virus program will stop the program from starting up in order to protect your computer.
- **Scanning the entire computer;** If you tell your anti-virus program to scan your entire computer, it will take each file, one by one and run it through a series of programs in order to determine whether the said file contains malware or malicious software. This type of scanning takes much longer than usual obviously because the amount of files the anti-virus must go through is huge.