INFORMATION SECURITY

Introduction

The requirements of information security within an organization have undergone two major changes in the last several decades. Before the widespread use of data processing equipment, the security of information felt to be valuable to an organization was provided primarily by physical and administrative means. An example of the former is the use of rugged filing cabinets with a combination lock for storing sensitive documents.

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.

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

Consider the following examples of security violations:

1. User A transmits a file to user B. The file contains sensitive information (e.g., payroll records) that is to be protected from disclosure. User C, who is not authorized to read the file, is able to monitor the transmission and capture a copy of the file during its transmission.

2. A network manager, D, transmits a message to a computer, E, under its management. The message instructs computer E to update an authorization file to include the identities of a number of new users who are to be given access to that computer. User F intercepts the message, alters its contents to add or delete entries, and then forwards the message to E, which accepts the message as coming from manager D and updates its authorization file accordingly.

3. Rather than intercept a message, user F constructs its own message with the desired entries and transmits that message to E as if it had come from manager D. Computer E accepts the message as coming from manager D and updates its authorization file accordingly.

4. An employee is fired without warning. The personnel manager sends a message to a server system to invalidate the employee's account. When the invalidation is accomplished, the server is to post a notice to the employee's file as confirmation of the action. The employee is able to intercept the message and delay it long enough to make a final access to the server to retrieve sensitive information. The message is then forwarded, the action taken, and the confirmation posted. The employee's action may go unnoticed for some considerable time.

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

Internetwork security is both fascinating and complex. Some of the reasons follow:

1. Security involving communications and networks is not as simple as it might first appear to the novice. The requirements seem to be straightforward; indeed, most of the major requirements for security services can be given self-explanatory one-word labels: confidentiality, authentication, nonrepudiation, integrity. But the mechanisms used to meet those requirements can be quite complex, and understanding them may involve rather subtle reasoning.

2. In developing a particular security mechanism or algorithm, one must always consider potential attacks on those security features. In many cases, successful attacks are designed by looking at the problem in a completely different way, therefore exploiting an unexpected weakness in the mechanism.

3. Because of point 2, the procedures used to provide particular services are often counterintuitive: It is not obvious from the statement of a particular requirement that such elaborate measures are needed. It is only when the various countermeasures are considered that the measures used make sense.

4. Having designed various security mechanisms, it is necessary to decide where to use them. This is true both in terms of physical placement (e.g., at what points in a network are certain security mechanisms needed) and in a logical sense [e.g., at what layer or layers of an architecture such as TCP/IP (Transmission Control Protocol/Internet Protocol) should mechanisms be placed].

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

ATTACKS, SERVICES, AND MECHANISMS

To assess the security needs of an organization effectively and to evaluate and choose various security products and policies, the manager responsible for security needs some systematic way of defining the requirements for security and characterizing the approaches to satisfying those requirements. One approach is to consider three aspects of information security:

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

Security Attacks

Attacks on the security of a computer system or network are best characterized by viewing the function of the computer system as providing information. In general, there is a flow of information from a source, such as a file or a region of main memory, to a destination, such as another file or a user. This normal flow is depicted in figure 1. The remaining parts of the figure show the following four general categories of attack:

- **Interruption:** An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on **availability.** Examples include destruction of a piece of hardware, such as a hard disk, the cutting of a communication line, or the disabling of the file management system.
- **Interception:** All unauthorized party gains access to an asset. this is an attack on confidentiality. The unauthorized party could be a person, a program, or a computer. Examples include wiretapping to capture data in a network, and the illicit copying of files or programs.

- **Modification:** An unauthorized party not only gains access to but tampers with an asset. This is an attack on **integrity.** Examples include changing values in a data file, altering a program so that it performs differently, and modifying the content of messages being transmitted in a network.

- **Fabrication:** An unauthorized party inserts counterfeit objects into the system. This is an attack on **authenticity.** Examples include the insertion of spurious messages in a network or the addition of records to a file.

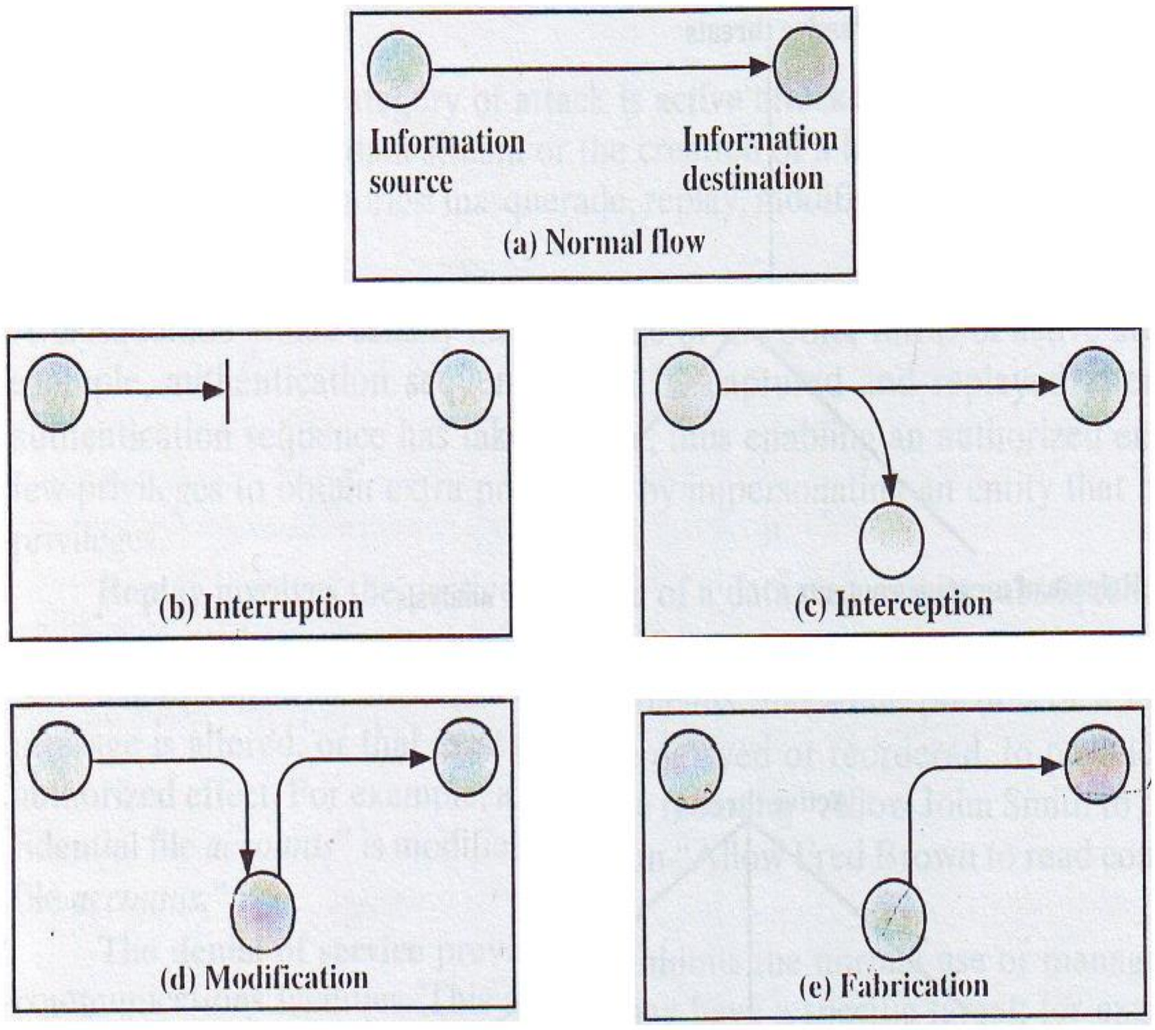


Figure 1:Security Attacks

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 2.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.

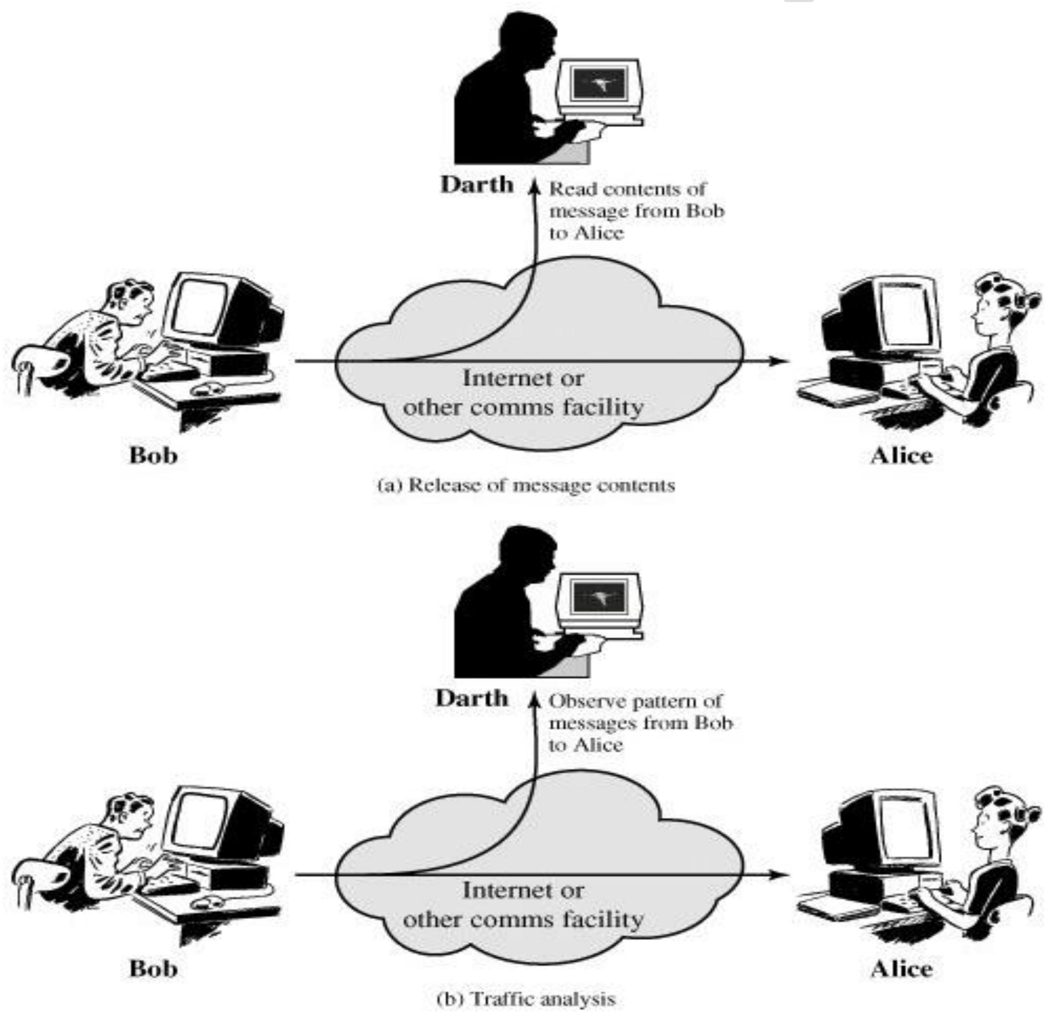

Darth
Read contents of message from Bob to Alice
Internet or other comms facility
Bob
Alice
(a) Release of message contents
Darth
Observe pattern of messages from Bob to Alice
Internet or other comms facility
Bob
Alice
(b) Traffic analysis

Figure 2:Passive Attacks

A second type of passive attack, **traffic analysis**, is subtler (Figure 2.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 3.a). A masquerade attack usually includes one of the other forms of active attack.

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect (Figure 3.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 3.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."

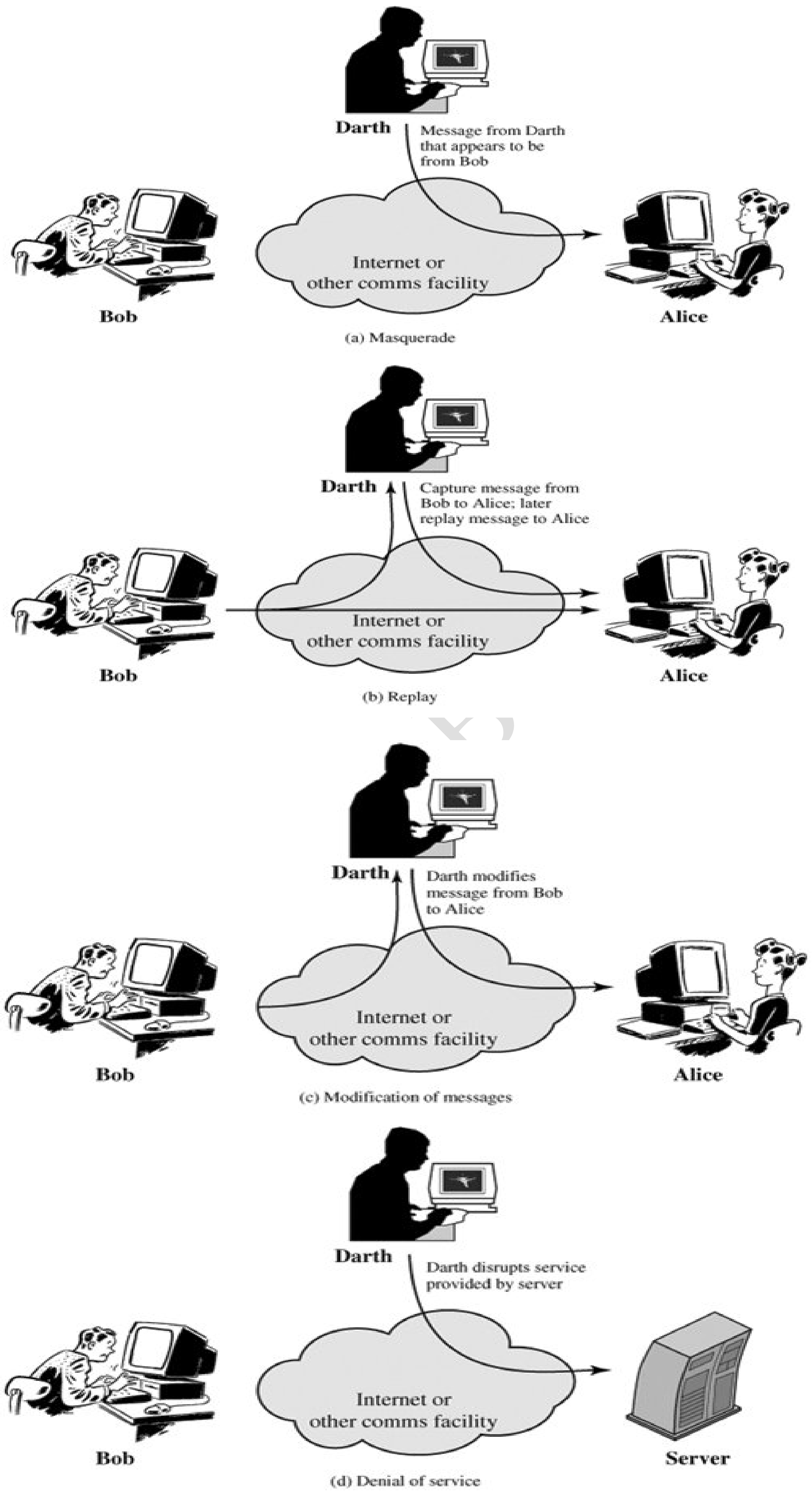

Darth
Message from Darth that appears to be from Bob
Internet or other comms facility
Bob
Alice
(a) Masquerade
Darth
Capture message from Bob to Alice; later replay message to Alice
Internet or other comms facility
Bob
Alice
(b) Replay
Darth
Darth modifies message from Bob to Alice
Internet or other comms facility
Bob
Alice
(c) Modification of messages
Darth
Darth disrupts service provided by server
Internet or other comms facility
Bob
Server
(d) Denial of service

Figure 3:Active Attacks

The **denial of service** prevents or inhibits the normal use or management of communications facilities (Figure 3.d). This attack may have a specific target; for example, the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

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.

Mechanisms

We can note that there is one particular element that underlies most of the security mechanisms in use :cryptographic techniques . Encryption or encryption-like Transformations of information are the most common means of providing security.

Security Services

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 ongoing 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. For example, when a TCP connection is set up between two systems, this broad protection prevents the release of any user data transmitted over the TCP connection. Narrower forms of this service can also be defined, including the protection of a single message or even specific fields within a message. These refinements are less useful than the broad approach and may even be more complex and expensive to implement.

The other aspect of confidentiality is the protection of traffic flow from analysis. This requires that an attacker not be able to observe the source and destination, frequency, length, or other characteristics of the traffic on a communications facility.

Data Integrity:

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

A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. 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.

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

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.

Availability Service

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

Classical Encryption Techniques

Symmetric Cipher Model

A symmetric encryption scheme has five ingredients (Figure 6):

- 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 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.

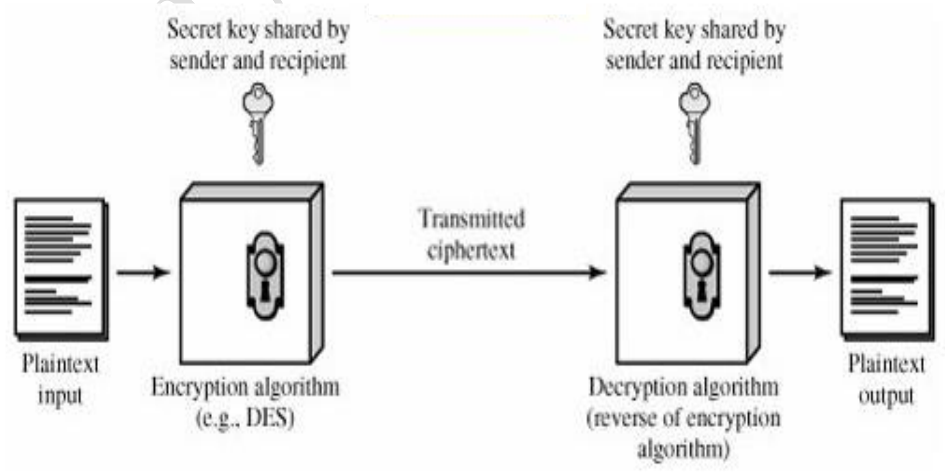

Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Figure 6. Simplified Model of Conventional Encryption

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more 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.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 7. A source produces a message in plaintext, $X = [X_1, X_2, ..., X_M]$. 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 $K = [K_1, K_2, ..., K_J]$ 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.

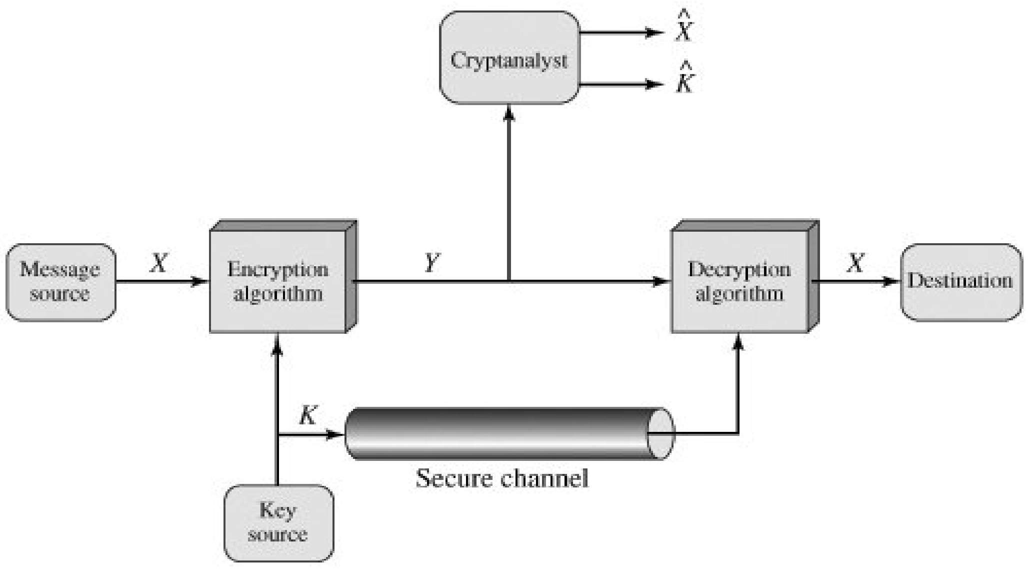


Figure 7. Model of Conventional Cryptosystem

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

Cryptography

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.

Substitution Techniques

The two basic building blocks of all encryption techniques are substitution and transposition. We examine these in the next two sections. Finally, we discuss a system that combines both substitution and transposition.

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

Caesar Cipher

The earliest known use of a substitution cipher, and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing 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 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
```

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
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 p, substitute the ciphertext letter C:

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

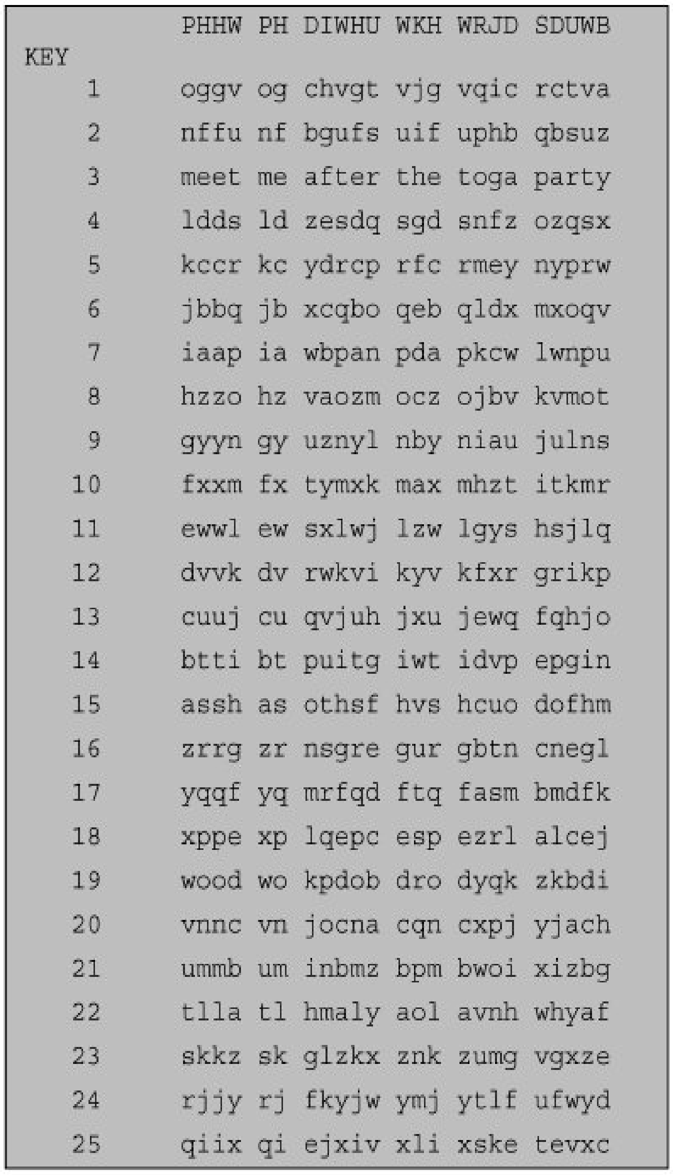
A shift may be of any amount, so that the general Caesar algorithm is

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

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

p = D(k, C) = (C - k) mod 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 8 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.



	PHHW PH DIWHU WKH WRJD SDUWB
KEY	
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 8. Brute-Force Cryptanalysis of Caesar Cipher

Three important characteristics of this problem enabled us to use a brute-force 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.

Monoalphabetic Ciphers

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

```
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 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
```

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! possible keys.

There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language. To see how such a cryptanalysis might proceed, we give a partial example here.The ciphertext to be solved is

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
```

As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 9. If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match. In any case, the relative frequencies of the letters in the ciphertext (in percentages) are as follows:

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

Comparing this breakdown with Figure 9, it seems likely that cipher letters P and Z are the equivalents of plain letters e and t, but it is not certain which is which. The letters S, U, O, M, and H are all of relatively high frequency and probably correspond to plain letters from the set {a, h, i, n, o, r, s}.The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {b, j, k, q, v, x, z}.

There are a number of ways to proceed at this point. We could make some tentative assignments and start to fill in the plaintext to see if it looks like a reasonable "skeleton" of a message. A more systematic approach is to look for other regularities. For example, certain words may be known to be in the text. Or we could look for repeating sequences of cipher letters and try to deduce their plaintext equivalents.

A powerful tool is to look at the frequency of two-letter combinations, known as digrams. A table similar to Figure 9 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our ciphertext, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the ciphertext, and we can translate that sequence as "the." This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track.

Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th_t. If so, S equates with a.

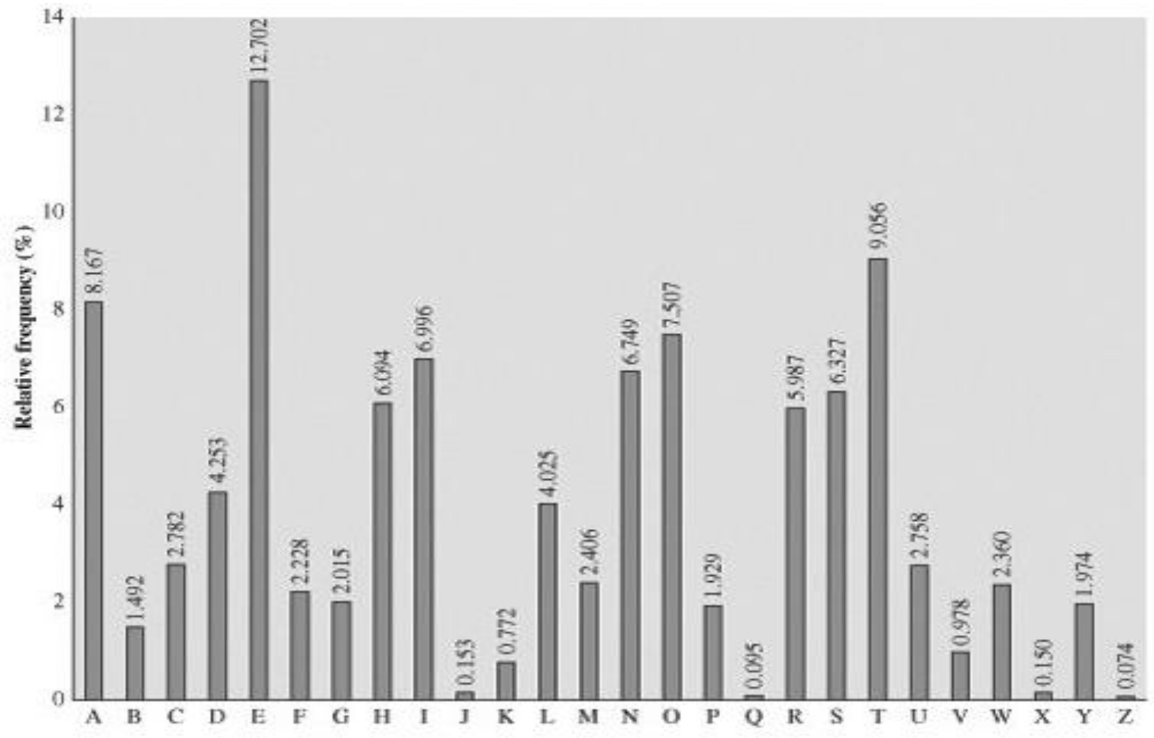


Figure 9. Relative Frequency of Letters in English Text

So far, then, we have

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
 t a        e  e te  a that e e a       a
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
   e t   ta t ha e ee  a e  th    t  a
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
 e  e e tat  e   the   t
```

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

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

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 x 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 witl 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 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, developed by the mathematician Lester Hill in 1929. The encryption algorithm takes m successive plaintext letters and substitutes for them m ciphertext letters. The substitution is determined by m linear equations in which each character is assigned a numerical value (a = 0, b = 1 ... z = 25). For m = 3, the system can be described as follows:

$c_1 = (k_{11P1} + k_{12P2} + k_{13P3})$ mod 26

$c_2 = (k_{21P1} + k_{22P2} + k_{23P3})$ mod 26

$c_3 = (k_{31P1} + k_{32P2} + k3_{3P3})$ mod 26

This can be expressed in term of column 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

C = KP mod 26

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

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

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

The first three letters of the plaintext are represented by the vector

$$\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}. \text{Then } \mathbf{K}\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix} = \text{LNS. Continuing in this fashion,}$$

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 $K K^{-1} = K^{-1}K = I$, 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:

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

It is easily seen that if the matrix K^{-1} is applied to the ciphertext, then the plaintext is recovered. To explain how the inverse of a matrix is determined, we make an exceedingly brief excursion into linear algebra. For any square matrix (m x m) the determinant equals the sum of all the products that can be formed by taking exactly one element from each row and exactly one element from each column, with certain of the product terms preceded by a minus sign. For a 2 x 2 matrix

$$\begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

the determinant is $k_{11}k_{22}$-$k_{12}k_{21}$. For a 3 x 3 matrix, the value of the determinant is $k_{11}k_{22}k_{33}$ + $k_{21}k_{32}k_{13}$ + $k_{31}k_{12}k_{23}$-$k_{31}k_{22}k_{13}$- $k_{21}k_{12}k_{33}$-$k_{11}k_{32}k_{23}$.

In general terms, the Hill system can be expressed as follows:

$C = E(K, P) = KP \bmod 26$

$P = D(K, P) = K^{-1}C \bmod 26 = K^{-1}KP = P$

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger matrix hides more frequency information. Thus a 3 x 3 Hill cipher hides not only single-letter but also two-letter frequency information.

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.

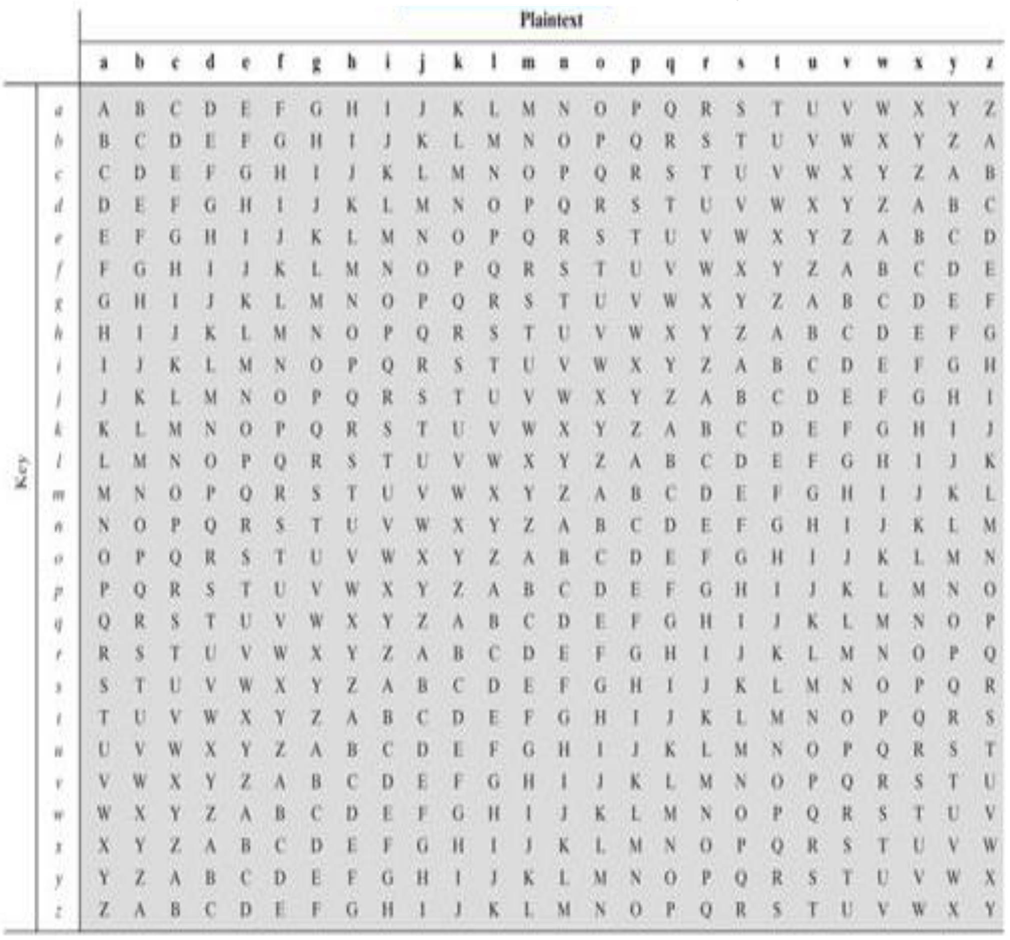
The best known, and one of the simplest, such algorithm is referred to as the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers, with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that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

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

```
key:              deceptivedeceptivedeceptive
plaintext:        wearediscoveredsaveyourself
ciphertext:       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.



		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

Table 1. The Modern Vigenère Tableau

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 reencrypted 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
```

Public-Key Cryptosystems

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

A public-key encryption scheme has six ingredients (Figure 13):

- Plaintext: This is the readable message or data that is fed into the algorithm as input.
- Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.
- Public and private keys: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

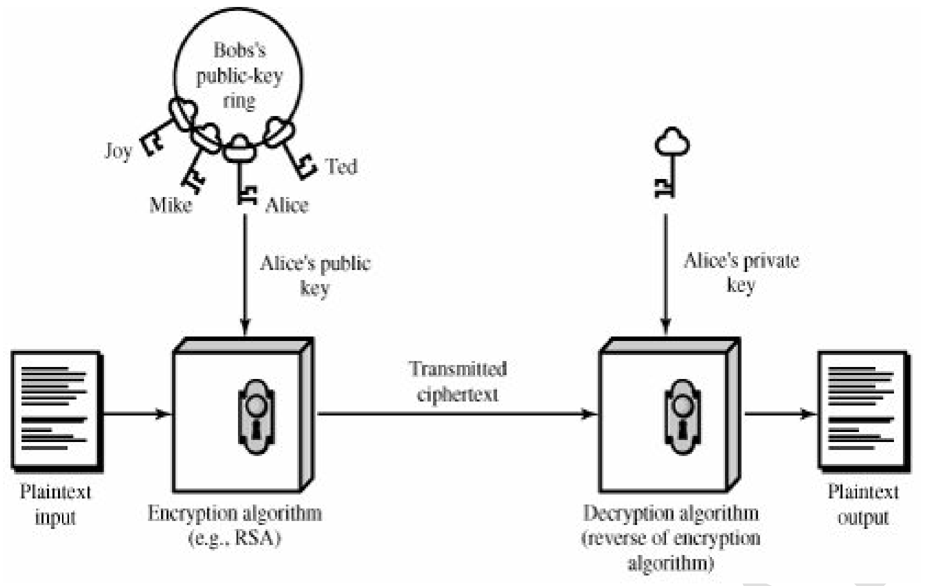


Figure 13. Public-Key Cryptography

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 13 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.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 14. There is some source A that produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M,]$. The M elements of X are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, PU_b, and a private key, PR_b. PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.

With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$:

$Y = E(PU_b, X)$

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

$X = D(PR_b, Y)$

An adversary, observing Y and having access to PU_b but not having access to PR_b or X, must attempt to recover X and/or PR_b. 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 $\hat{X}$ Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover PR_b by generating an estimate $\hat{PR_b}$.

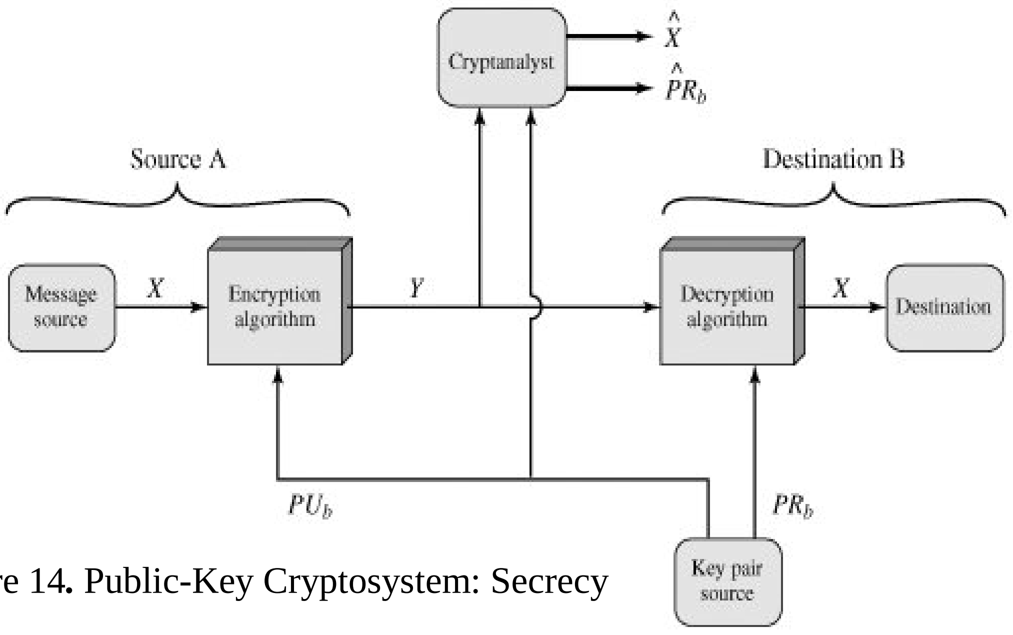


Figure 14. Public-Key Cryptosystem: Secrecy

The RSA 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. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$C = M^e \bmod n$

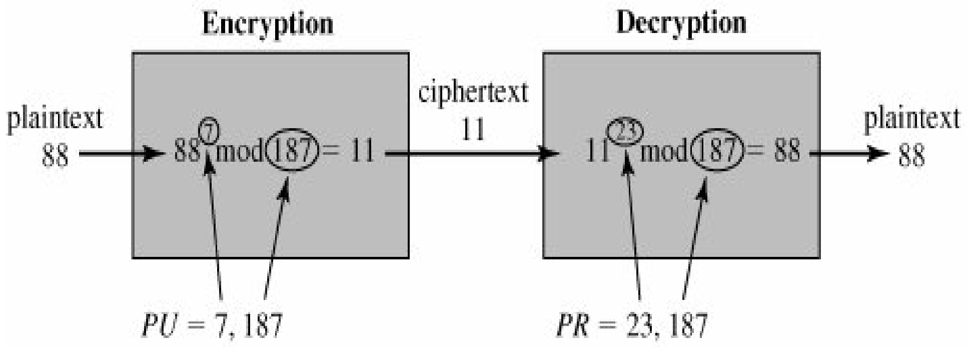
$M = C^d \bmod n$

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 PR = {d, n}. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

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

An example, is shown below. For this example, the keys were generated as follows:

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



Block Cipher Principles

Stream Ciphers and Block Ciphers

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. 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.

Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based symmetric cryptographic applications make use of block ciphers.

The Data Encryption Standard

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.

Figure 15. The RSA Algorithm

The overall scheme for DES encryption is illustrated in Figure 10. 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.

L ocessing
o Figure 16. Example of RSA Algorithm t passes
th duce 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

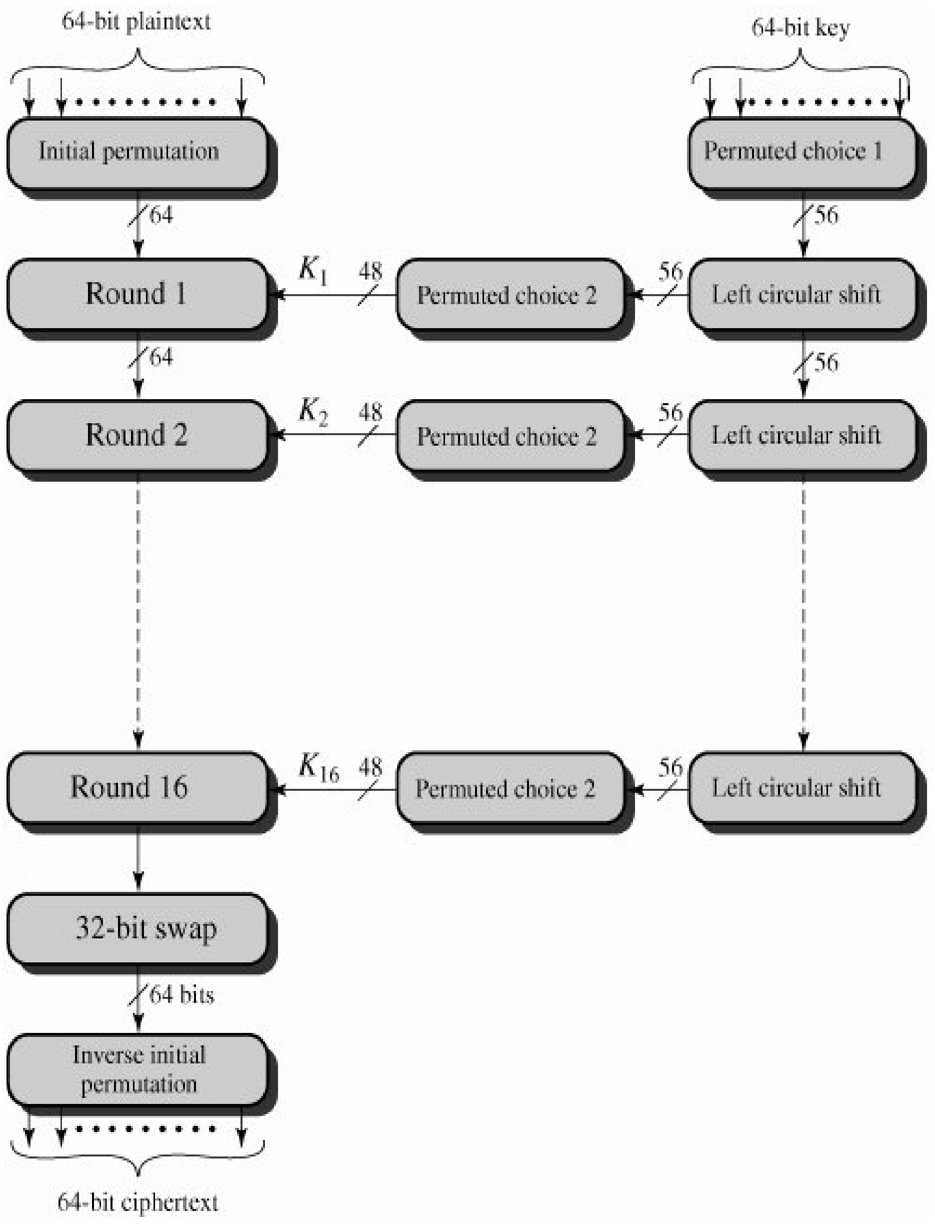


Figure 10: General Depiction of DES Encryption Algorithm

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.

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.

The right-hand portion of Figure 10 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* (K_i) 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 2a and 2b, 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 2. Permutation Tables for DES

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

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

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 M_i is a binary digit. Then the permutation X = 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 = $IP^{-1}(X) = IP^{-1}(IP(M))$, it can be seen that the original ordering of the bits is restored.

Details of Single Round

Figure 11 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). The overall processing at each round can be summarized in the following formulas:

$L_i = R_{i-1}$

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

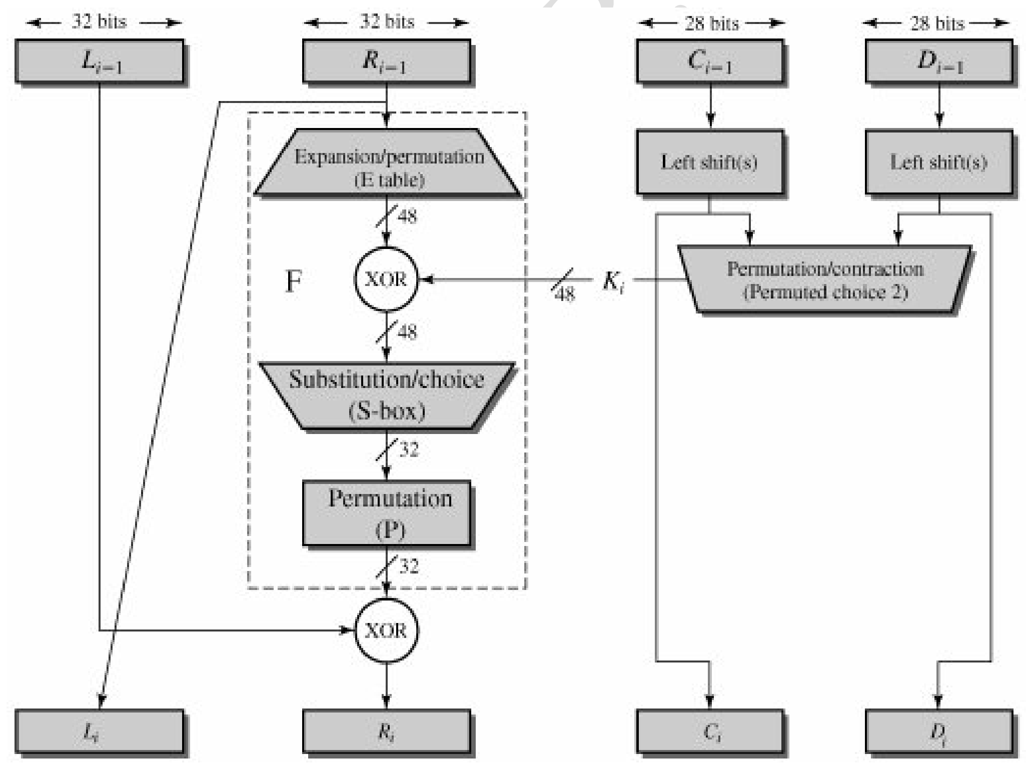


Figure 11. Single Round of DES Algorithm

The round key K_i 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 (Table 2c). The resulting 48 bits are XORed with K_i. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 2d.

The role of the S-boxes in the function F is illustrated in Figure 12. 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, 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 S_1 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.

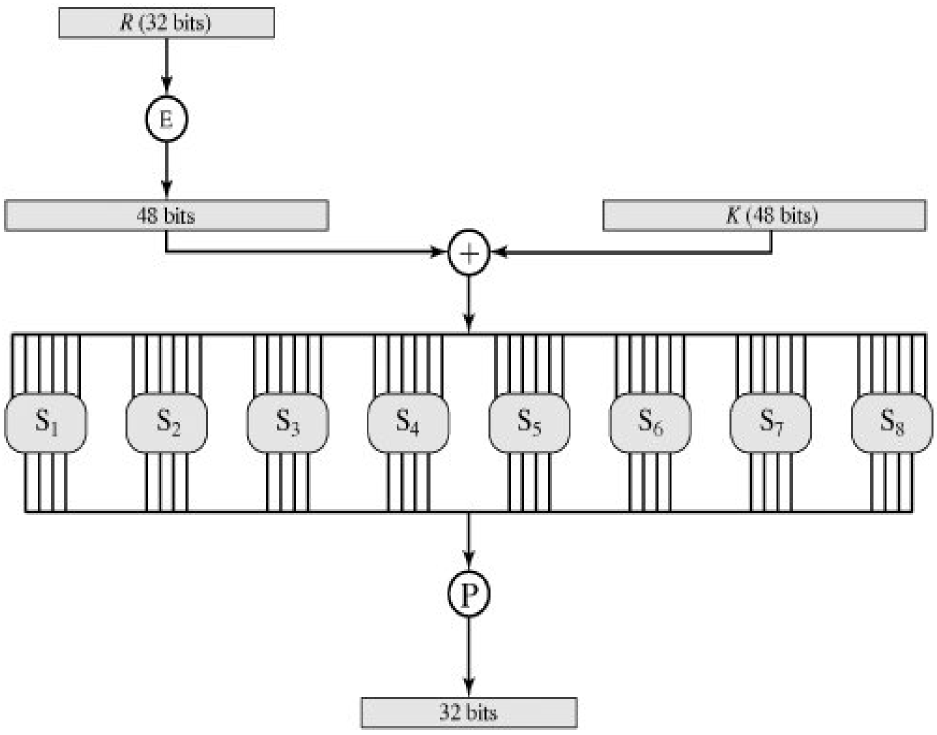
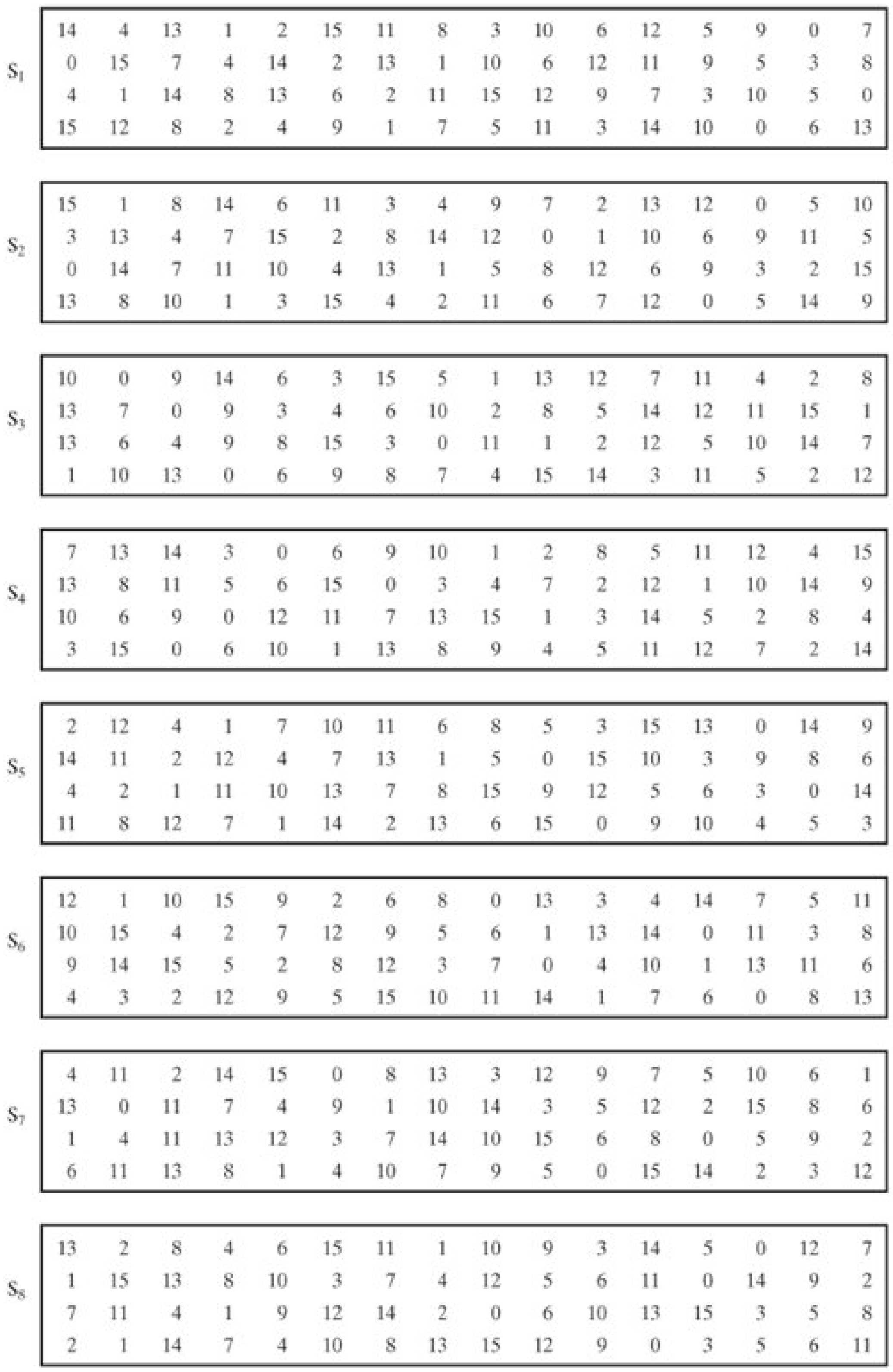


Figure 12. Calculation of F(R, K)

Table 3. 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

Each row of an S-box defines a general reversible substitution.The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (K_i). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... efgh ijkl mnop ...

this becomes

... defghi hijklm lmnopq ...

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.Table 3. Definition of DES S-Boxes

Key Generation

Returning to Figure 10 and Figure 11, 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 4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 4b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 4d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 4c), which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

Table 4. 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

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

The Avalanche Effect

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

DES exhibits a strong avalanche effect. Table 5 shows some results. In Table 5a, two plaintexts that differ by one bit were used:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

with the key

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

Table 5. Avalanche Effect in DES			
(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2

2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

The Table 5a shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions.

Table 5b shows a similar test in which a single plaintext is input:

01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100

with two keys that differ in only one bit position:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

INFORMATION HIDING

Introduction

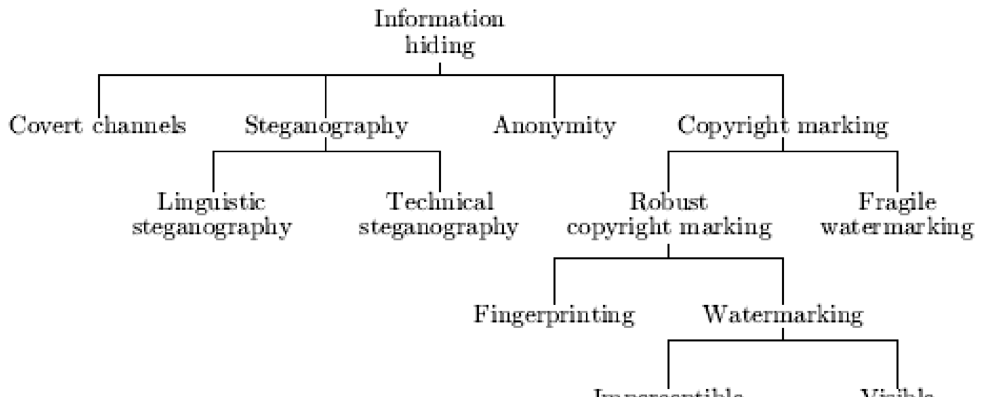
As audio, video, and other works become available in digital form, the ease with which perfect copies can be made, may lead to large-scale unauthorized copying which might undermine the music, film, book, and software publishing industries. These concerns over protecting copyright have triggered significant research to find ways to hide copyright messages and serial numbers into digital media; the idea is that the latter can help to identify copyright violators, and the former to prosecute them.

At the same time, moves by various governments to restrict the availability of encryption services have motivated people to study methods by which private messages can be embedded in seemingly innocuous cover messages. There are a number of other applications driving interest in the subject of information hiding.

Classification of Information Hiding Techniques

Information hiding is the process of embedding a code inside some innocent-looking cover data. Typically, this information is required to be robust against intentional removal by malicious parties. In contrast to cryptography, where the existence but not the meaning of the information is known, information hiding aims to hide entirely the existence of any embedded information.

covert channels (see figure), communication paths that were neither designed nor intended to transfer information at all. *Anonymity* means that the real author of a message is not shown. Anonymity can be im



to make it impossible or very difficult to find out the real author of a message.

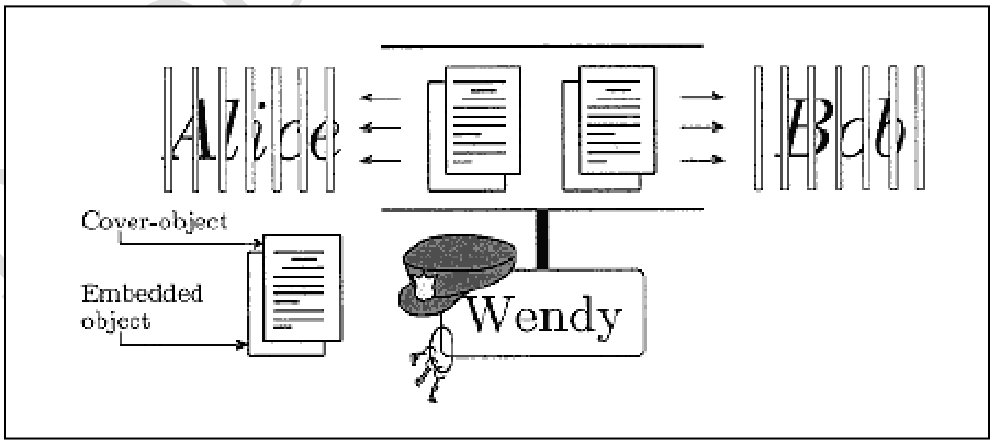
Digital watermarking has recently become a popular research area due to the proliferation of digital data (image, audio, or video) in the Internet age and the need to find a way to protect the copyright of these materials.

The Relationship between watermarking and *steganography* is that both describe techniques that are used to convey information in a hidden manner. However, the underlying philosophy of the two is different. Steganography typically relates to covert point-to-point communication between two parties. Thus, steganographic methods are usually not robust against modification of the data, or have only limited robustness and protect the embedded information against technical modifications that may occur during transmission and storage. Watermarking, on the other hand, is usually a one-to-many communication and the hidden message should be robust to attempts aimed to remove it. Thus, watermarking is used whenever the cover data is available to parties who know the existence of hidden data and may have the interest to alter or remove it. The most popular application of watermarking is copyright protection.

Fingerprinting used to distinguish distributed data sets, is another application of robust copyright marking and has its own special requirements.

Principles of Steganography

The "classic" model for invisible communication was proposed as the "prisoners' problem." Alice and Bob are arrested for some crime and are thrown in two different cells. They want to develop an escape plan, but unfortunately all communications between each other are arbitrated by a warden named Wendy. She will not let them communicate through encryption and if she notices any suspicious communication, she will place them in solitary confinement and thus suppress the exchange of all messages. So both parties must communicate invisibly in order not to arouse Wendy's suspicion; they have to set up a *subliminal channel*. A practical way to do so is to hide meaningful information in some harmless message: Bob could, for instance, create a picture of a blue cow lying on a green meadow and send this piece of modern art to Alice. Wendy has no idea that the colors of the objects in the picture transmit information. we will make the (for an actual prison perhaps unrealistic) assumption that Alice and Bob have access to computer systems in their cells and are able to exchange messages in many different formats (e.g., text, digital images, digital sound, etc.).



Unfortunately there are other problems which may hinder the escape of Alice and Bob. Wendy may alter the message Bob has sent to Alice. For example, she could change the color of Bob's cow to red, and so destroy the information; she then acts as an *active warden.* Even worse, if she acts in a *malicious* way, she could forge messages and send a message to one of the prisoners through the subliminal channel while pretending to be the other.

The above model is generally applicable to many situations in which invisible communication—*steganography*— takes place. Alice and Bob represent two communication parties, wanting to exchange secret information invisibly.

The warden Wendy represents an eavesdropper who is able to read and probably alter messages sent between the communication partners (see Figure above).

Whereas cryptographic techniques try to conceal the contents of a message, steganography goes yet a bit further: it tries to hide the fact that a communication even exists. Two people can communicate covertly by exchanging unclassified messages containing confidential information. Both parties have to take the presence of a *passive, active* or even *malicious* attacker into account.

Frameworks for Secret Communication

Most applications of steganography follow one general principle, illustrated in Figure. Alice, who wants to share a secret message *m* with Bob, randomly chooses (using the private random source *r)* a harmless message *c*, called *cover-object,* which can be transmitted to Bob without raising suspicion, and embeds the secret message into *c,* probably by using a key *k*, called *stego-key.* Alice therefore changes the cover *c* to a *stego-object s.* This must be done in a very careful way, so that a third party, knowing only the apparently harmless message *s*, cannot detect the existence of the secret. In a "perfect" system, a normal cover should not be distinguishable from a stego-object, neither by a human nor by a computer looking for statistical pattern. Theoretically, covers could be any computer-readable data such as image files, digital sound, or written text.

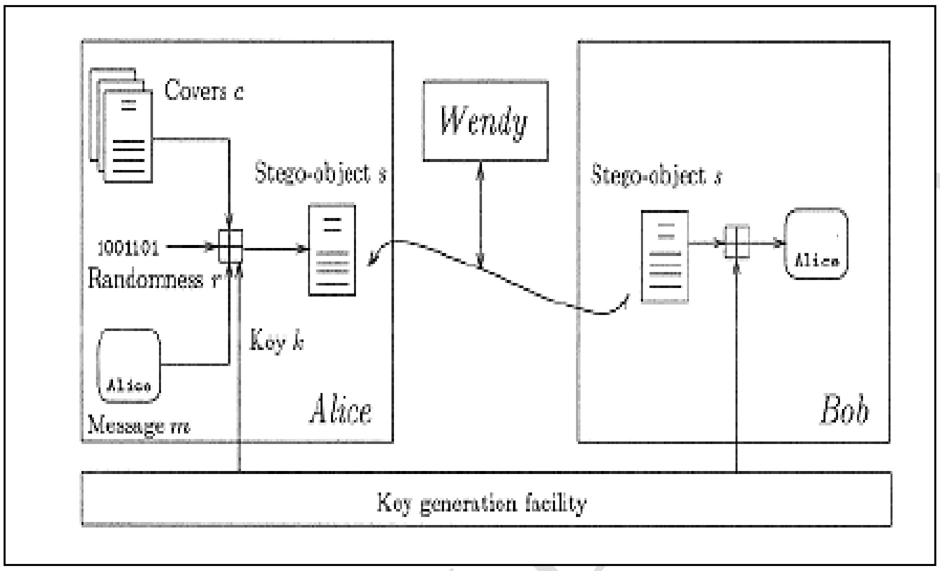
Alice then transmits *s* over an insecure channel to Bob and hopes that Wendy will not notice the embedded message. Bob can reconstruct *m* since he knows the embedding method used by Alice and has access to the key *k* used in the embedding process. This extraction process should be possible *without* the original cover *c*.

A third person watching the communication should not be able to decide whether the sender is *active* in the sense that he sends covers containing secret messages rather than covers without additional information. More formally, if an observer has access to a set {*c*1, . . . , *cn}* of cover-objects transmitted between both communication parties, he should be unable to decide which cover-objects *ci* contain secret information. Thus, the security of invisible communication lies mainly in the inability to distinguish cover-objects from stego-objects.

In practice however, not all data can be used as cover for secret communication, since the modifications employed in the embedding process should not be visible to anyone not involved in the communication process. This fact requires the cover to contain sufficient redundant data, which can be replaced by secret information.

Obviously a cover should never be used twice, since an attacker who has access to two "versions" of one cover can easily detect and possibly reconstruct the message. To avoid accidental reuse, both sender and receiver should destroy all covers they have already used for information transfer.

In the literature there are basically three types of steganographic protocols: *pure steganography, secret keysteganography,* and *public key* *steganography;* the latter is based on principles of public key cryptography. In the following subsections, all three types will be discussed.



Pure Steganography

We call a steganographic system which does not require the prior exchange of some secret information (like a stego-key) *pure steganography*. Both sender and receiver must have access to the embedding and extraction algorithm, but the algorithms should not be public.

Secret Key Steganography

With pure steganography, no information (apart from the functions *E* and *D)* is required to start the communication process; the security of the system thus depends entirely on its secrecy. This is not very secure in practice. So we must assume that Wendy knows the algorithm Alice and Bob use for information transfer. In theory, she is able to extract information out of every cover sent between Alice and Bob. The security of a steganographic system should thus rely on some secret information traded by Alice and Bob, the *stego-key*. Without knowledge of this key, nobody should be able to extract secret information out of the cover.

A secret key steganography system is similar to a symmetric cipher: the sender chooses a cover *c* and embeds the secret message into *c* using a secret key *k*. If the key used in the embedding process is known to the receiver, he can reverse the process and extract the secret message. Anyone who does not know the secret key should not be able to obtain evidence of the encoded information. Again, the cover *c* and the stego-object can be perceptually similar.

Public Key Steganography

As in public key cryptography, public key steganography does not rely on the exchange of a secret key. Public key steganography systems require the use of two keys, one private and one public key; the public key is stored in a public database. Whereas the public key is used in the embedding process, the secret key is used to reconstruct the secret message.

Active and Malicious Attackers

During the design of a steganographic system special attention has to be paid to the presence of active and malicious attackers. Active attackers are able to change a cover during the communication process; Wendy could capture one stego-object sent from Alice to Bob, modify it and forward the result to Bob. It is a general assumption that an active attacker is not able to change the cover and its semantics entirely, but only make minor changes so that the original and the modified cover-object stay perceptually or semantically similar. An attacker is malicious if he forges messages or starts steganography protocols under the name of one communication partner.

Active Attackers: Robust Steganography

Steganographic systems are extremely sensitive to cover modifications, such as image processing techniques (like smoothing, filtering, and image transformations) in the case of digital images and filtering in the case of digital sound. But even a *lossy compression* can result in total information loss. Lossy compression techniques try to reduce the amount of information by removing imperceptible signal components and so often remove the secret information which has previously been added.

An active attacker, who is not able to extract or prove the existence of a secret message, thus can simply add random noise to the transmitted cover and so try to destroy the information. In the case of digital images, an attacker could also apply image processing techniques or convert the image to another file format. All of these techniques can be harmful to the secret communication. Another practical requirement for a steganography system therefore is *robustness.* A system is called robust if the embedded information cannot be altered without making drastic changes to the stego-object.

Malicious Attackers: Secure Steganography

In the presence of a malicious attacker, robustness is not enough. If the embedding method is not dependent on some secret information shared by sender and receiver, (i.e., in the case of pure steganography or public key steganography) an attacker can forge messages, since the recipient is not able to verify the correctness of the sender's identity. Thus, to avoid such an attack, the algorithm must be robust and secure. We can define a *secure steganographic algorithm* in terms of four requirements:

• Messages are hidden using a public algorithm and a secret key; the secret key must identify the sender uniquely;

• Only a holder of the correct key can detect, extract, and prove the existence of the hidden message. Nobody else
should be able to find any statistical evidence of a message's existence;

• Even if the enemy knows (or is able to select) the contents of one hidden message, he should have no chance of detecting others;

• It is computationally infeasible to detect hidden messages.

A Survey of Steganographic Techniques

There are several approaches in classifying steganographic systems. One could categorize them according to the type of covers used for secret communication. A classification according to the cover modifications applied in the embedding process is another possibility. According to the second approach, some of steganographic methods categories are:

• **Substitution systems** substitute redundant parts of a cover with a secret message;

• **Transform domain techniques** embed secret information in a transform space of the signal (e.g., in the frequency domain);

• **Spread spectrum techniques** adopt ideas from spread spectrum communication;

Preliminary Definitions

Throughout the following sections we want to refer to the cover used in the embedding step as *c*. We will further assume (without loss of generality) that any cover can be represented by a sequence of numbers c_i of length $l(c)$ (i.e., $1 \leq i \leq l(c)$). In the case of digital sound this could be just the sequence of samples over time; in the case of a digital image, a sequence can be obtained by vectorizing the image (i.e., by lining up all pixels in a left-to-right and top-to-bottom order). Possible values of c_i are {0,1} in the case of binary images or integers greater than 0 and less than 256 in the case of quantized images or sound. We will denote the stego-object by *s* which is again a sequence *si* of length $l(c)$.

Sometimes we have to index all cover-elements c_i; we will use the symbol *j* for such an index. If the index is itself indexed by some set, we use the notation *ji*. When we refer to the *ji*th cover-element we mean c_{ji}. We will refer to a stegokey as *k;* the structure of *k* will be explained separately in each steganographic application. The secret message will be denoted by *m*, the length of *m* by $l(m)$, and the bits forming *m* by m_i, $1 \leq i \leq l(m)$. Unless otherwise stated, we assume that m_i Є{0, 1}.

A color value is normally a three-component vector in a *color space* (a set of possible colors). A well-known color space is RGB. Since the colors red, green, and blue are *additive primaries,* every color can be specified as a weighted sum of a red, green, and a blue component. A vector in RGB space describes the intensities of these components. Another space, known as YCbCr, distinguishes between a luminance (*Y)* and two chrominance (*Cb,Cr)* components. Whereas the *Y* component accounts for the brightness of a color, *Cb* and *Cr* distinguish between the color grades. A color vector in RGB can be converted to YCbCr using the transform:

$$\begin{aligned} Y &= 0.299\,R + 0.587\,G + 0.114\,B \\ Cb &= 0.5 + (B - Y)/2 \\ Cr &= 0.5 + (R - Y)/1.6 \end{aligned}$$

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 a passive attacker.

Least Significant Bit Substitution

These approaches are common in steganography and are relatively easy to apply in image and audio. A surprising amount of information can be hidden with little, if any, perceptible impact to the carriers.

The embedding process consists of choosing a subset $\{j_1, \ldots, j_{l(m)}\}$ of cover-elements and performing the substitution operation *cji* ↔ *mi* on them, which exchanges the LSB of c_{ji} by m_i (m_i can either be 1 or 0). One could also imagine a substitution operation which changes more than one bit of the cover, for instance by storing two message bits in the two least significant bits of one cover-element. In the extraction process, the LSB of the selected cover-elements are extracted and lined up to reconstruct the secret message.

In order to be able to decode the secret message, the receiver must have access to the sequence of element indices used in the embedding process. In the simplest case, the sender uses all cover-elements for information transfer, starting at the first element. Since the secret message will normally have less bits than *l(c),* the embedding process will be finished long before the end of the cover. In this case, the sender can leave all other cover elements unchanged. This can, however, lead to a serious security problem: the first part of the cover will have different statistical properties than the second part, where no modifications have been made. To overcome this problem, enlarge the secret message with random bits so that *l(c)* = *l(m)* in an attempt to create an equal change at the beginning and the end of the cover. The embedding process thus changes far more elements than the transmission of the secret would require. Therefore the probability that an attacker will suspect secret communication increases.

A more sophisticated approach is the use of a pseudorandom number generator to spread the secret message over the cover in a rather random manner.

Image Downgrading

Image downgrading is a special case of a substitution system in which images act both as secret messages and covers. Given a cover-image and a secret image of equal dimensions, the sender exchanges the four least significant bits of the cover's grayscale (or color) 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 secret image. While the degradation of the cover is not visually noticeable in many cases, 4 bits are sufficient to transmit a rough approximation of the secret image.

Cover-Regions and Parity Bits

We will call any nonempty subset of $\{c_1, \ldots, c_{l(c)}\}$ 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

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

In the embedding step, $l(m)$ disjoint cover-regions I_i $(1 \le i \le l(m))$ are selected, each encodes one secret bit *mi* in the parity bit $p(I_i)$. If the parity bit of one cover-region I_i does not match with the secret bit m_i to encode, one LSB of the values in I_i is flipped. This will result in $p(I_i) = m_i$. 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 pseudorandomly using the stego-key as a seed.

Palette-Based Images

In a palette-based image only a subset of colors from a specific color space can be used to colorize the image. Every palette-based image format consists of two parts: a *palette* specifying *N* colors as a list of indexed pairs (*i*, **c***i*), assigning a color vector **c***i* to every index *i*, and the actual image data which assign a palette index to every pixel rather than the color value itself. If only a small number of color values are used throughout the image, this approach greatly reduces the file size. Two of the most popular formats are the graphics interchange format (GIF) and the BMP bitmap format. However, due to the availability of sophisticated compression techniques, their use declines.

Generally, there are two ways to encode information in a palette-based image: either the palette or the image data can be manipulated. The LSB of the color vectors could be used for information transfer.

Alternatively, information can be encoded in the image data. Since neighboring palette color values need not be perceptually similar, the approach of simply changing the LSB of some image data fails. Some steganographic applications therefore sort the palette so that neighboring colors are perceptually similar before they start the embedding process.

Quantization for Embedding

Quantization of digital images can be used for embedding secret information. We briefly review quantization in the context of predictive coding here. In predictive coding, the intensity of each pixel is predicted based on the pixel values in a specific neighborhood; the prediction may be a linear or nonlinear function of the surrounding pixel values. In its simplest form, the difference *ei* between adjacent pixels x_i and x_{i+1} is calculated and fed into a quantizer Q which outputs a discrete approximation Δ_i of the difference signal x_i - x_{i-1} (i.e., $\Delta_i = Q(x_i - x_{i-1})$). Thus, in each quantization step a quantization error is introduced. For highly correlated signals we can expect Δ_i to be close to zero, so an entropy coder—which tries to create a minimum-redundancy code to be transmitted—will be efficient. At the receiver side the difference signal is dequantized and added to the last signal sample in order to construct an estimate for the sequence x_i.

For steganographic purposes the quantization error in a predictive coding scheme can be utilized; specifically, we adjust the difference signal Δ_i so that it transmits additional information. In this scheme, the stego-key consists of a table which assigns a specific bit to every possible value of Δ_i; for instance, the following assignment could be made:

Δ_i	-4	-3	-2	-1	0	1	2	3	4
	0	1	0	1	1	1	0	0	1

In order to store the ithmessage bit in the cover-signal, the quantized difference signal Δ_i is computed. If Δ_i does not match (according to the secret table) with secret bit to be encoded, Δ_i is replaced by the nearest Δ_j where the associated bit equals the secret message bit. At the receiver site, the message is decoded according to the difference signal Δ_i and the stego-key.

Information Hiding in Binary Images

Binary images—like digitized fax data—contain redundancies in the way black and white pixels are distributed.

A binary image is divided into rectangular image blocks *Bi;* let $P_0(B_i)$ be the percentage of black pixels in the image block *Bi* and $P_1(B_i)$ the percentage of white pixels, respectively. Basically, one block embeds a 1, if $P_1(B_i) > 50\%$ and a 0, if $P_0(B_i) > 50\%$. In the embedding process the color of some pixels is changed so that the desired relation holds. Modifications are carried out at those pixels whose neighbors have the opposite color; in sharply contrasted binary images, modifications are carried out at the boundaries of black and white pixels. These rules assure that the modifications are not generally noticeable.

Unused or Reserved Space in Computer Systems

Taking advantage of unused or reserved space to hold covert information provides a means of hiding information without perceptually degrading the carrier. For example: the way operating systems store files typically results in unused space that appears to be allocated to a file. For example, under Windows 95 operating system, drives formatted as FAT16 (MS-DOS compatible) without compression typically use cluster sizes of 32 kilobytes (Kb). This means that the minimum space allocated to a file is 32 Kb. If a file is 1 Kb in size, then an additional 31 Kb is "wasted." This "extra" space can be used to hide information without showing up in the directory. Unused space in file headers of image and audio can also be used to hold "extra" information.

Another method of hiding information in file systems is to create a hidden partition. These partitions are not seen if the system is started normally. However, in many cases, running a disk configuration utility (such as DOS's FDISK) exposes the hidden partition.

Protocols in the OSI network model have characteristics that can be used to hide information. TCP/IP packets used to transport information across the Internet have unused space in the packet headers. The TCP packet header has six unused (reserved) bits and the IP packet header has two reserved bits. Thousands of packets are transmitted with each communication channel, which provides an excellent covert communication channel if unchecked. Methods of message detection and understanding the thresholds of current technology are necessary to uncover such activities.

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 robust to attacks, such as compression, cropping, and some image processing. However, while they are more robust to various kinds of signal processing, they remain imperceptible to the human sensory system.

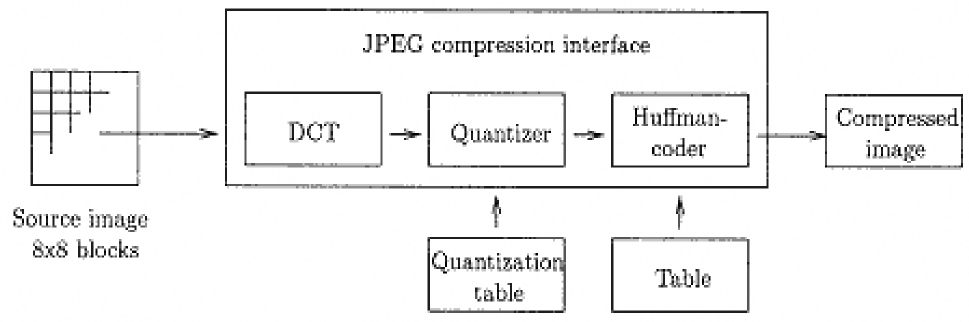
Many transform domain variations exist. One method is to use the discrete cosine transformation (DCT) to embed information in images; another would be the use of wavelet transforms. Transformations can be applied over the entire image, to blocks throughout the image, or other variations.

In digital image processing, the two-dimensional version of the DCT is used:

$$S(u,v) = \frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} s(x,y)\cos\left(\frac{\pi u(2x+1)}{2N}\right)\cos\left(\frac{\pi v(2y+1)}{2N}\right)$$

$$s(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)S(u,v)\cos\left(\frac{\pi u(2x+1)}{2N}\right)\cos\left(\frac{\pi v(2y+1)}{2N}\right)$$

where $C(u) = 1/\sqrt{2}$ if $u = 0$ and $C(u) = 1$ otherwise.
The two-dimensional DCT is the "heart" of the most popular lossy digital image compression system used today: the JPEG system (see Figure). JPEG first converts the image to be compressed into the YCbCr color space and breaks up each color plane into 8×8 blocks of pixels. Then, all blocks are DCT transformed. In a quantization step all DCT coefficients are divided by some predefined quantization values and rounded to the nearest integer (according to a *quality factor*, the quantization values can be scaled by a constant).



Outline of the JPEG image compression algorithm

The purpose of this process is to modulate the influence of the different spectral components on the image.

The resulting quantized DCT coefficients are compressed using an entropy coder (e.g., Huffman or arithmetic coding). In the JPEG decoding step all DCT coefficients are dequantized (i.e., multiplied with the quantization values which had been used in the encoding step). Afterwards an inverse DCT is performed to reconstruct the data. The restored picture will be close to (but not identical with) the original one.

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.

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 $B_i = D\{b_i\}$ 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 (u_1, v_1) and (u_2, v_2). 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.

Algorithm: DCT encoding process

```
for $i = 1,...,l(M)$ do
    choose one cover-block $b_i$
    $B_i = D\{b_i\}$
    if $m_i = 0$ then
        if $B_i(u_1, v_1) > B_i(u_2, v_2)$ then
            swap $B_i(u_1, v_1)$ and $B_i(u_2, v_2)$
        end if
    else
        if $B_i(u_1, v_1) < B_i(u_2, v_2)$ then
            swap $Bi(u_1, v_1)$ and $B_i(u_2, v_2)$
        end if
    end if
    adjust both values so that $|B_i(u_1, v_1) - B_i(u_2, v_2)| > x$
    $b_i' = D^{-1}\{B_i\}$
end for
create stego-image out of all $b_i'$
```

One block encodes a "1," if $B_i(u_1, v_1) > B_i(u_2, v_2)$, 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) affect the relative sizes of the coefficients, the algorithm ensures that $|B_i(u_1, v_1) - Bi(u_2, v_2)| > x$ 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.

Algorithm: DCT decoding process

```
for $i = 1,...,l(M)$ do
    get cover-block $bi$ associated with bit $i$
    $B_i = D\{b_i\}$
    if $B_i(u_1, v_1) \leq B_i(u_2, v_2)$ then
        $m_i = 0$
    else
        $m_i = 1$
    end if
end for
```

Data Hiding in Text

Soft-copy text is in many ways the most difficult place to hide data. (Hard-copy text can be treated as a highly structured image and is readily amenable to a variety of techniques such as slight variations in letter forms, kerning, baseline, etc.) This is due largely to the relative lack of redundant information in a text file as compared with a picture or a sound bite. While it is often possible to make imperceptible modifications to a picture, even an extra letter or period in text may be noticed by a casual reader. Data hiding in text is an exercise in the discovery of modifications that are not noticed by readers. We considered three major methods of encoding data: *open space methods* that encode through manipulation of white space (unused space on the printed page), *syntactic methods* that utilize punctuation, and *semantic methods* that encode using manipulation of the words themselves.

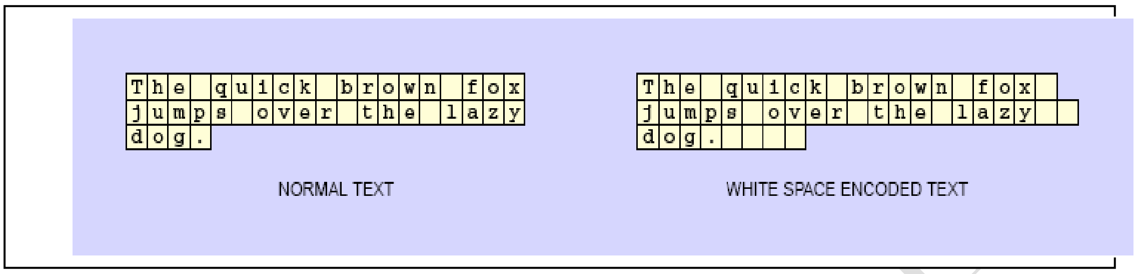
Open space methods

There are two reasons why the manipulation of white space in particular yields useful results. First, changing the number of trailing spaces has little chance of changing the meaning of a phrase or sentence. Second, a casual reader is unlikely to take notice of slight modifications to white space.

We describe three methods of using white space to encode data. The methods exploit *inter-sentence spacing*, *end-of-line spaces*, and *inter-word spacing*.

The first method encodes a binary message into a text by placing either one or two spaces after each terminating character, e.g., a period for English prose, a semicolon for C-code, etc. A single space encodes a “0,” while two spaces encode a “1.” This method has a number of inherent problems. It is inefficient, requiring a great deal of text to encode a very few bits. (One bit per sentence equates to a data rate of approximately one bit per 160 bytes assuming sentences are on average two 80-character lines of text.) Its ability to encode depends on the structure of the text. Many word processors automatically set the number of spaces after periods to one or two characters. Finally, inconsistent use of white space is not transparent.

A second method of exploiting white space to encode data is to insert spaces at the end of lines. The data are encoded allowing for a


The quick brown fox jumps over the lazy dog.
NORMAL TEXT
The quick brown fox jumps over the lazy dog.
WHITE SPACE ENCODED TEXT

predetermined number of spaces at the end of each line (see Figure).

An advantages of this method is that it can be done with any text, and it will go unnoticed by readers, since this additional white space is peripheral to the text. A problem unique to this method is that the hidden data cannot be retrieved from hard copy.

A third method of using white space to encode data involves right-justification of text. Data are encoded by controlling where the extra spaces are placed. One space between words is interpreted as a “0.” Two spaces are interpreted as a “1.” This method results in several bits encoded on each line.

Syntactic methods

That white space is considered arbitrary is both its strength and its weakness where data hiding is concerned. While one may not notice its manipulation, a word processor may inadvertently change the number of spaces, destroying the hidden data. Robustness, in light of document reformatting, is one reason to look for other methods of data hiding in text. In addition, the use of syntactic and semantic methods generally does not interfere with the open space methods. These methods can be applied in parallel.

There are many circumstances where punctuation is ambiguous or when mispunctuation has low impact on the meaning of the text. For example, the phrases “bread, butter, and milk” and “bread, butter and milk” are both considered correct usage of commas in a list.

We can exploit the fact that the choice of form is arbitrary. Alternation between forms can represent binary data, e.g., anytime the first phrase structure (characterized by a comma appearing before the “and”) occurs, a “1” is inferred, and anytime the second phrase structure is found, a “0” is inferred. Other examples include the controlled use of contractions and abbreviations. While written English affords numerous cases for the application of syntactic data hiding, these situations occur infrequently in typical prose. The expected data rate of these methods is on the order of only several bits per kilobyte of text.

Although many of the rules of punctuation are ambiguous or redundant, inconsistent use of punctuation is noticeable to even casual readers. Finally, there are cases where changing the punctuation will impact the clarity, or even meaning, of the text considerably. This method should be used with caution.

Syntactic methods include changing the diction and structure of text without significantly altering meaning or tone. For example, the sentence "Before the night is over, I will have finished" could be stated "I will have finished before the night is over." These methods are more transparent than the punctuation methods, but the opportunity to exploit them is limited.

Semantic methods

A final category of data hiding in text involves changing the words themselves. Semantic methods are similar to the syntactic method.

Rather than encoding binary data by exploiting ambiguity of form, these methods assign two synonyms primary or secondary value. For example, the word "big" could be considered primary and "large" secondary.

Whether a word has primary or secondary value bears no relevance to how often it will be used, but, when decoding, primary words will be read as ones, secondary words as zeros (see Table).

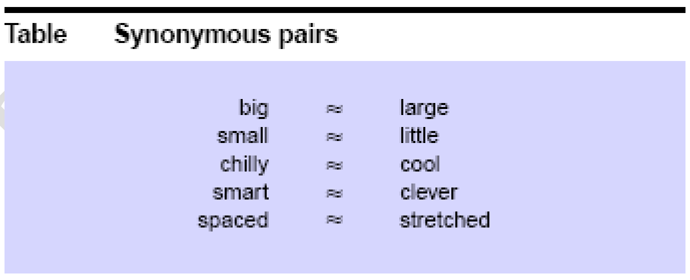


Table Synonymous pairs

big	≈	large
small	≈	little
chilly	≈	cool
smart	≈	clever
spaced	≈	stretched

Problems occur when the nuances of meaning interfere with the desire to encode data. For example, there is a problem with choice of the synonym pair "cool" and "chilly." Calling someone "cool" has very different connotations than calling them "chilly."