CHAPTER TWO

Block Ciphers and Stream Ciphers

A basic problem in cryptography is devising procedures to transform messages (*plaintext*) into cryptograms (*ciphertext*) that can withstand intense cryptanalysis–the techniques used by opponents to penetrate encrypted communications and recover the original information.

The procedures used to accomplish such transformations involve either a *code system* or a *cipher system*. Code systems require a code book or dictionary that translates words, phrases, and sentences of plaintext vocabulary into their equivalent ciphertext code groups. However, the number of plaintext groups that can be converted depends on the size of the code book. Therefore, not every message can be encoded, and the versatility of these code systems is limited.

On the other hand, cipher systems are versatile. They require two basic elements: a *cryptographic algorithm* (a procedure, or a set of rules or steps that are constant in nature); and a set of variable *cryptographic keys*. A key is a relatively short, secret sequence of numbers or characters selected by the user.

After introducing several concepts relevant to ciphers, this chapter discusses two particularly useful ciphers: block ciphers and stream ciphers. Both conventional algorithms (e.g., DES) and public-key algorithms (e.g., the RSA algorithm and the trapdoor knapsack algorithm) are covered under the subject of block ciphers.

Both block and stream ciphers can be used in communications and data processing systems. With a block cipher, data are encrypted and decrypted in blocks, whose length are predetermined by the algorithm's designer. With a stream cipher, the algorithm's user determines the length of data to be encrypted and decrypted. This flexibility requires that stream ciphers, in addition to the algorithm and key, employ another parameter defined as an initializing vector.

Different modes of encryption can be obtained with block and stream ciphers by employing feedback methods (*chaining*), which establish dependencies to past information. Chaining not only strengthens a cipher, but can also be used to authenticate data even when privacy is not required. At the end of the chapter, a comparison is made between block and stream ciphers. Their relative strengths and ease of implementation are discussed.

CRYPTOGRAPHIC ALGORITHMS

The cryptographic algorithm can be thought of as an extremely large number of transformations, the particular transformation in effect depending on the cryptographic key being used. Each transformation changes sequences of intelligible data (plaintext) into sequences of apparently random data (ciphertext). The transformation from plaintext to ciphertext is known as *encipherment* or *encryption.* Each transformation must have a unique inverse operation, also identified by a cryptographic key. The inverse transformation from ciphertext to plaintext is called *decipherment* or *decryption.* (The term that encompasses both enciphering and deciphering operations is *ciphering.*)

There are two types of cryptographic algorithms, *conventional* and *public-key.* With a conventional cryptographic algorithm, the enciphering and deciphering keys are either identical, or, if different, are such that each key can be easily computed from the other. Thus knowledge of the enciphering key is equivalent to knowledge of the deciphering key—when you have one, you also have the other.[1]

A public-key algorithm, on the other hand, permits many users or nodes within a communications system to encipher data using the same public key, but only the specific user or node possessing the secret deciphering key can "unlock" or recover the data. In contrast, a conventional cryptographic algorithm provides effective data security between two users or nodes within a communications system only if these users or nodes have knowledge of the same secret key.[2]

A parameter of a cryptographic algorithm that provides security because of its secrecy is defined as a *cryptographic variable.* The cryptographic key used in a conventional cryptographic algorithm and the private key used in a public-key algorithm are examples of cryptographic variables. They are analogous to the secret combination for a safe.

Enciphering And Deciphering

Consider a representation for the process of enciphering and deciphering with a cryptographic algorithm. (Boldface capital letters are used to define sets, whereas set members are identified by either the corresponding lowercase letters or in some cases the same capital letter not in boldface.) Let **P** represent the collection of all possible plaintext combinations, and **C** the col-

[1] In a conventional cryptographic algorithm it is common to treat the enciphering key and corresponding deciphering key as identical quantities, even though they may differ.

[2] The assumption is made here that the algorithm is known to the opponent and therefore that the strength of the system depends on the key. Moreover, to be useful, the approaches described above must be based on a cryptographic algorithm of validated strength (e.g., DES). The public-key concept is relatively new, and even though several public-key algorithms have recently evolved, their strength has yet to be validated. Therefore, emphasis is given here to encryption schemes based on conventional algorithms such as DES.

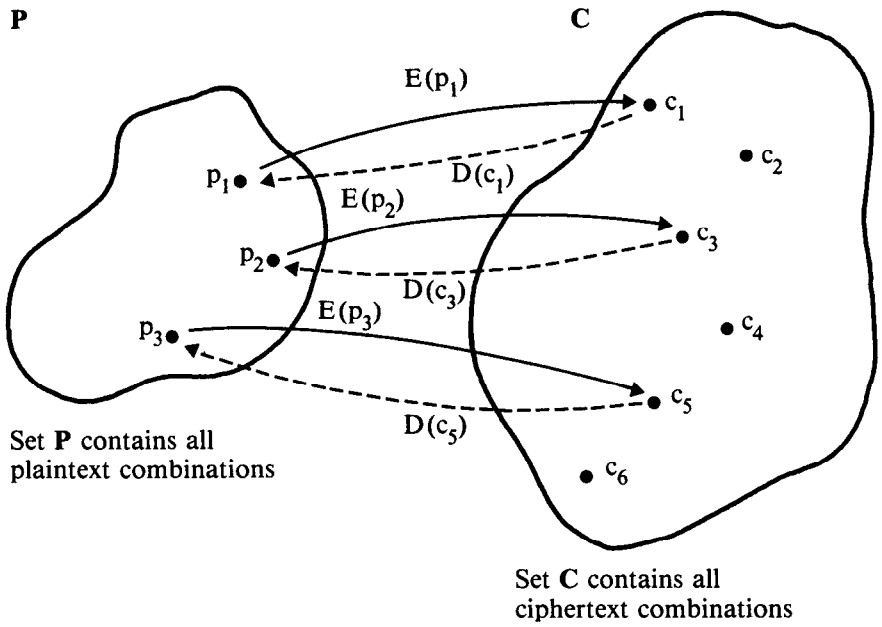
lection of all possible ciphertext combinations. The sets **P** and **C** are described by displaying their members inside braces.

$$\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$$
$$\mathbf{C} = \{c_1, c_2, \ldots, c_m\}$$

The notation $|\mathbf{P}|$ represents the number of elements contained in the set **P**. Hence $|\mathbf{P}| = n$ and $|\mathbf{C}| = m$.

The enciphering process (Figure 2-1) can be described by a rule (E for encipher) that associates with each element p in **P** a single element, $c = E(p)$, in **C**. Each plaintext combination is assigned to a single ciphertext combination.

The deciphering process is described by another rule (D for decipher) that relates each ciphertext element E(p) in **C** with its original plaintext, thus assuring that the plaintext is correctly recovered from the ciphertext. It is assumed here that the number of ciphertext combinations (six in Figure 2-1, i.e., c_1 through c_6) is larger than the number of plaintext combinations



Rule E (encryption process, solid lines) assigns to each element in **P** one element in **C**.

Rule D (decryption process, broken lines) assigns to each of the elements in **C** previously selected by Rule E, one element in **P**, such that the correct plaintext combination is recovered.

Figure 2-1. The Ciphering Process

(three in Figure 2-1, i.e., p_1 through p_3). This situation can be illustrated with a trivial example where plaintext consists of 26 alphabetic characters and ciphertext consists of 26 alphabetic and 10 numeric characters. Thus any one of the 36 ciphertext symbols can be used as a substitute for any one of the 26 plaintext symbols.

The ideas discussed so far can be expressed in mathematical terms by using the concept of a function. A function may also be called a transformation, an operator, or a mapping. This concept can be explained in terms of the ciphering operation illustrated in Figure 2-1. The *function* is defined by the following:

1. A set **P** called the *domain* of the function.
2. A set **C** called the *co-domain* of the function.
3. A *rule* E which associates with each element p of **P** a single element c of **C**.

The function that describes the encipher operation is defined by two sets (**P** and **C**) and a rule which assigns to each element in **P** one element in **C**. Hence the encipher operation can be described by the notation (**P**, **C**, E). It is customary to use the same symbol for the function and its rule. Hence, if (**P**, **C**, E) is a function, then it is said that E is a function from **P** to **C**. This statement can be written as

$$E : \mathbf{P} \rightarrow \mathbf{C}$$

If p is an element in **P** and c the element in **C** that correponds to it under the transformation (function) E, then one writes

$$E(p) = c$$

The set of all E(p) in **P**, also expressed as E(**P**), is defined as the *range* (or *image*) of E. Hence the range E(**P**) is a subset of **C**. In Figure 2-1, the domain of E is **P**, whereas the range of E is the set of elements $\{c_1, c_3, c_5\}$.

There are two properties of functions that need to be distinguished at this point. A function $f : \mathbf{P} \rightarrow \mathbf{C}$ is called *one-to-one* whenever no two different elements in **P** are represented by the same element in **C**; that is, whenever $p_i \neq p_j$ for p_i and p_j in **P** implies that $f(p_i) \neq f(p_j)$. An equivalent statement is that if $f(p_i) = f(p_j)$, then $p_i = p_j$ if f is a one-to-one function. Since plaintext can be recovered correctly only if each ciphertext element represents one and only one plaintext element, all functions representing a cryptographic algorithm must be one-to-one. Otherwise, upon decipherment there would be more than one possible recovered plaintext, thus introducing ambiguity into the decipherment process.

The number of possible one-to-one functions from the set of plaintext elements (**P**) to the set of ciphertext elements (**C**) is determined as follows. The first plaintext element may be transformed to any of $|\mathbf{C}|$ elements, the second plaintext element to $|\mathbf{C}| - 1$ elements, whereas the last plaintext ele-

ment may be mapped to any of $(|\mathbf{C}| - |\mathbf{P}|) + 1$ elements. Therefore, the total number of one-to-one functions is equal to the product of the number of elements in **C** available to each plaintext element, namely

$$|\mathbf{C}| \cdot (|\mathbf{C}| - 1) \cdot \ldots \cdot (|\mathbf{C}| - |\mathbf{P}| + 1) = \frac{|\mathbf{C}|!}{(|\mathbf{C}| - |\mathbf{P}|)!}$$

where

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \qquad \text{(called n factorial)}$$

(Note that $0! = 1$.) In the example shown in Figure 2-1, where $|\mathbf{P}| = 3$ and $|\mathbf{C}| = 6$, there are 120 possible one-to-one functions ($6 \cdot 5 \cdot 4 = 120$), of which only one is shown.

If **S** denotes the set of possible one-to-one functions from **P** to **C**, then there are $|\mathbf{S}|$ such functions, any one of which is a candidate to be used for ciphering. Specifying a cipher key is the same as selecting one of these functions. (How this is achieved in an actual design is explained in Chapter 3). Since the cryptographic key is a cryptographic variable, the symbol v is used to denote a key and the symbol **V** to denote a set of keys. (The symbols **V** and v are used here to avoid conflicts with the symbols K and k, which are used below to denote specific keys.) Since the total number of possible keys is equal to $r = |\mathbf{V}|$, the set **V** can be expressed as

$$\mathbf{V} = \{v_1, v_2, \ldots, v_r\}$$

Let

$$\mathbf{E} = \{E_{v_1}, E_{v_2}, \ldots, E_{v_r}\}$$

specify the corresponding set of functions defining the encipherment procedure, and let

$$\mathbf{D} = \{D_{v_1}, D_{v_2}, \ldots, D_{v_r}\}$$

specify the corresponding set of functions defining the decipherment procedure. Thus the algorithm consists of enciphering (E) and deciphering (D) procedures, where **E** represents the set of all possible enciphering functions (or transformations) and **D** represents the set of all possible deciphering functions.

If the number of keys which can be independently specified exceeds the number of one-to-one functions (i.e., $|\mathbf{V}| > |\mathbf{S}|$) there must be cases where all plaintext-to-ciphertext correspondences are identical even though different keys are used (i.e., $E_{v_i} = E_{v_j}$, even though $v_i \neq v_j$). Such keys are called *equivalent keys*. Even if the number of keys is less than the number of one-to-one functions, $|\mathbf{V}| < |\mathbf{S}|$, equivalent keys may exist. In fact, for highly complex algorithms, it may be too difficult to prove or disprove the existence of equivalent keys.

Nevertheless, a good design principle that reduces the likelihood of equivalent keys is to ensure that the number of possible keys is much less than the number of possible one-to-one functions, (i.e., the condition $|\mathbf{V}| \ll |\mathbf{S}|$ is satisfied). For DES, the following conditions hold.

$|\mathbf{P}| = |\mathbf{C}| = 2^{64}$ (64 binary digits of data are enciphered at a time)

$|\mathbf{V}| = 2^{56}$ (56 binary digits uniquely identify a key)

$|\mathbf{S}| = (2^{64})!$

Since $2^{(64-56)} = 256$, it follows that

$$|\mathbf{S}| = 256 \cdot |\mathbf{V}| \cdot (2^{64} - 1)!$$

and therefore it can be seen that $|\mathbf{V}| \ll |\mathbf{S}|$ for DES.

A function $f : \mathbf{P} \rightarrow \mathbf{C}$ is called *onto* if the range of f is all of **C** (i.e., for any given c in **C** there exists at least one p in **P** such that $f(p) = c$). The function shown in Figure 2-1 is not onto, since some ciphertext combinations (c_2, c_4, and c_6) will not be generated as a result of enciphering all possible plaintext combinations.

It has been established that all functions associated with a cryptographic algorithm must be one-to-one. If they are also onto, then the number of elements in the sets **P** and **C** will be equal (i.e., the number of plaintext combinations is equal to the number of ciphertext combinations). Figure 2-2 shows two such cases, where there are $3 \cdot 2 \cdot 1 = 3!$ functions that are one-to-one and onto. In general, there are $|\mathbf{P}|!$ such functions if $|\mathbf{P}| = |\mathbf{C}|$. In mathematical terms, this implies that each function f_v has an *inverse function,* f_v^{-1}:

$$c = f_v(p)$$

and

$$p = f_v^{-1}(c)$$

Work Factor

To implement a cryptographic algorithm conveniently, the key must be fixed in length, relatively short, and capable of being used repeatedly without weakening security. However, an algorithm that uses a finite key can, theoretically, always be broken (if by no other means than trial and error using every possible key). The only question concerns how much work and resources the opponent must expend. Fortunately, it is not necessary to implement unbreakable algorithms provided that the work (or work factor) required to break the algorithm is sufficiently great to discourage an opponent from attacking it.

Work factor measures what is needed to carry out a specific analysis or attack against a cryptographic algorithm. The attack is conducted under a given set of assumptions which includes the information available to achieve

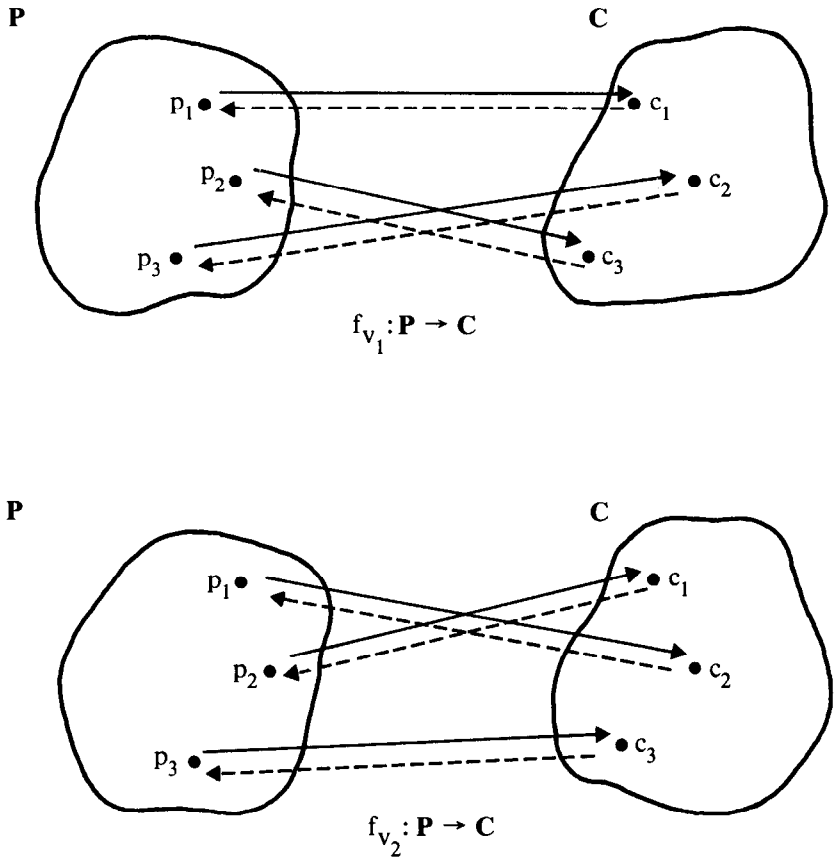


Figure 2-2. Examples of Functions that are One-To-One and Onto

a predetermined goal such as the recovery of the plaintext or key.[3] A good cryptographic algorithm design maximizes the amount of work that an opponent must expend to break it. Thus, for a given algorithm and set of assumptions, the work factor is an expression of the minimum amount of work necessary for a successful attack.

In practice, there is no universally accepted, fixed set of parameters used to express the work factor. However, it is frequently measured in one or more of the following: cryptanalyst hours, number of mathematical or logical operations, computing resources (such as data storage and processing requirements), special hardware, and calendar time. To be useful, the work factor should be expressed using parameters which can be translated for the purpose of comparison into a common base, such as cost in dollars.

[3]With some cryptographic attacks, there may only be a probability of success associated with the recovery of the plaintext or key.

Types of Attacks

In an *exhaustive attack,* an attempt is made to recover the plaintext or key by using direct search methods. For example, in key exhaustion, a known plaintext is enciphered with a trial key and the result is compared for equality with the known corresponding ciphertext.[4] If only ciphertext is available, it can be decrypted with the trial key and the resulting plaintext can be inspected to see if it makes any sense. In this way, it can be determined if the trial key is a candidate for the unknown key or not.

Exhaustive attacks can be thwarted by making the number of required trials very large. However, the work factor of an exhaustive attack, which is directly proportional to the number of trials, is easily determined even when the number of trials is so large that the attack is not feasible. This is not the case with some other attacks.

In an *analytical attack*, a set of mathematical equations (obtained from a definition of the cryptographic algorithm)[5] is solved for the variable or variables representing the unknown message or key. One way to thwart this purely mathematical attack is to construct the algorithm so that each plaintext bit is a sufficiently complex mathematical function of the ciphertext and key, and each key bit is a sufficiently complex mathematical function of the ciphertext and plaintext. If the mathematical equations describing the algorithm's operation are so complex that an analytical attack cannot be successful, then a work factor for this method cannot be calculated. In that case, one usually says that the work factor is very large, implying that the algorithm cannot be broken in the practical sense.

Designing an Algorithm

It is possible to design *unbreakable ciphers* [1]. To do so, the key must be randomly selected (i.e., each key must have the same chance of being chosen) and used only once. Furthermore, the length of the key must be equal to or greater than the length of the plaintext to be enciphered.[6] Unfortunately, long keys of this type, known as *one-time tapes,* are impractical for most applications where there is considerable message traffic, since a large number of keys must be transported and stored before communications can be established.

There are two ways to design a strong cryptographic algorithm [2]. First, one can study the possible methods of solution available to the cryptanalyst —describing them in the most general terms possible—and then define a set of design rules to thwart any one of these methods. An algorithm is then constructed which can resist these general methods of solution. Second, one can construct an algorithm in such a way that breaking it requires the

[4] This attack method assumes that the opponent knows the cryptographic algorithm and possesses a fragment of plaintext and corresponding ciphertext.

[5] The attack assumes that the opponent has knowledge of the cryptographic algorithm.

[6] The cipher is unbreakable because *every* message of the same length is equally likely to have yielded the given ciphertext.

solution of some known problem, but one that is difficult to solve. The DES algorithm was designed using the first approach (Chapter 3), whereas some public-key algorithms have been designed using the second approach (Chapter 2).

Any procedure for attacking a cryptographic algorithm requires that certain cryptographic information (such as ciphertext, plaintext and corresponding ciphertext) be available to carry out the attack. Therefore, the set of procedures that can be used to attack an algorithm depends on the information available to an opponent. Knowing the cryptographic information an opponent might reasonably be able to obtain is thus the basis for determining the class of attacks that the algorithm must be designed to resist.

The cryptographic algorithm, as well as the key, could be kept secret—an approach employed by the military where tight security measures can be enforced. (However, even here, it is ordinarily assumed during threat analyses that attackers have everything except keys and, where applicable, sequencing variables.) In nonmilitary sectors, however, where comparable security measures are impractical or unenforceable, it is unlikely that the secrecy of an algorithm installed at many locations with differing levels of physical security can be maintained for an extended period of time. Moreover, where there are many competing organizations and businesses, a policy of keeping the algorithm secret would promote the widespread use of differing and therefore incompatible algorithms with varying levels of cryptographic strength. An approach that overcomes these difficulties is to adopt a single standard algorithm whose strength has been carefully validated. Such an algorithm would be in the public domain, and its security would depend only on the secrecy of the cryptographic key. This strategy was used by the NBS in adopting the DES algorithm.

Data useful in attacking cryptographic algorithms can be categorized as follows.

1. Ciphertext only.
2. Unselected plaintext and corresponding ciphertext.
3. Selected plaintext and corresponding ciphertext.
4. Selected ciphertext and corresponding plaintext.

Encrypted messages (ciphertext) can be intercepted by wiretapping during transmission; encrypted data files can be copied or stolen from their storage locations (see Attack Scenarios, Chapter 1).

A fragment of plaintext can usually be deduced from some intercepted ciphertext because of the highly formatted text present in most messages and data files. On the other hand, an opponent who could obtain the use of a cryptographic device containing a secret key might (depending on the particular implementation) be able to encipher selected plaintext or decipher selected ciphertext. However, proper physical security and access control procedures are an effective means to prevent unauthorized use of cryptographic devices.

While an opponent's access to certain information (such as ciphertext)

cannot be denied, other information may become known as a result of one or more of the following:

1. A deliberate act that depends on the opponent's skill, daring, and persistence.
2. An unintentional act involving carelessness or ignorance on the part of a cryptographic system's user.
3. An unknown and hence unanticipated event for which no present defense exists.

Except in rare cases, it is impossible to state absolutely that certain information will never become available to an opponent under all operating conditions and environments in which the algorithm may be implemented. Therefore, a conservative approach must be used in algorithm design. It is assumed that the opponent has a wide range of information that might be useful in attacking the algorithm. The algorithm is then designed to resist all known attacks made possible by this information.

Also, it is impossible to state absolutely that an algorithm is free from all possible attacks. Therefore, a conservative approach must likewise be used in the design of a system, such as a communication or file security system, which implements a cryptographic algorithm. It is assumed that the opponent has knowledge of a wide range of attacks that might be capable of breaking the algorithm. The system is then designed to deny the opponent the information needed to carry out the attacks.

In summary, the design of a *strong* cryptographic algorithm must satisfy the following conditions:

1. The mathematical equations describing the algorithm's operation are so complex that, for all practical purposes, it is not possible to solve them using analytical methods.
2. The cost or time required to recover the message or key is too great when using methods that are mathematically less complicated, because either too many computational steps are required (as in the case of message or key exhaustion), or too much data storage is required (as in the case of attacks requiring large accumulations of information such as frequency tables and dictionaries).

Furthermore, it is assumed that the above conditions are satisfied even when the opponent has the following advantages:

1. Relatively large amounts of plaintext (specified by the opponent, if he so desires) and corresponding ciphertext are available.
2. Relatively large amounts of ciphertext (specified by the opponent) and corresponding plaintext are available.
3. All details of the algorithm are available. (It is not assumed that

cryptographic strength depends on maintaining the secrecy of the algorithm.)

4. A number of large high-speed computers (determined by the resources available to the opponent) can be used for cryptanalysis.

The distinction between strong and unbreakable should be apparent. While in theory a strong algorithm can always be broken, in the practical sense it cannot. Unbreakable is an absolute attribute and means that even with an unlimited amount of computational power, data storage, and calendar time, there is no way to obtain the message or key through cryptanalysis. So to speak, strong is a variable, and unbreakable is its maximum value.

Block ciphers and stream ciphers are two fundamentally different approaches which can be used to achieve strong encryption-based protection schemes. The study of these two approaches is thus basic to an understanding, and even a full appreciation, of the direction in which cryptography is currently moving.

Since the main thrust here is to show how cryptography can be used in computer systems, all cryptographic discussion will assume that information is expressed in binary form. The treatment is still general, since any characters can be encoded into binary equivalents.

BLOCK CIPHERS

A *block cipher* (Figure 2-3) transforms a string of input bits of fixed length (an input block) into a string of output bits of fixed length (an output block). The enciphering and deciphering functions are such that every bit in the output block depends jointly on every bit in the input block and on every bit in the key.

A cipher's *blocksize* (the number of bits in a block) is determined by considerations of cryptographic strength, and it must be large enough to

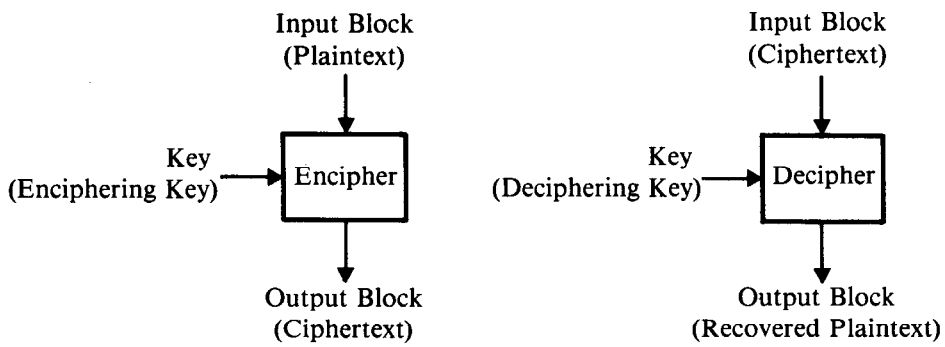


Figure 2-3. Block Cipher

foil simple *message exhaustion* attacks. For example, by enciphering all possible plaintext combinations with a given key, an opponent could build a dictionary of ciphertext (sorted into sequence) and corresponding plaintext. A message could then be recovered by searching the dictionary and relating each intercepted ciphertext block to its corresponding plaintext block. However, if the blocksize is large enough, the dictionary will be too large to construct or store.

In the method of message exhaustion described above, the opponent must be able to encipher data with a key being used by the cryptographic system. In a public-key cryptographic system, the public enciphering-key (or enciphering-transformation) is available to anyone. In a conventional cryptographic system, a conservative assumption is made that the opponent has access to a cryptographic device containing a secret key, even though proper physical security and access control are effective measures against such unauthorized access.[7] (While the opponent can encipher data using the cryptographic device, the key remains unknown.)

Other attacks must also be considered before arriving at an acceptable blocksize. For example, advantage could be taken of the fact that some data blocks are more likely to occur than others. Therefore, the frequency of occurrence should be taken into account. This type of attack is called *block frequency analysis* and uses statistical methods. It is similar to an analysis which could be performed on a simple substitution cipher by taking into account letter frequencies.

By expressing cipher operations in purely mathematical form as a set of equations, it may be possible to solve for the unknown variables directly using analytical methods. This approach is called a *deterministic attack*. To foil deterministic attacks, every bit in the output block must be a sufficiently complex mathematical function of every bit in the input block and key. This property is defined as *strong intersymbol dependence*. From the discussion of work factor, it thus follows that a complex mathematical function must be one for which it is computationally infeasible to solve for the key, even if plaintext and corresponding ciphertext are known (i.e., the work factor is too high).

As part of a mathematical structure for further analysis, several terms useful in a discussion of block ciphers and stream ciphers are defined below.

X: Input (plaintext)
Y: Output (ciphertext)
K: Cryptographic (or cipher) key
Z: Initializing vector (seed value)
U: Intermediate initializing vector
R: Cryptographic bit-stream

[7] Another form of message exhaustion does not require access to the cryptographic device. Instead, each possible plaintext combination is enciphered with each possible key. The opponent then builds a dictionary of plaintext and corresponding ciphertext for each possible key. Later, interception of plaintext and corresponding ciphertext allows the unknown key to be determined.

Since computer data are in binary format, vector notation is used to express such quantities. An input block (X) of b bits is thus denoted by

$$X = (x_1, x_2, \ldots, x_b)$$

where x_i is a 0 or 1 for each i = 1, 2, . . . , b. Using, as before, the notation |*| to represent the number of elements in *, the number of elements in the vector X is denoted |X|. Note that in the example above, the length of X is b, (i.e., |X| = b). In some situations, it is helpful to speak of a sequence or time-sequence of vectors. Here, a sequence of n input blocks is denoted by

$$(X(1), X(2), \ldots, X(n))$$

and specifies the time sequence or relative order of encipherment of each block. If each input block contains b bits, then the vector of input bits at time i is denoted by

$$X(i) = (x_1(i), x_2(i), \ldots, x_b(i))$$

and |X(i)| = b.

In describing a block cipher, it is not necessary to distinguish between the encipherment of block X at time i and the encipherment of the same block X at time j. Simply, encipherment of block X at any time will result in the same block Y. Of course, it is assumed that the same cryptographic key is used. This independence with regard to the order of encipherment does not hold when block chaining is used (a concept discussed later).

Before further details are introduced, a frequently used operation, *modulo 2 addition* or *Exclusive OR* (symbol ⊕), is defined (Table 2-1).

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 2-1. Modulo 2 Addition

From the rules for modulo 2 addition, it follows that

$$A \oplus A = 0$$

$$A \oplus 0 = A$$

$$A \oplus 1 = \overline{A}$$

where $\bar{A}$ is the complement of A. $\bar{A}$ is obtained by inverting the bits in A, that is, 0 becomes 1 and 1 becomes 0. It follows that if

$$A \oplus B = C$$

then

$$A = C \oplus B$$
$$B = C \oplus A$$

(Note that $A \oplus B \oplus B = A \oplus 0 = A = C \oplus B$.)

Let K be a key in the set $\{K1, K2, \ldots, Kn\}$ of possible keys and let f_K be a function in the set $\{f_{K1}, f_{K2}, \ldots, f_{Kn}\}$ of one-to-one functions corresponding to these keys that transforms an input block (X) of b bits into an output block (Y) of b bits, (i.e., $|Y| = |X| = b$). Hence there are 2^b possible plaintext combinations and 2^b possible ciphertext combinations within the domain and co-domain of each function f_K, respectively. In general, only the condition $|Y| \geqslant |X|$ need be satisfied to yield an unambiguous system (a system where no two plaintext combinations map to the same ciphertext combination). For engineering reasons, however, the choice $|Y| = |X|$ is usually made. In that case, the function f_K is one-to-one as well as onto, and hence the inverse function (f_K^{-1}) also exists (Figure 2-2).

Conventional Algorithms

A block-cipher design similar to that used in DES algorithm is now considered. The operations of encipherment and decipherment are described as follows (Figure 2-4).

$$f_K(X) = Y$$

for encipherment, and

$$f_K^{-1}(Y) = X$$

for decipherment. Subscript K designates which particular key (and hence function, f_K) is selected out of the set of all possible keys (and hence functions).

Although f_K must be one-to-one for decipherment to be possible, it is interesting that a one-to-one function f_K can in the most simple case be constructed from a *many-to-one* function (a function that produces the same output for several different inputs). Let such a many-to-one function be defined as g_K. The idea here is to *exercise* g *in the encipherment as well as in the decipherment process.*

To achieve this, the input block (X) consisting of b bits is split into two blocks, L(0) (left) and R(0) (right), each consisting of b/2 bits. Hence X can be expressed as a concatenation of L(0) and R(0):

$$X = L(0),R(0)$$

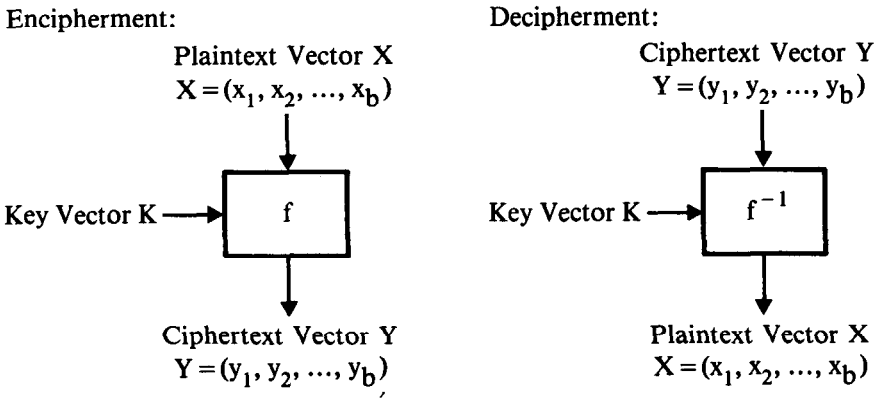


Figure 2-4. Block Cipher (Conventional Cryptographic Algorithm)

g transforms R(0) into $g_K(R(0))$ under control of cipher key K; as indicated in Figure 2-5. L(0) is brought into play by adding it modulo 2 to $g_K(R(0))$ to obtain R(1):

$$R(1) = L(0) \oplus g_K(R(0))$$

The operation is completed by setting L(1) equal to R(0).

If L(1),R(1) represents the ciphertext or scrambled version of L(0),R(0), then the question arises how this ciphertext could be unscrambled without introducing an inverse operation for function g_K. With this goal in mind, the reader should observe that since the ciphertext contains L(1) and since L(1)

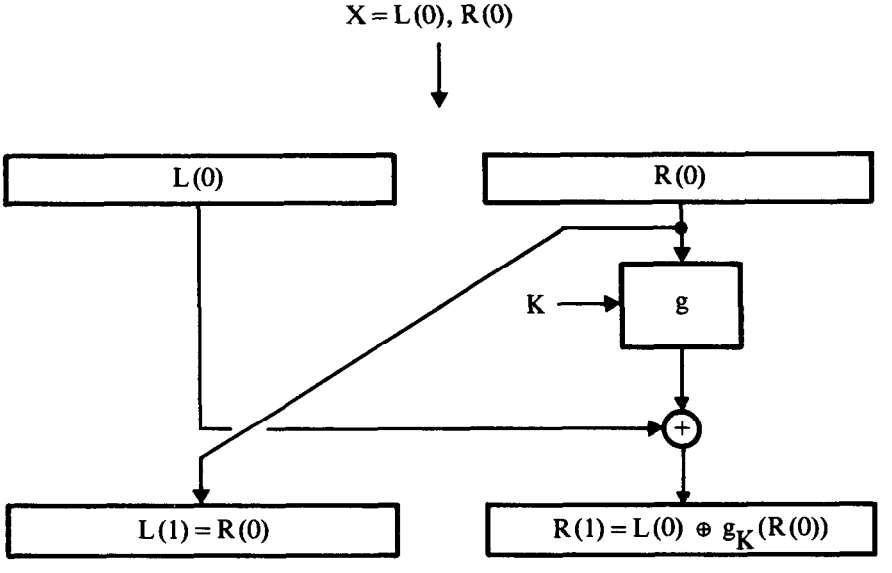


Figure 2-5. Transformation of Input Block (L(0), R(0))

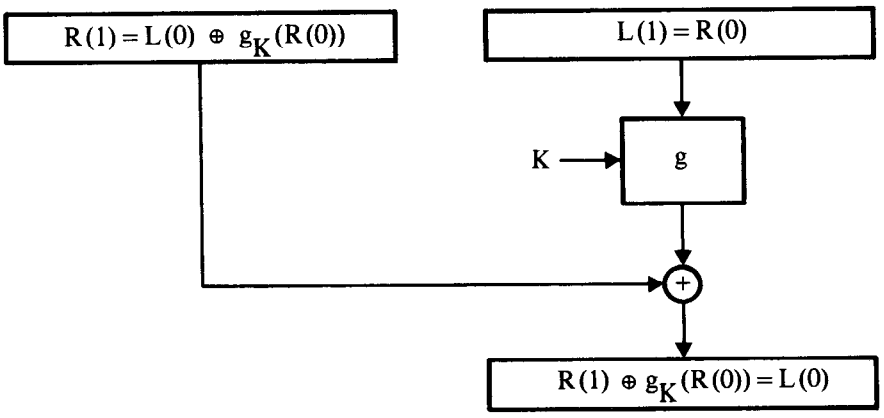


Figure 2-6. Recovery of L(0)

equals R(0), half of the original plaintext is immediately recovered. The remaining half, L(0), can also be recovered, as indicated in Figure 2-6, by recreating $g_K(R(0))$ from R(0) and adding $g_K(R(0))$ modulo 2 to R(1):

$$\begin{aligned} R(1) \oplus g_K(R(0)) &= L(0) \oplus g_K(R(0)) \oplus g_K(R(0)) \\ &= L(0) \end{aligned}$$

However, to use the procedure in Figure 2-5 for encipherment as well as decipherment, the left and right halves of the output are interchanged. That is, the ciphertext (Y) is defined as

$$Y = [L(0) \oplus g_K(R(0))], R(0)$$

This scheme is, of course, extremely weak, since half of the input block, namely R(0), remains unenciphered in the output block. However, cryptographic strength can be obtained by repeating the process (exercising g) n times, where n is called the *number of rounds*, and by using a different key for each round. The basic idea for a two-round system is illustrated in Figure 2-7. The reader should understand that deciphering in such a system is possible only if the internal keys, K(1) and K(2) in Figure 2-7, are exercised in the order K(1), K(2) for encipherment and K(2), K(1) for decipherment. In general, the plaintext can be recovered in an n-round system by exercising the internal keys in the order K(1), K(2), . . . , K(n − 1), K(n) for encipherment and K(n), K(n − 1), . . . , K(2), K(1) for decipherment.

So far it has been assumed that the same key had to be used for encipherment and decipherment, that is,

$$\begin{aligned} f_K(X) &= Y \quad \text{(for encipherment)} \\ f_K^{-1}(Y) &= X \quad \text{(for decipherment)} \end{aligned}$$

where the internal keys are derived from the external key K (the key supplied by the user). However, in the n-round system, the following relations also hold:

$$f_K(X) = Y$$
$$f_{K'}(Y) = X$$

The external keys, K and K′, are defined to have the following schedule of internal keys:

round:	1,	2,	. . . , n − 1,	n
K:	K(1),	K(2),	. . . , K(n − 1),	K(n) (2-1)
K′:	K(n),	K(n − 1),	. . . , K(2),	K(1) (2-2)

Hence, it follows that f_K^{-1} is equivalent to $f_{K'}$.

As discussed earlier, the ciphering process can in general be described by a set of functions, namely

$$\mathbf{E} = \{E_{v_1}, E_{v_2}, \ldots, E_{v_r}\}$$

for encipherment, and

$$\mathbf{D} = \{D_{v_1}, D_{v_2}, \ldots, D_{v_r}\}$$

for decipherment. Selecting a common key (K) for encipherment and decipherment thus determines the enciphering transformation (E_K) as well as the deciphering transformation (D_K). A cryptographic system using the approach shown in Figure 2-7 could, however, be described by defining only one set of functions, that is, by defining only **E**. The set of functions for the deciphering process does not have to be separately specified, since for each key K that is used for enciphering with function (E_K) there is a key K′ that can be used for deciphering with function ($E_{K'}$). In the former case, a common key is used together with the sets of enciphering and deciphering functions (**E** and **D**, respectively) and in the latter case two different keys are used for enciphering and deciphering together with one set of enciphering functions, **E**. It follows, therefore, that

$$E_{K'}(E_K(X)) = E_K(E_{K'}(X)) = X$$

for all possible plaintext (X). And, if K and K′ were to have a much more complex relationship than the one indicated by Equations 2-1 and 2-2, the cryptographic scheme shown in Figure 2-7 could be used as a public-key cryptographic algorithm.

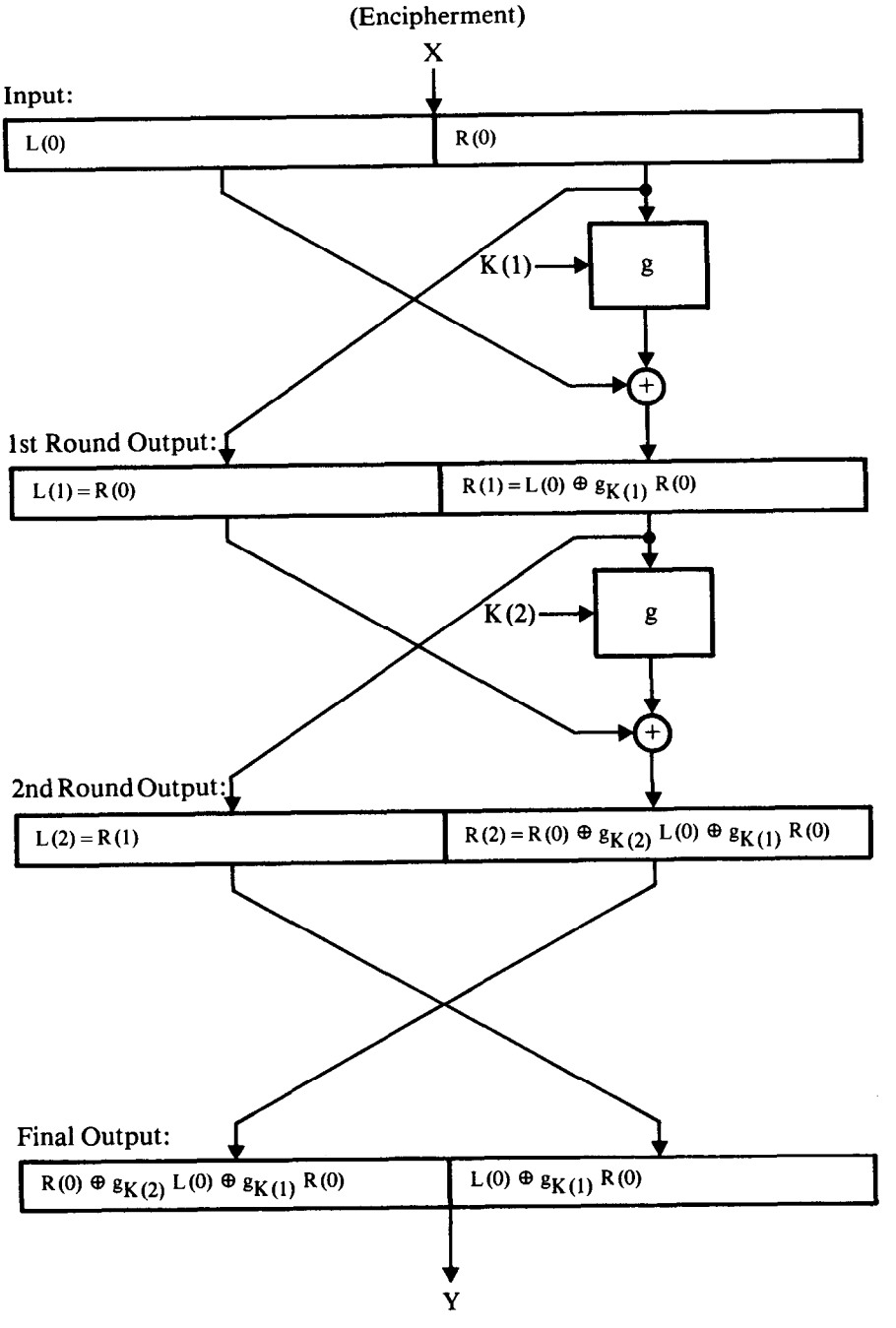


Figure 2-7. A Two-Round Block Cipher

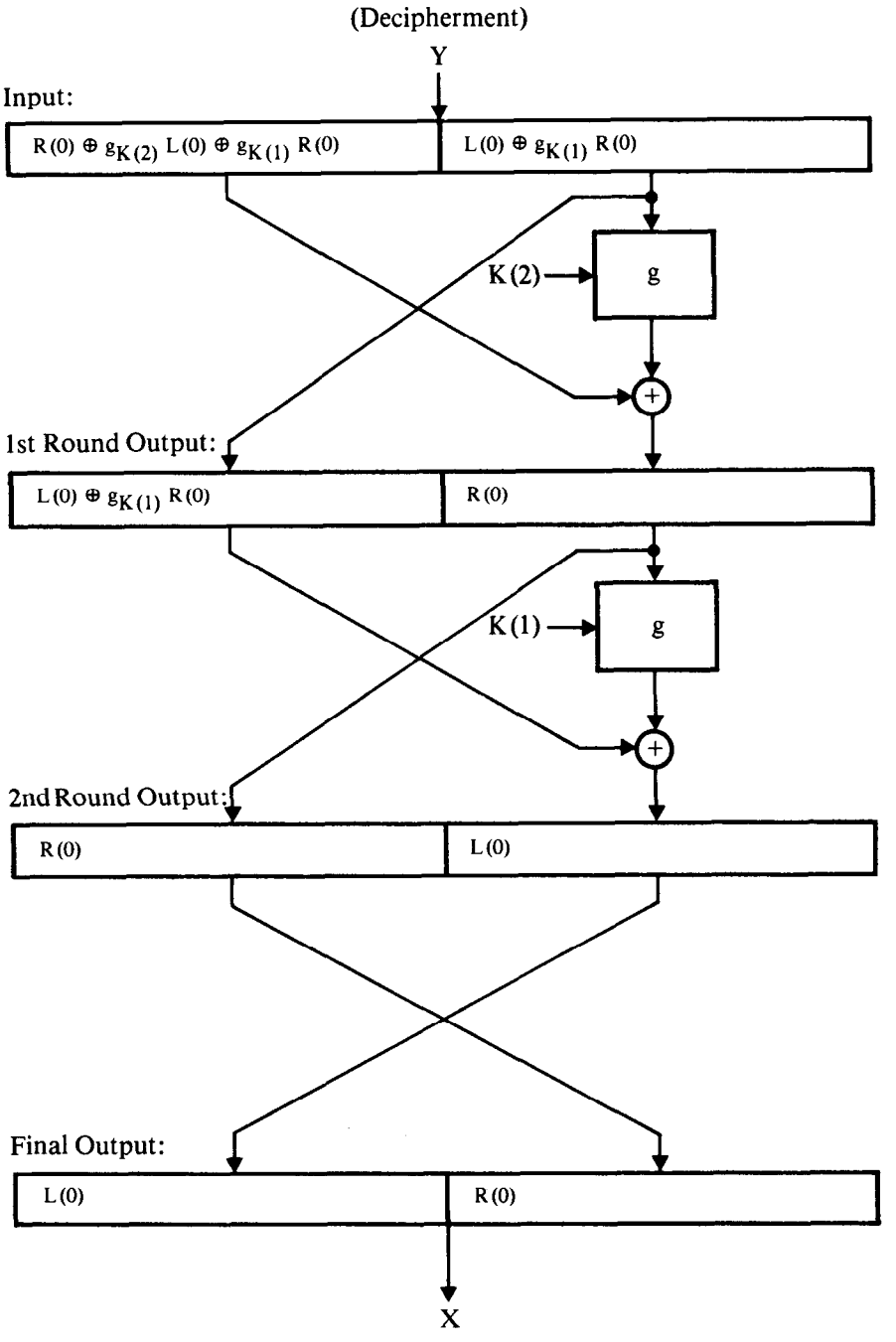


Figure 2-7 (cont'd). A Two-Round Block Cipher

Public-Key Algorithms

Public-key cryptography [3] uses an enciphering key (PK) which is in the public domain and a deciphering key (SK) which is kept secret. Anyone can encipher data using the public key of another user, but only those users with knowledge of the secret key can decipher enciphered data. The enciphering algorithm (E) and the deciphering algorithm (D) might be different, though it is possible for E and D to be identical. (In the discussion that follows, it is assumed that E and D are made public.)

To be used privately, or for private data communications, a public-key algorithm must have the following properties:

1. Users must be able to compute a pair of public and private keys, PK and SK, efficiently.
2. Knowledge of PK must not permit SK to be computed efficiently. (Note: There is no requirement that knowledge of SK prevent PK from being computed efficiently.)
3. Encipherment followed by decipherment causes the original message (X) to be recovered, that is,

$$D_{SK}(E_{PK}(X)) = X$$

for all X in the domain of E_{PK}.

If, in addition to meeting conditions (1) and (2), the public-key algorithm is such that decipherment followed by encipherment causes the original message (X) to be recovered, that is,

$$E_{PK}(D_{SK}(X)) = X$$

for all X in the domain of D_{SK}, then the algorithm can be used to generate a digital signature[8] that authenticates the message's sender (see Chapter 9).

Greater design restrictions are placed on a public-key algorithm than on a conventional algorithm because the public key represents additional information which the opponent can use to attack the algorithm. A public-key algorithm must be designed to withstand attacks made possible by this additional information. (See Cryptographic Strength Considerations.)

In a conventional algorithm, such as DES, the designer has complete freedom to choose the substitutions, permutation, number of rounds, and key schedule (i.e., key bits used in each round) without considering whether the enciphering process reveals the deciphering process. In DES, the deciphering process can be automatically determined if the enciphering key is known, since all steps taken in the enciphering process can easily be retraced to obtain the deciphering process.

On the other hand, in a public-key algorithm it must not be possible to

[8] Merkle and Hellman [4] have shown that digital signatures can be obtained if $E_{PK}(D_{SK}(X)) = X$ holds for only a fraction of the set of possible Xs.

retrace the steps in the enciphering process to determine the deciphering process. Enciphering follows one path and deciphering follows a different path, and knowledge of the former must not reveal the latter.

There are other notable differences between conventional and public-key algorithms. The public-key algorithms invented thus far [4–6] are easily described in mathematical terms, and rely for their strength on the underlying assumption that a particular, known mathematical problem is difficult to solve. On the other hand, a conventional algorithm like DES is designed so that the mathematical equations describing its operation are so complex that for all practical purposes it is not possible to solve them using analytical methods.

Another difference relates to the disciplines needed to attack an algorithm. With a public-key algorithm, these disciplines appear to be few in number and fixed by the algorithm's mathematical description. With a conventional algorithm, on the other hand, the designer has the freedom to ensure that many (possibly chosen) disciplines are required.

Also, the manner in which keys are generated is different for conventional and public-key algorithms. In a conventional algorithm, the key can be randomly selected in a straightforward way, since knowledge of the enciphering key is equivalent to knowledge of the deciphering key, and vice versa. However, in a public-key algorithm, the relationship between the public and private keys is purposely made obscure (i.e., knowledge of the public key does not reveal the private key). Thus, a special procedure is needed to compute the public and private keys, and this procedure must also be computationally efficient.

RSA Algorithm

The RSA algorithm [5] (named for the algorithm's inventors: Rivest, Shamir, and Adleman) is based on the fact that in the current computing art factorization of composite numbers with large prime factors involves overwhelming computations. Indeed, cumulative experience has shown this problem to be intractable [7]. (For more details, see Cryptographic Strength Considerations.)

A number p (p = 1, 2, 3, . . .) is called *prime* if its only divisors are the trivial ones, ±1 and ±p, otherwise it is called *composite*. The primes below 100 are

2	13	31	53	73
3	17	37	59	79
5	19	41	61	83
7	23	43	67	89
11	29	47	71	97

All primes are odd except for the number 2. Every composite number can be factored uniquely into prime factors. For example, 6 is a composite number whose factors are 2 and 3 (i.e., $6 = 2 \cdot 3$). The composite number 999,999

on the other hand is factored by the prime numbers 3, 7, 11, 13, and 37 (i.e., $999{,}999 = 3^3 \cdot 7 \cdot 11 \cdot 13 \cdot 37$).

To describe the RSA algorithm, the following quantities are defined.

1.	p and q are primes	(secret)
2.	$r = p \cdot q$	(nonsecret)
3.	$\phi(r) = (p - 1)(q - 1)$	(secret)
4.	SK is the private key	(secret)
5.	PK is the public key	(nonsecret)
6.	X is the message (plaintext)	(secret)
7.	Y is the ciphertext	(nonsecret)

Because the suggested approach involves modulo arithmetic, congruences are defined in the way they were first introduced by Gauss. Two integers a and b are *congruent* for the modulus m if their difference $a - b$ is divisible by the integer m.[9] This is expressed in the symbolic statement

$$a \equiv b \pmod{m}$$

When a and b are not congruent, they are called *incongruent* for the modulus m, and this is written

$$a \not\equiv b \pmod{m}$$

For any pair of integers a and b, one or the other alternative holds (i.e., a and b are either congruent or incongruent). For example,

$$16 \equiv 1 \pmod{5}$$
$$-7 \equiv 15 \pmod{11}$$
$$-7 \not\equiv 15 \pmod{3}$$

One can state the congruence $a \equiv b \pmod{m}$ slightly differently by saying that b is congruent to a when it differs from a by some multiple (c) of m.

$$b = a + cm$$

The RSA algorithm is based on an extension of Euler's theorem [7], which states that

$$a^{\phi(r)} \equiv 1 \pmod{r}$$

[9] Congruent means agreeing with or corresponding to while modulus (shortened to mod) signifies "little measure."

where

1. a must be relatively prime to r. (Integers a and b are *relatively prime* if their greatest common divisor, gcd, is one.)
2. $\phi(r) = r(1 - 1/p_1)(1 - 1/p_2) \ldots (1 - 1/p_n)$, where $p_1, p_2, \ldots, p_n$ are the prime factors of r.

$\phi(r)$ is Euler's ϕ-function of r (also called indicator or totient) which determines how many of the numbers $1, 2, \ldots, r$ are relatively prime to r.

For example, the composite number $20 = 2^2 \cdot 5$ has two prime factors, 2 and 5. Thus there are $\phi(20) = 20(1 - \frac{1}{2})(1 - \frac{1}{5}) = 8$ integers which are relatively prime to 20 (i.e., which have neither 2 or 5 as a factor):

$$1, 3, 7, 9, 11, 13, 17, 19$$

In the discussion that follows, the reader is expected to be familiar with elementary number theory [7].

To obtain the mathematical relationship between the public and private keys, PK and SK, Euler's result is extended as follows. First, it is shown that $a \equiv b \pmod r$ implies that $a^m \equiv b^m \pmod r$ for any exponent m [7]. Thus Euler's formula $a^{\phi(r)} \equiv 1 \pmod r$ can be rewirtten as

$$a^{m\,\phi(r)} \equiv 1 \pmod r \tag{2-3}$$

where, as before, a is relatively prime to r. From the fact that $a \equiv b \pmod r$ implies that $ac \equiv bc \pmod r$ for any integer c, and from Equation 2-3, it follows that

$$X^{m\,\phi(r)\,+\,1} \equiv X \pmod r \tag{2-4}$$

where plaintext X is relatively prime to r (a restriction that is removed below). Let the public key (PK) and the secret key (SK) be chosen so that

$$SK \cdot PK = m\phi(r) + 1 \tag{2-5}$$

or, equivalently,

$$SK \cdot PK \equiv 1 \pmod{\phi(r)} \tag{2-6}$$

(A method for finding SK and PK satisfying this equation is discussed below.) Equation 2-4 can therefore be rewritten as

$$X^{SK \cdot PK} \equiv X \pmod r$$

which holds true for any plaintext (X) that is relatively prime to the modulus (r). (Actually, as shown below, the relation holds for any plaintext (X), and thus the restriction can be removed.)

Encipherment and decipherment can now be interpreted as follows:

$$E_{PK}(X) = Y \equiv X^{PK} \pmod{r} \tag{2-7}$$

$$D_{SK}(Y) \equiv Y^{SK} \pmod{r} \equiv X^{PK \cdot SK} \pmod{r} \equiv X \pmod{r} \tag{2-8}$$

Moreover, because multiplication is a commutative operation (i.e., SK·PK = PK·SK), it follows that encipherment followed by decipherment is equivalent to decipherment followed by encipherment:

$$D_{SK}(E_{PK}(X)) = E_{PK}(D_{SK}(X)) \equiv X \pmod{r} \tag{2-9}$$

As mentioned above, this property is useful for generating digital signatures (see Digital Signatures, Chapter 10).

Because $X^{PK} \pmod{r} \equiv (X + mr)^{PK} \pmod{r}$ for any integer m, each plaintext X, X+r, X+2r, . . . , results in the same ciphertext. Thus the transformation from plaintext to ciphertext is many-to-one. But restricting X to the set $\{0, 1, \ldots, r-1\}$ makes the transformation one-to-one, and thus encipherment and decipherment can be achieved as described in Equations 2-7 and 2-8.

Consider the example in which r equals $2 \cdot 3 = 6$ and $\phi(r)$ therefore equals $1 \cdot 2 = 2$. As predicted by Euler's theorem, $X^{\phi(r)} \equiv 1 \pmod{r}$ for values of X in the set $\{0, 1, \ldots, 5\}$ which are relatively prime to $r = 6$. However, one observes that $X^{\phi(r)+1} \equiv X \pmod{r}$ for all values of X in the set $\{0, 1, \ldots, 5\}$, as shown in Table 2-2. A proof is now given that the relationship $X^{m\phi(r)+1} \equiv X$

X	$X^{\phi(r)} \pmod{r}$	$X^{\phi(r)+1} \pmod{r}$
0	0	0
1	1	1
2	4	2
3	3	3
4	4	4
5	1	5

Legend:
$p = 2$, $q = 3$, $r = 6$, $\phi(r) = 2$
Set of Xs relatively prime to r: {1, 5}
Set of Xs relatively prime to p: {1, 3, 5}
Set of Xs relatively prime to q: {1, 2, 4, 5}

Table 2-2. Evaluation of $X^{\phi(r)+1} \pmod{r}$

(mod r) holds for any plaintext, X, where r = pq is the product of two prime factors and X is restricted to the set $\{0, 1, \ldots, r-1\}$—a condition which is necessary for encipherment and decipherment.

The theorem holds trivially for X = 0, and so only the case X > 0 must be considered. If X is not relatively prime to r = pq, then X must contain either p or q as a factor. Suppose p is a factor of X, so that the relation X = cp holds for some positive integer c. Since X is restricted to the set $\{0, 1, \ldots, r-1\}$, and r equals pq, it follows that X must be relatively prime to q. Otherwise, X would also contain q as a factor, in which case it would exceed r − 1. Using Euler's theorem, we have

$$X^{\phi(q)} \equiv 1 \pmod{q}$$

where $\phi(q) = q - 1$. But

$$X^{m(p-1)\phi(q)} \equiv 1^{m(p-1)} \equiv 1 \pmod{q}$$

for any integer m, and $(p-1)\phi(q) = (p-1)(q-1) = \phi(r)$, so that

$$X^{m\phi(r)} \equiv 1 \pmod{q}$$

or, for some integer n

$$1 = X^{m\phi(r)} + nq$$

Multiplying each side by X = cp results in

$$\begin{aligned} X &= X^{m\phi(r)+1} + (nq)(cp) \\ &= X^{m\phi(r)+1} + ncr \end{aligned}$$

or,

$$X^{m\phi(r)+1} \equiv X \pmod{r}$$

The case in which q is a factor of X can be handled in the same manner, thus completing the proof.

Procedures are now discussed for using the proposed algorithm. In particular it is shown how a user can create a pair of keys: public key (PK) and secret key (SK).

The user selects two prime numbers, p and q, where $p \neq q$. The product r = pq is made public, but p and q are kept secret. Note, for example, that the choice p = q is unacceptable, since p could then be obtained by taking the square root of the publicly known modulus (r). Even if the difference d = (p − q) is nonzero, d must still be unpredictable, since otherwise p and q could be determined from r. Note that (p + q) is the square root of $(p-q)^2 + 4r$, and q is half the difference of (p + q) and (p − q).

The public and secret keys must now be selected such that they satisfy Equation 2-6, that is,

$$PK \cdot SK \equiv 1 \pmod{\phi(r)}$$

In addition, it must be easy to compute PK and SK. The question thus arises as to how PK and SK can be chosen to satisfy these requirements. The following theorem [7] provides the answer.

Let the notation d = (a, n) be used to indicate that d is the greatest common divisor (gcd) of a and n. Then the congruence $aX \equiv b \pmod{n}$ is solvable (i.e., an integer X can be found that satisfies the congruence) only if the gcd of a and n divides b, and when this is the case there are d solutions [7].

If a and n are respectively defined as SK and $\phi(r)$, then gcd (SK, $\phi(r)$) divides 1 if and only if gcd (SK, $\phi(r)$) = 1, that is, if and only if SK is relatively prime to $\phi(r)$. And so, the congruence $SK \cdot X \equiv 1 \pmod{\phi(r)}$, where X is defined as PK, has a solution only if SK is relatively prime to $\phi(r)$. (Note that if a = PK, the solution X would be SK.) Moreover, because Euclid's algorithm (discussed below) provides an efficient method both to test whether a randomly chosen SK is relatively prime to $\phi(r)$ and to find the solution (X) of the congruence $SK \cdot X \equiv 1 \pmod{\phi(r)}$, the theorem above provides an efficient means of finding PK and SK.

For example, let p = 47 and q = 61. (Methods for generating prime numbers are treated separately. See Testing for Primality.) Thus r = pq = 2867 and $\phi(r) = (p - 1)(q - 1) = 2760$.

The method for determining the gcd of two integers, and therefore, a test as to whether two integers are relatively prime, is based on the *Euclidean algorithm* (from Euclid's *Elementa*, seventh book, circa 300 B.C.); namely, if a = bn + c, then the gcd of a and b equals the gcd of b and c. Thus, one can solve for gcd (a, b) by progressively reducing the size of the numbers whose gcd we are trying to find. For purposes of illustration, let $a = 38 = 2 \cdot 19$ and $b = 26 = 2 \cdot 13$. Observe that 19 and 13 are primes, and therefore that 2 is the greatest common divisor of a and b. The same result is obtained with Euclid's algorithm.

1. $38 = 26 \cdot 1 + 12$ 26 divides 38 one time with a remainder of 12
2. $26 = 12 \cdot 2 + 2$ 12 divides 26 two times with a remainder of 2
3. $12 = 2 \cdot 6$

The last nonvanishing remainder (the value of 2 in the above example) is the gcd of a = 38 and b = 26. Even for very large integers, the Euclidean algorithm requires only a small number of steps to find the gcd.

With the aid of Euclid's algorithm, it can now be shown (for the example p = 47, q = 61, and $\phi(r) = 2760$) that SK = 167 is a candidate for the secret key.

$$2760 = 167 \cdot 16 + 88 \qquad (2\text{-}10a)$$

$$167 = 88 \cdot 1 + 79 \qquad (2\text{-}10b)$$

$$88 = 79 \cdot 1 + 9 \tag{2-10c}$$

$$79 = 9 \cdot 8 + 7 \tag{2-10d}$$

$$9 = 7 \cdot 1 + 2 \tag{2-10e}$$

$$7 = 2 \cdot 3 + 1 \quad \text{(1 is the last nonvanishing remainder)} \tag{2-10f}$$

$$2 = 1 \cdot 2 \tag{2-10g}$$

The value of PK can be found by using a variation of Euclid's algorithm, which has already been used in computing the gcd of SK and $\phi(r)$. The goal is to rewrite Equations 2-10a through 2-10g in such a way that the final result is in the form

$$(\text{factor}_1 \cdot \text{SK}) + (\text{factor}_2 \cdot \phi(r)) = 1$$

in which case, factor_1 is interpreted as PK. (Note that this expression is equivalent to $\text{PK} \cdot \text{SK} \equiv 1 \pmod{\phi(r)}$.)

Let SK = 167 and $\phi(r) = 2760$, where p = 47 and q = 61. The public key can be computed using Equation 2-10f.

$$1 = 7 - 2 \cdot 3 \tag{2-11a}$$

Substituting $2 = 9 - 7 \cdot 1$ (Equation 2-10e) into Equation 2-11a results in

$$1 = 7 - 9 \cdot 3 + 7 \cdot 3 = 7 \cdot 4 - 9 \cdot 3 \tag{2-11b}$$

Substituting $7 = 79 - 9 \cdot 8$ (Equation 2-10d) into Equation 2-11b results in

$$1 = 79 \cdot 4 - 9 \cdot 32 - 9 \cdot 3 = 79 \cdot 4 - 9 \cdot 35 \tag{2-11c}$$

Substituting $9 = 88 - 79 \cdot 1$ (Equation 2-10c) into Equation 2-11c results in

$$1 = 79 \cdot 4 - 88 \cdot 35 + 79 \cdot 35 = 79 \cdot 39 - 88 \cdot 35 \tag{2-11d}$$

Substituting $79 = 167 - 88 \cdot 1$ (Equation 2-10b) into Equation 2-11d results in

$$1 = 167 \cdot 39 - 88 \cdot 39 - 88 \cdot 35 = 167 \cdot 39 - 88 \cdot 74 \tag{2-11e}$$

Finally, substituting $88 = 2760 - 167 \cdot 16$ (Equation 2-10a) into Equation 2-11e results in

$$1 = 167 \cdot 1223 - 2760 \cdot 74 \tag{2-11f}$$

From Equation 2-11f, it can be seen that 1223 is the multiplicative inverse of 167 modulo 2760, and therefore that PK = 1223 is the public key corresponding to SK = 167.

In summary, the following numerical values were obtained in the example.

$p = 47$	(chosen)
$q = 61$	(chosen)
$r = pq = 47 \cdot 61 = 2867$	(derived)
$\phi(r) = (p - 1)(q - 1) = 46 \cdot 60 = 2760$	(derived)
$SK = 167$	(chosen)
$PK = 1223$	(derived)

A message to be enciphered is first divided into a series of blocks such that the value of each block does not exceed $r - 1$. (Otherwise, a unique plaintext representation is not possible.) This could be achieved by substituting a two-digit number for each letter of the message). For example, blank = 00, A = 01, B = 02, . . . , Z = 26. Thus, the message "RSA ALGORITHM" would be written in blocks as

1819 0100 0112 0715 1809 2008 1300

The first plaintext block, 1819, is enciphered by raising it to the power PK = 1223, dividing by r = 2867, and taking the remainder, 2756, as the ciphertext. Likewise, 2756 is deciphered by raising it to the power SK = 167, dividing by r = 2867, and taking the remainder, 1819, as the recovered plaintext. The total ciphertext of the example is as follows:

2756 2001 0542 0669 2347 0408 1815

Since PK = 10011000111 in binary (or $2^{10} + 2^7 + 2^6 + 2^2 + 2^1 + 2^0$ or 1024 + 128 + 64 + 4 + 2 + 1), the first plaintext block, 1819, is enciphered as:

$$\begin{aligned} 1819^{1223} &\equiv 1819^{1024} \cdot 1819^{128} \cdot 1819^{64} \cdot 1819^{4} \cdot 1819^{2} \cdot 1819^{1} \\ &\equiv 2756 \pmod{2867} \end{aligned}$$

Since PK contains 11 bits, there are 10 repeated squaring operations needed to compute the intermediate quantities: $1819^2, 1819^4, 1819^8, \ldots, 1819^{1024}$. The cumulative total is then multiplied by each intermediate result if there is a corresponding 1 bit in the key.[10] Except for the value of the exponent, the operations of encipherment and decipherment are the same.

The following summary describes the procedure for selecting keys and performing the steps of encipherment and decipherment:

1. Two secret prime numbers, p and q, are selected randomly.

[10]The computation is easier than it may seem, since the mod r can be applied to each intermediate result with the same end result.

2. The public modulus, r = pq, is calculated.
3. The secret Euler totient function, $\phi(r) = (p - 1)(q - 1)$, is calculated.
4. A quantity, K, is selected, which is relatively prime to $\phi(r)$. K is defined as either the secret key, SK, or the public key, PK.
5. The multiplicative inverse of K modulo $\phi(r)$ is calculated using Euclid's algorithm, and this quantity is defined to be either the public key, PK, or the secret key, SK, depending on the choice made in (4).
6. Encipherment is performed by raising the plaintext, X (whose value is in the range 0 to r − 1), to the power of PK modulo r, thus producing the ciphertext, Y (whose value is also in the range 0 to r − 1).
7. Decipherment is performed by raising the ciphertext, Y, to the power of SK modulo r.

The Distribution of Primes

To thwart an opponent using exhaustive methods to obtain the secret primes, one must choose p and q from a sufficiently large set. But at the same time the method used to find p and q must be computationally efficient.

The largest tables of prime numbers ordinarily contain only a few thousand entries and are too small to be of use. On the other hand, computing and storing a table of prime numbers large enough to provide adequate security is clearly out of the question.

At the present, the most practical method of selecting primes suitable for use in the RSA algorithm is to test randomly selected integers until the required number of primes have been found. The approach works only because the proportion of primes to nonprimes is high enough.

By actual count, one finds that each group of 100 numbers from 1 to 1000 (1 to 100, 101 to 200, etc.) contains respectively, the following number of primes:

25, 21, 16, 16, 17, 14, 16, 14, 15, 14

In each group of 100 numbers from 1,000,001 to 1,001,000, the corresponding frequency of primes is

6, 10, 8, 8, 7, 7, 10, 5, 6, 8

and from 10,000,001 to 10,001,000 the corresponding frequency is

2, 6, 6, 6, 5, 4, 7, 10, 9, 6

A computation by M. Kraitchik [7] shows that for each group of 100 numbers in the interval from $10^{12} + 1$ to $10^{12} + 1000$ the corresponding frequency of primes is

4, 6, 2, 4, 2, 4, 3, 5, 1, 6

Even though the prime numbers gradually become more scarce as the numbers within the groups become larger, there are still infinitely many primes.

According to the prime number theorem, the ratio of $\pi(x)$, the number of primes in the interval from 2 to x, and $x/\ln(x)$ approaches 1 as x becomes very large, that is,

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln(x)} = 1$$

where $\ln(x)$ is the (natural) logarithm of x to the base $e = 2.71828\ldots$ For different intervals, a comparison of the actual number of primes [7] to the estimated number of primes (given by $x/\ln(x)$) is shown in Table 2-3.[11]

x	a π (x)	b x/ln(x)	a/b
1,000	168	145	1.159
10,000	1,229	1,086	1.132
100,000	9,592	8,686	1.104
1,000,000	78,498	72,382	1.084
10,000,000	664,579	620,421	1.071
100,000,000	5,761,455	5,428,681	1.061
1,000,000,000	50,847,478	48,254,942	1.054

Table 2-3. Number of Primes in Interval 2 to x

The probability that a randomly selected value in the interval from 2 to x is prime is approximately equal to $\pi(x)/(x-1)$, that is, the ratio of the number of primes ($\pi(x)$) to the total number of integers $(x-1)$. It can be shown that on the average about $(x-1)/\pi(x) \simeq \ln(x)$ values must be tested before a prime is found.[12] For example, if the magnitude of p and q were on the order of 2^{200}, then about $\ln(2^{200}) = 140$ trials (or 70 trials using odd numbers) would be needed to find a prime. (See Cyptographic Strength Considerations for a discussion of the magnitude of r.)

Testing for Primality

Several methods can be used to test a randomly selected number for primality. However, the most straightforward approaches are not computationally

[11] A better approximation of $\pi(x)$ can be obtained by evaluating the integral $\int_2^x dt/\ln(t)$.

[12] If the probability of finding a prime number is equal to p at each trial, then it takes on the average 1/p trials to find a prime number, assuming that the trials are statistically independent.

feasible. For example, a test could be based on Wilson's theorem [7], which states that

$$(p-1)! \equiv -1 \pmod{p} \qquad \text{if p is prime}$$

where

$$(p-1)! = 2 \cdot 3 \cdot \ldots \cdot (p-1)$$

In all other cases (except n = 4), it can be shown that

$$(n-1)! \equiv 0 \pmod{n} \qquad \text{if n is not prime}$$

Several examples are shown below.

$$\begin{aligned}
(2-1)! &\equiv -1 \pmod 2 & \qquad (7-1)! &\equiv -1 \pmod 7 \\
(3-1)! &\equiv -1 \pmod 3 & \qquad (8-1)! &\equiv \ 0 \pmod 8 \\
(4-1)! &\equiv \ 2 \pmod 4 & \qquad (9-1)! &\equiv \ 0 \pmod 9 \\
(5-1)! &\equiv -1 \pmod 5 & \qquad (10-1)! &\equiv \ 0 \pmod{10} \\
(6-1)! &\equiv \ 0 \pmod 6 & \qquad (11-1)! &\equiv -1 \pmod{11}
\end{aligned}$$

It should be obvious, however, that a test based on Wilson's theorem is useless for large values of p, since too many multiplications would be required to compute $(p-1)!$.

A different test could be based on the simple fact that if a number n is not prime, then n must contain a factor less than or equal to the square root of n. But even here the test is useless for large primes p, since to show that p is not divisible by any number between 2 and $\sqrt{p}$, and thus prove that p is prime, would still require too many computations.

The methods described thus far will determine with absolute certainty whether a number is prime or composite. However, adopting a procedure that is less reliable permits a favorable trade-off between computation time and the risk of accepting a number as prime when it is really composite. (Efficient procedures for testing a large number for primality are given in references 8 through 11.) To test a large number n for primality, one could use the elegant "probabilistic" algorithm of Solovay and Strassen [8]. It picks a random number a from a uniform distribution $(1, 2, \ldots, n-1)$ and tests whether

$$\gcd(a, n) = 1 \qquad \text{and} \qquad J(a, n) \equiv a^{(n-1)/2} \pmod{n} \qquad (2\text{-}12)$$

where J(a, n) is the *Jacobi symbol* [12]. If n is prime, then Equation 2-12 always holds. If n is composite, the Equation 2-12 will be false with probability of at least 1/2.

The number n can now be tested for primality by using a set of integers, $\mathbf{A} = \{a_1, a_2, \ldots, a_m\}$, where each a in **A** is less than n. The test requires that,

for each value of a in **A**, Equation 2-12 holds. Thus n is found to be composite if there is an a in **A** for which Equation 2-12 does not hold; otherwise n is accepted as prime.

The procedure does not guarantee that a selected number is prime, but only that it has not failed the test of primality. The greater the number of integers in **A**, the greater the probability that a selected number is prime. This can be argued as follows. If **A** contains m randomly selected integers from 1 to $n - 1$, then the probability that Equation 2-12 holds when n is composite is less than 0.5 for each value of a in **A**. So for a composite number, the probability that Equation 2-12 holds for all m values in **A** is less than 0.5^m. In other words, the probability that a composite number will pass the primality test is less than 0.5^m. If m is large, then the chance for error is small. For example, 0.5^m is 0.00098 and 0.00000095 for $m = 10$ and $m = 20$, respectively.

When n is odd, $a \leqslant n$, and gcd (a, n) = 1, the Jacobi symbol, J(a, n), has a value in $\{-1, 1\}$ and can be efficiently computed by the following recursive procedure [5]:

$$\begin{aligned} J(a, n) = \text{if } a = 1 \text{ then } 1 \text{ else} \\ \text{if } a \text{ is even then } J(a/2, n)(-1)^{(n^2-1)/8} \\ \text{else } J(n \pmod a, a)(-1)^{(a-1)(n-1)/4} \end{aligned}$$

A simple numerical example of testing a number for primality illustrates a different approach, one based on Euler's theorem. (The method is not recommended, but is given here because it is easy to understand.) Recall that Euler's theorem states that if p is prime, then

$$a^{p-1} \equiv 1 \pmod p$$

where a and p are relatively prime.

The number p is tested for primality by using a set of integers $\mathbf{A} = \{a_1, a_2, \ldots, a_m\}$ where each a in **A** is less than p. The test consists of ensuring that for each value of a in **A**, 1 is the remainder obtained when a^{p-1} is divided by p. (The procedure for evaluating $a^{p-1} \pmod p$ is the same as that described earlier for enciphering and deciphering data with the RSA algorithm.) Thus p is found to be composite if there is an a in **A** for which 1 is not the remainder obtained when a^{p-1} is divided by p; otherwise p is accepted as prime.

A further example illustrates the procedure's result when a prime number ($p = 1151$) and a composite number ($n = 1147$) are tested for primality using the set of integers $\mathbf{A} = \{106, 750, 479, 808, 1111, 223, 55, 848, 378, 729\}$ (Table 2-4). If the test for primality was based on the set $\mathbf{A} = \{750, 1111, 223\}$, then an incorrect conclusion would have been reached for the value $n = 1147$ (i.e., one would have said that the composite number 1147 is prime).

a	$a^{p-1} \pmod p$	$a^{n-1} \pmod n$
106	1	915
750	1	1
479	1	566
808	1	591
1111	1	1
223	1	1
55	1	841
848	1	1120
378	1	776
729	1	667

Table 2-4. Test of a Prime Number (p = 1151) and a Composite Number (n = 1147) for Primality

Cryptographic Strength Considerations

One approach that enables an opponent to break the RSA algorithm is to factor r. Once p and q are known, (p − 1) and (q − 1) can be used to compute $\phi(r) = (p - 1)(q - 1)$, and then SK could be calculated from $\phi(r)$ and PK by using Euclid's algorithm.

However, in the proposed scheme, each user chooses a pair of secret primes (p and q) which are large enough so that factorization of the non-secret modulus (r = pq) is not feasible, even with the help of high-speed computers, and given the fastest known method of factoring. It is therefore absolutely essential that r is large enough to make the work needed to factor r sufficiently great.

The fastest known factoring algorithm is that of Richard Schroeppel [5]. It can factor r in approximately $\ln(r)^{\mathrm{sqrt}(\ln(r)/\ln(\ln(r)))}$ steps. (ln denotes the natural logarithm function.) As a first order approximation, assume that the computation time needed to perform one step in the Schroeppel algorithm is the same as that to search one key in a hypothetical exhaustive attack against DES. In this case a blocksize of 388 bits would mean that the work needed to factor r is equivalent to the work needed to exhaust 2^{56} DES keys. Instead, if the computation time required to perform a step in the Schroeppel algorithm were 1000 (1 million) times greater than that required to search a single key in DES, then a blocksize of 280 (186) bits would be required to maintain equivalency.

According to the algorithm's inventors [5], additional protection against sophisticated factoring algorithms can be achieved by ensuring that the following conditions are met:

1. p and q differ in length by only a few bits.

2. Each number $(p - 1)$ and $(q - 1)$ contains a large prime factor, p' and q', respectively.
3. The gcd of $(p - 1)$ and $(q - 1)$ is small.

Moreover, it has also been pointed out that further protection is possible by ensuring that $(p' - 1)$ and $(q' - 1)$ have large prime factors, p'' and q'', respectively [13,14].

To find a suitable p, first find a large prime p'' and let p' be the first prime in the sequence $i \cdot p'' + 1$, for $i = 2, 4, 6, \ldots$, etc. Repeating the process, let p be the first prime in the sequence $i \cdot p' + 1$, for $i = 2, 4, 6, \ldots$, and so on. (A value for q can be found in a similar fashion.)

Without regard for the usual methods of factoring composite numbers, it is noteworthy that r could easily be factored if either $\phi(r)$ or SK were available. The significance of this fact is that it is just as hard to determine $\phi(r)$ or SK as it is to factor r. By way of an illustration, if $\phi(r)$ were available, then r could be obtained by the following steps:

1. Obtain $(p + q)$ from r and $\phi(r) = r - (p + q) + 1$.
2. Obtain $(p - q)$ from the equation $(p + q)^2 = p^2 + 2r + q^2 = (p - q)^2 + 4r$ by taking the square root of $(p + q)^2 - 4r$.
3. Obtain q as half the difference of $(p + q)$ and $(p - q)$.

On the other hand, having SK would permit $SK \cdot PK - 1$ to be computed, which is a multiple of $\phi(r)$. But an efficient method of factoring r is available if a multiple of $\phi(r)$ is known [9].

It should be obvious that finding a number (X) not relatively prime to r would be equivalent to breaking the algorithm. This is because the gcd of X and r would be equal to either p or q, and its value could be easily computed using Euclid's algorithm. However, in the practical sense, there is no need to be concerned that the algorithm will be broken by finding such a number (X), provided that r is sufficiently large. In the interval from 1 to r there are

$$\phi(r) = (p - 1)(q - 1) = pq - (p + q) + 1$$

numbers relatively prime to r, and

$$r - \phi(r) = (p + q) - 1$$

numbers not relatively prime to r. The probability of accidentally discovering a number having p or q as a factor is therefore equal to

$$\frac{r - \phi(r)}{r} = 1 - \frac{\phi(r)}{r} = \frac{p + q - 1}{pq} \simeq \frac{1}{q} + \frac{1}{p}$$

which is extremely small for large values of p and q.

Factoring large numbers is a well-known problem that has engaged mathematicians for many hundreds of years. Experience has shown it to be an in-

tractable problem. Yet this evidence does not prove that the cryptographic approach is strong. In fact, until the advent of high-speed computers, mathematicians weren't looking for methods that might require very complicated tests. Furthermore, the general problem of factoring and the special case of factoring associated with the RSA algorithm are different. The classical problem of factoring, not yet solved despite a considerable effort, can be stated as follows:

Factor a composite number r, where r may be any product of two or more prime factors.

The cryptographic problem, which must take into account attacks using selected ciphertext and is not yet sufficiently investigated, can be stated as follows:

Factor a composite number r, where r is the product of two prime factors[13] *(r = pq) and where there exists a public key PK and a secret key SK that satisfies the relation*

$$PK \cdot SK = 1 \ (mod \ (p-1)(q-1))$$

such that the opponent has knowledge of chosen ciphertext, $Y_1, Y_2, \ldots$, and corresponding recovered plaintext, $X_1, X_2, \ldots$ (without having knowledge of SK) which satisfy the relation

$$Y_i^{SK} \equiv X_i \ (mod \ r)$$

In the cryptographic problem, knowledge of PK (i.e., a value relatively prime to $(p-1)(q-1)$) does not provide an opponent with much information beyond that present in the classical problem. This is because in the classical problem it would be a simple matter to select a large prime (i.e., a value relatively prime to $(p-1)(q-1)$) that could be used as a public key to carry out a chosen plaintext attack. However, in the cryptographic problem, the public key can be used in conjunction with a chosen ciphertext attack to produce quantities that are functions of both PK and SK. In this sense, the public key is potentially of greater value in the cryptographic problem than it is in the classical problem.

As yet there is no evidence to support a claim that the additional information available to the opponent in the cryptographic problem will allow the modulus to be factored. However, one cannot conclude that the problem of factoring in the cryptographic problem is hard merely on the basis that the classical problem of factoring is known to be hard. And while factoring the modulus in the RSA algorithm leads to breaking the algorithm, there is no proof that breaking the algorithm is the same as solving the classical problem of factoring.

It is entirely possible that the proposed RSA algorithm is cryptographically strong. However, this conclusion cannot be reached from previous work done to solve the theoretic problems of factoring composite numbers. It can only be reached by taking into account the requirements that must be satis-

[13] In a more general approach, the RSA algorithm could specify that r is the product of more than two prime factors.

fied for strong algorithms and by performing a thorough validation. Since the algorithm is in the public domain and has become a topic of great interest among academicians and cryptologists [15-21], it is only fair to say that a validation effort of sorts has already begun. Nevertheless, in addition to such an effort, a well-organized approach by a group of dedicated people whose only task is to uncover weaknesses in the algorithm is needed. Finally, it would be highly desirable for the National Security Agency, where significant cryptographic expertise resides, to certify the algorithm's strength. This certification would be based on a similar government-organized validation effort.

Trapdoor Knapsack Algorithm[14]

A public-key algorithm can also be based on the classical problem in number theory known as the knapsack problem [4]. The following is an introduction to this approach. Let A be a nonsecret (published) vector of n integers $(a_1, a_2, \ldots, a_n)$ and let X be a secret vector of n binary digits (0s and 1s) whose components are designated $(x_1, x_2, \ldots, x_n)$, that is,

$$A = (a_1, a_2, \ldots, a_n)$$
$$X = (x_1, x_2, \ldots, x_n)$$

Defining Y to be the *dot product* of A and X results, by definition, in

$$Y = A \cdot X = a_1x_1 + a_2x_2 + \ldots + a_nx_n = \sum_{i=1}^{n} a_ix_i$$

Calculation of Y is simple, involving only a sum of at most n integers. However, finding X from Y and A is generally difficult when n is large and A is properly chosen. This is called the *knapsack problem*.

Let the knapsack problem be illustrated by the following simple example: If

$$X = (1, 0, 1, 1, 0, 0, 0, 1)$$
$$A = (2453, 6394, 941, 1076, 4791, 4404, 9549, 6639)$$

then

$$Y = A \cdot X = 2453 + 941 + 1076 + 6639 = 11109$$

In the knapsack problem, one is asked to find X such that $A \cdot X = Y$, where A and Y are given. In the most general case, one would like to have a function g to calculate X from A and Y such that g satisfies the relation $X = g(A, Y)$.

One way to find X is by the method of direct search. (In the above example, there are $2^8 = 256$ values for X.) This consists of computing $A \cdot X$

[14] At the time of publication of this book, the trapdoor knapsack algorithm reportedly has been broken [22].

for each enumerated value of X, and comparing the result with Y for equality. Function g, in this case, is a procedure to test all possibilities for X, and select the first which works. However, if the number of elements in A (and thus in X) is large, and A is properly chosen, then such an exhaustive approach is not practical. A different method of solution would be required.

In the described public-key algorithm, A represents the public key. Anyone can produce ciphertext Y from plaintext X by the equation $Y = A \cdot X$. But for this approach to be cryptographically strong, it must not be computationally feasible to obtain X from information assumed to be known to the cryptanalyst, thus preventing the process from being inverted by discovery of function g.

An example of a cryptographically weak approach (since it allows the process to be easily inverted) is a public key A whose elements satisfy the following conditions:

$$a_{i+1} > a_1 + a_2 + \ldots + a_i = \sum_{j=1}^{i} a_j; \qquad i = 1, 2, \ldots, n-1$$

Using the notation

$$\begin{aligned} Y_1 &= x_1 a_1 \\ Y_2 &= x_1 a_1 + x_2 a_2 \\ &\vdots \\ Y_n &= x_1 a_1 + x_2 a_2 + \ldots + x_n a_n \end{aligned}$$

where $Y = Y_n$ is the ciphertext, one can recover X from Y_n and A as follows. If Y_n is less than a_n, then x_n is set equal to 0 and Y_{n-1} is set equal to Y_n. Otherwise, x_n is set equal to 1 and Y_{n-1} is set equal to $Y_n - a_n$. Now, using the computed value of Y_{n-1}, one can compute the values of x_{n-1} and Y_{n-2} in a similar manner. The procedure continues until $X = (x_1, x_2, \ldots, x_n)$ has been recovered.

The recovery of X can be illustrated by the following example:

$$\begin{aligned} A &= (15, 92, 108, 279, 563, 1172, 2243, 4468) \\ Y &= A \cdot X = 4870 \end{aligned} \qquad (2\text{-}13)$$

Thus

$$\begin{aligned} x_8 &= 1, && \text{since } Y_8 (= 4870) > a_8 (= 4468) \\ x_7 &= 0, && \text{since } Y_7 (= 402) < a_7 (= 2243) \\ x_6 &= 0, && \text{since } Y_6 (= 402) < a_6 (= 1172) \\ x_5 &= 0, && \text{since } Y_5 (= 402) < a_5 (= 563) \\ x_4 &= 1, && \text{since } Y_4 (= 402) > a_4 (= 279) \end{aligned}$$

$$x_3 = 1, \quad \text{since } Y_3(= 123) > a_3(= 108)$$
$$x_2 = 0, \quad \text{since } Y_2(= 15) < a_2(= 92)$$
$$x_1 = 1, \quad \text{since } Y_1(= 15) = a_1(= 15)$$

and the value of X is (1, 0, 1, 1, 0, 0, 0, 1).

A trapdoor knapsack [4] is one in which the careful choice of vector A allows the designer to recover X from Y easily using the secret trapdoor (identified by the secret key), but which makes it difficult for anyone else to find the solution. The introduction of a secret quantity makes it possible to find a transformation such that X = g′ (A, Y, secret quantity), where the function g′ is easily calculated. The way the problem is solved here is to transform Y to Y′ by the following method:

1. Choose secret integers, r and t, which are relatively prime.
2. Calculate another quantity, s (also kept secret) from r and t, which is the multiplicative inverse of t modulo r.

In that case, the relations

$$Y' = Y\, s \pmod r$$
$$Y' = A'X$$

exist which allow easy recovery of X, since A′ falls into the class of knapsack problems which have easy solutions. In other words, a trapdoor is introduced (identified by the secret parameters r and t) that transforms a hard knapsack problem (vector A) into a trivial knapsack problem (vector A′).

To construct a trapdoor knapsack, let $A' = (a'_1, a'_2, \ldots, a'_n)$ be a secret vector of n integers such that $a'_i > a'_1 + a'_2 + \ldots + a'_{i-1}$ for all i. The vector

$$A' = (15, 92, 108, 279, 563, 1172, 2243, 4468)$$

used in the last example (Equation 2-13) satisfies this condition. Now choose secret integers r and t such that these three conditions hold:

1. $r > a'_1 + a'_2 + \ldots + a'_n$
2. $r > t$
3. r and t are relatively prime (i.e., gcd (r, t) = 1)

The choice r = 9291 and t = 2393 satisfies the necessary conditions. That they are relatively prime can be shown as follows, using Euclid's algorithm:

$$9291 = 2393 \cdot 3 + 2112$$
$$2393 = 2112 \cdot 1 + 281$$
$$2112 = 281 \cdot 7 + 145$$

$$281 = 145 \cdot 1 + 136$$
$$145 = 136 \cdot 1 + 9$$
$$136 = 9 \cdot 15 + 1$$
$$9 = 1 \cdot 9$$

The last nonvanishing remainder (the value 1 in the above computation) is the gcd of 9291 and 2393.

The easily solved knapsack vector A' is now transformed into a trapdoor knapsack vector A via the relation

$$a_i \equiv a_i' t \pmod{r}$$

Since

$$Y = A \cdot X = \sum_{i=1}^{n} a_i x_i$$

it follows that

$$Y \equiv \sum_{i=1}^{n} [a_i' t \pmod{r}]\, x_i$$
$$\equiv \sum_{i=1}^{n} a_i' x_i t \pmod{r}$$

Defining

$$Y' = \sum_{i=1}^{n} a_i' x_i = A' \cdot X$$

to be the transformed ciphertext from which X can be easily recovered, since A' is chosen that way, one obtains

$$Y \equiv Y' t \pmod{r} \qquad (2\text{-}14)$$

The idea here is to use the secret quantities t and r to transform Y to Y' and thus transform the hard knapsack problem into an easy one. To achieve this let a quantity s be defined such that

$$st \equiv 1 \pmod{r}$$

Hence s is the multiplicative inverse of t modulo r. If one defines t and r to be relatively prime (as stated in condition 3), there is a unique solution for s.

(This was discussed before in conjunction with the RSA algorithm. See Equations 2-10a through 2-10g.) Furthermore, to assure a unique relation between plaintext and ciphertext, choose a value of r that exceeds the maximum value of Y, that is,

$$r > \sum_{i=1}^{n} a_i$$

which satisfies condition 1.
Multiplying Y by s (see Equation 2-14) results in

$$Ys \equiv Y'st \pmod{r} \equiv Y' \pmod{r}$$

or equivalently

$$Y' \equiv Ys \pmod{r}$$

which is the desired result.

In the current example, $a_1 = 8022 \equiv 15 \cdot 2393 \pmod{9291}$, $a_2 = 6463 \equiv 92 \cdot 2393 \pmod{9291}$, and so on, and therefore vector A can be computed to be

$$A = (8022, 6463, 7587, 7986, 64, 8005, 6592, 7274)$$

Vector A (the public key) is published by the user. Anyone desiring to communicate a message (X) to the user enciphers the message using vector A. The ciphertext (Y) is obtained via the relation $Y = A \cdot X = \sum_{i=1}^{n} a_i x_i$. To recover the original message (X) from the ciphertext (Y), Y is transformed into Y′ using s, namely

$$Y' = Y \cdot s \bmod r$$

and the solution is obtained using the knapsack vector A′.

In the present example, the value of s is computed by rewriting the equations previously obtained with Euclid's algorithm:

$$1 = 136 - 9 \cdot 15$$
$$1 = 16 \cdot 136 - 16 \cdot 145$$
$$1 = 16 \cdot 281 - 31 \cdot 145$$
$$1 = 233 \cdot 281 - 31 \cdot 2112$$
$$1 = 233 \cdot 2393 - 264 \cdot 2112$$
$$1 = 1025 \cdot 2393 - 264 \cdot 9291$$

Thus s = 1025 is the multiplicative inverse of t = 2393 modulo r = 9291.

The trapdoor knapsack public-key algorithm is illustrated by the following example:

$A' = (15, 92, 108, 279, 563, 1172, 2243, 4468)$	(secret, chosen)
$r = 9291$	(secret, chosen)
$t = 2393$	(secret, chosen)
$s = 1025$	(secret, derived)
$A = (8022, 6463, 7587, 7986, 65, 8005, 6592, 7274)$	(nonsecret, derived)

A message

$$X = (1, 0, 1, 1, 0, 0, 0, 1)$$

is enciphered using vector A, as follows.

$$Y = A \cdot X = (8022 + 7587 + 7986 + 7274) = 30869$$

Multiplying Y by the secret value of s results in

$$Y' = Y \cdot s \pmod{r} = 30869 \cdot 1025 \pmod{9291} = 4870$$

Subsequently, $X = (1, 0, 1, 1, 0, 0, 0, 1)$ is recovered from $Y' = 4870$ and vector A', as previously shown in Equation 2-13.

STREAM CIPHERS

A *stream cipher* (Figure 2-8) employs a bit-stream generator to produce a stream of binary digits called a *cryptographic bit-stream,*[15] which is then combined either with plaintext (via the $\boxplus$ operator) to produce ciphertext, or with ciphertext (via the $\boxplus^{-1}$ operator) to recover plaintext.

Vernam [23] was the first to recognize the merit of a cipher in which ciphertext (Y) is produced from plaintext (X) by combining it with a secret bit-stream (R) via a simple and efficient operation. In his cipher, Vernam used an Exclusive-OR operation, or modulo 2 addition (Table 2-1), to combine the bit-streams. Thus encipherment and decipherment are defined by $X \oplus R = Y$ and $Y \oplus R = X$, respectively, and the condition $\boxplus = \boxplus^{-1} = \oplus$ is satisfied. Since in most stream cipher designs modulo 2 addition is used as the combining operation, it will be used in the remainder of the discussion on stream ciphers.

[15]Traditionally, the term key-stream has been used to denote the output of the bit-stream generator. Instead, the term cryptographic bit-stream is used here to avoid possible confusion with a fixed-length cryptographic key in cases where a cryptographic algorithm is used as the bit-stream generator.

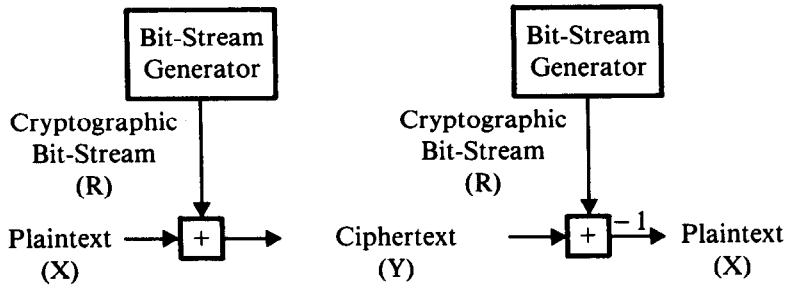


Figure 2-8. Stream Cipher Concept

If the bit-stream generator were truly random, an unbreakable cipher could be obtained by Exclusive-ORing the plaintext and cryptographic bit-stream. (See the discussion of one-time tape systems, Cryptographic Algorithms.) In that case, the cryptographic bit-stream is used directly as the key and is equal in length to the message. But because the cryptographic bit-stream is random, it must be provided to the users in advance via some independent and secure channel. This, of course, introduces insurmountable logistical problems if the intended data traffic is very large. Hence, for practical reasons, the bit-stream generator must be implemented as an algorithmic procedure, so that the cryptographic bit-stream can be produced by both users. In such an approach (Figure 2-9), the bit-stream generator is a key-controlled algorithm and must produce a cryptographic bit-stream which is cryptographically strong.

When modulo 2 addition is used as the combining operation, each bit in the output ciphertext (recovered plaintext), is dependent upon the corresponding bit in the input plaintext (ciphertext), but not upon any other bits in the input plaintext (ciphertext). This is in marked contrast to the block cipher which exhibits a much more complex relationship between bits in the plaintext (ciphertext) and bits in the ciphertext (recovered plaintext). Both approaches, however, have comparable strength.

In a stream cipher, the ciphering algorithm (G) uses a cipher key (v) to

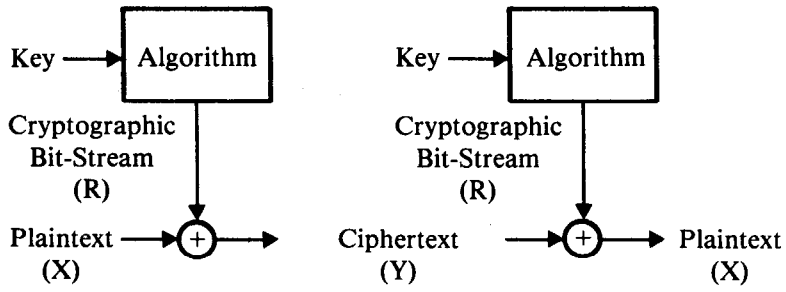


Figure 2-9. Stream Cipher Using an Algorithmic Bit-Stream Generator and Modulo 2 Addition

generate a cryptographic bit-stream (R). If the set of keys is represented, as before, by

$$\mathbf{V} = \{v_1, v_2, \ldots, v_r\}$$

it follows that the set of enciphering and deciphering functions (G) can be expressed as follows.

$$\mathbf{G} = \{g_{v_1}, g_{v_2}, \ldots, g_{v_r}\}$$

where g_v represents a key-selected transformation which generates a particular bit-stream. Function g should not be confused with the function introduced earlier for the block cipher design (Figure 2-7).

In a stream cipher, the algorithm may generate its bit-stream on a bit-by-bit basis, or in blocks of bits. This is of no real consequence. All such systems are stream ciphers, or variations thereof. Some variations, however, have important characteristics. Moreover, since bit streams can be generated in blocks, it is always possible for a block cipher to be used to obtain a stream cipher. However, in a communications system, because both the sender and receiver must produce cryptographic bit-streams that are equal and secret, their keys must also be equal and secret. In effect, this means that a public-key algorithm can be used to obtain a stream cipher only if it is used as a conventional algorithm. That is, both sender and receiver use the same algorithm (E or D) and the same key. But the key must be kept secret.

Consider the general case where an input block (X) of b bits is enciphered by generating a cryptographic bit-stream (R) of b bits and Exclusive-ORing R with X to produce b bits of ciphertext (Y).

$$Y = X \oplus R$$

From the rules of modulo 2 addition (Table 2-1), it follows that X can be recovered by adding the same cryptographic bit-stream (R) to the ciphertext (Y).

$$X = Y \oplus R$$

The ciphering procedure using modulo 2 addition is thus extremely simple and easy to implement. However, care must be taken to achieve a cryptographically strong design. If, for example, the opponent knows that modulo 2 addition has been performed, and plaintext (X) and corresponding ciphertext (Y) become available, he then could add both quantities together (modulo 2) and recover the cryptographic bit-stream.

$$X \oplus Y = X \oplus (X \oplus R) = R$$

Since the cryptographic key (K) is a constant quantity, it follows that the cryptographic bit-stream (R), or block of bits produced at each iteration of

the ciphering algorithm, will not change if it depends only on K. In this case, once the opponent has obtained R, he can decipher any intercepted ciphertext without ever knowing the key (K). This, of course, is unacceptable.

The stream cipher must not start from the same initial conditions in a predictable way, and thereby regenerate the same cryptographic bit-stream at each iteration of the algorithm. In other words, the stream cipher must not *reoriginate*.[16]

Since the key, even though it is secret, does not ensure an unpredictable cryptographic bit-stream, another quantity, defined as the *initializing vector* (Z), must be introduced into the ciphering process. (Other terms used are *seed* and *fill*.) In effect, different initializing vectors cause different cryptographic bit-streams to be generated. And the cryptographic bit-stream is unpredictable as long as the initializing vector satisfies one of the following conditions.

1. *Random.* Z is produced by some natural phenomenon whose statistics have been demonstrated to be random, and Z has enough combinations so that the probability of repeating is extremely small.
2. *Pseudo-random.* Z is produced by a deterministic process whose period (the interval between equal recurring values) is extremely large compared to the length of Z, and whose values have the statistical properties of randomness.
3. *Nonrepeating.* Under certain conditions, Z can be produced by a process that may be predictable, but whose period before repeating is so large that for practical purposes it is of no concern. A 64-bit nonresettable counter would satisfy this condition. Even if the opponent obtains the cryptographic bit-stream associated with one counter setting, he cannot determine what the bit-stream will be for a different counter setting.

In contrast to the cipher key, which must be kept secret, the initializing vector may be a nonsecret quantity. This is because the initializing vector either does not repeat, or else repeats with only a small probability (determined by the length of the initializing vector).

The cryptographic bit-stream R generated by the function g can now be expressed by

$$R = g_K(Z)$$

The encipher and decipher operations are thus defined by

$$Y = X \oplus R = X \oplus g_K(Z)$$
$$X = Y \oplus R = Y \oplus g_K(Z)$$

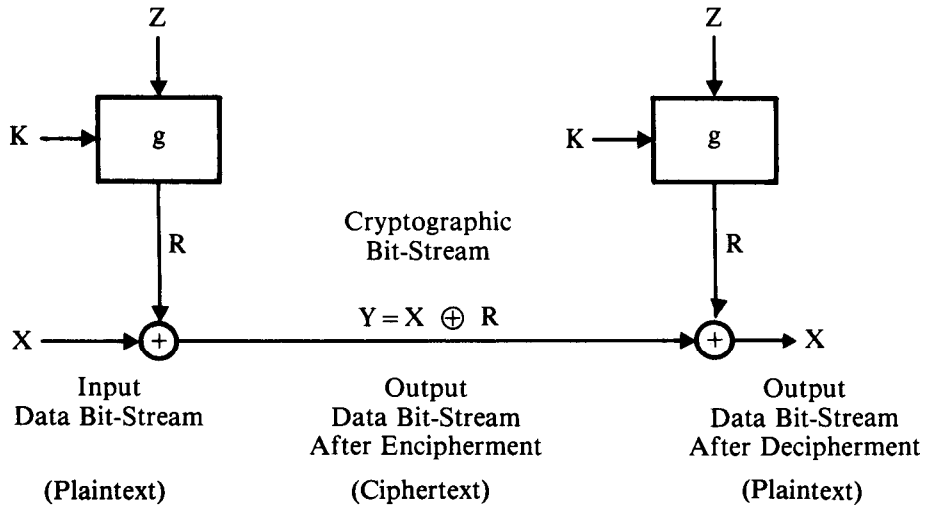
[16]This is not a requirement for the block cipher, since knowledge of plaintext and corresponding ciphertext does not permit an opponent to decipher without knowledge of the key.

It follows that the set of functions **G**, which determines the cryptographic bit stream, does not have to be a collection of one-to-one functions since an inverse operation is never needed. On the other hand, a set of functions $\mathbf{F} = \{f_{K,Z}\}$ does exist which relates plaintext and ciphertext using the keys and initializing vector as parameters

$$Y = f_{K,Z}(X)$$
$$X = f^{-1}_{K,Z}(Y)$$

where $f_{K,Z}$ is, of course, a one-to-one function. Since the length of the ciphertext is equal to the length of the plaintext, the number of plaintext combinations is equal to the number of ciphertext combinations. Hence $f_{K,Z}$ is also an onto function. (Note that the domain of $f_{K,Z}$ is the set of all plaintext combinations, the co-domain is the set of all ciphertext combinations.) The basic idea of a stream cipher is shown in Figure 2-10.

In a stream cipher, Z is used not only for providing cryptographic strength but also for establishing synchronization between communicating cryptographic devices. It assures that the same cryptographic bit-streams are



Legend: example of encipherment and decipherment

⊕	Plaintext	0 1 0 1
	Cryptographic Bit-Stream	0 0 1 1
	Ciphertext	0 1 1 0
⊕	Ciphertext	0 1 1 0
	Cryptographic Bit-Stream	0 0 1 1
	Recovered Plaintext	0 1 0 1

Figure 2-10. Stream Cipher

generated for the sender and the receiver. This may be accomplished by generating Z at the sending device and transmitting it in clear form to the receiver. An alternative method is for the receiver to determine Z by transmitting it to the sender. But this requires an additional initialization message, and hence is less efficient. However, it does provide a way to introduce a time-dependent parameter controlled by the receiver.

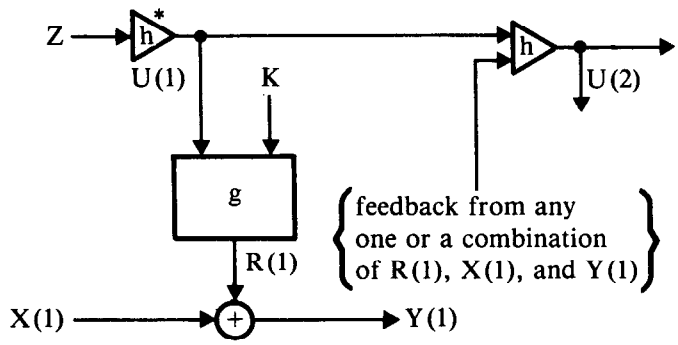
Recall that for a strong cryptographic algorithm it is assumed that the opponent has the advantage of relatively large amounts of selected plaintext and corresponding ciphertext. However, when the algorithm is a stream cipher this means that the opponent also has knowledge of large portions of the cryptographic bit-stream, since the cryptographic bit-stream can be reconstructed by modulo 2 addition of known plaintext and corresponding ciphertext. (Note that if $Y = X \oplus R$, then $Y \oplus X = R$.) It is important, therefore, that knowledge of part of the cryptographic bit-stream does not allow portions of the remaining cryptographic bit-stream to be determined. Hence, a necessary, although insufficient condition to achieve cryptographic strength with the stream cipher is for the bit-stream produced by the algorithm to be pseudo-random.

A bit-stream is considered to be pseudo-random if on statistical grounds one cannot reject the hypothesis that it is random (i.e., it passes all conceivable tests of randomness). Pseudo-randomness assures that it will be difficult for an opponent to use statistical attacks successfully against the cryptographic algorithm.

In addition to the opponent knowing relatively large amounts of selected plaintext and corresponding ciphertext, it is assumed for a strong stream cipher that the opponent knows the initializing vectors corresponding to the given plaintext and ciphertext.

Cryptographic systems usually treat initializing vectors as nonsecret quantities. Thus in a communications system, initializing vectors are no more difficult to intercept than ciphertext. Since the algorithm and cryptographic key are fixed, a variable cryptographic bit-stream is obtained by varying the initializing vector. One way to do this is to use a new initializing vector for each iteration of the ciphering algorithm (i.e., for each new block of bits produced in the cryptographic bit-stream). However, in a communications system, this has the disadvantage of increasing the amount of transmitted data, since the initializing vector bits are now added to each block of ciphertext bits. A more efficient approach is to use a single initializing vector for each message. (In general, a message consists of several blocks.) At the first iteration of the ciphering algorithm, the initializing vector is used (as before) to produce a block of bits in the cryptographic bit-stream, and these bits are then used to encipher the first block of plaintext. At all subsequent iterations of the ciphering algorithm, the initializing vector is altered by or determined from information obtained using *feedback* techniques. In this case, the bit-streams available at time i are used to produce an *intermediate initializing vector*, U, which is then used in the ciphering process at time $i + 1$.

A feedback can be obtained from several places: the cryptographic bit-stream, the plaintext, the ciphertext, or some combination thereof. Each of these approaches can give rise to a cryptographic system with differing char-



Note: The cryptographic bit-stream used to encipher the first block of plaintext bits depends only on the secret key and the non-secret initializing vector.

Figure 2-11. Encipherment of First Block of Plaintext Using a Stream Cipher

acteristics with respect to recovery from ciphertext errors. But regardless of what feedback technique is used, the first block of plaintext, X(1), is enciphered by a cryptographic bit-stream, R(1), which depends only on an initializing vector, Z, and a cipher key, K (Figure 2-11).

In the most general case, the length of the intermediate initializing vector (U) may not equal the length of the initializing vector (Z). For example, U(1) might be obtained by concatenating zero bits to Z. To accommodate such situations, a function h* is introduced to define how U(1) is obtained from Z:

$$U(1) = h^*(Z)$$

where encipherment of the first block of plaintext is given by

$$Y(1) = X(1) \oplus g_K(U(1))$$

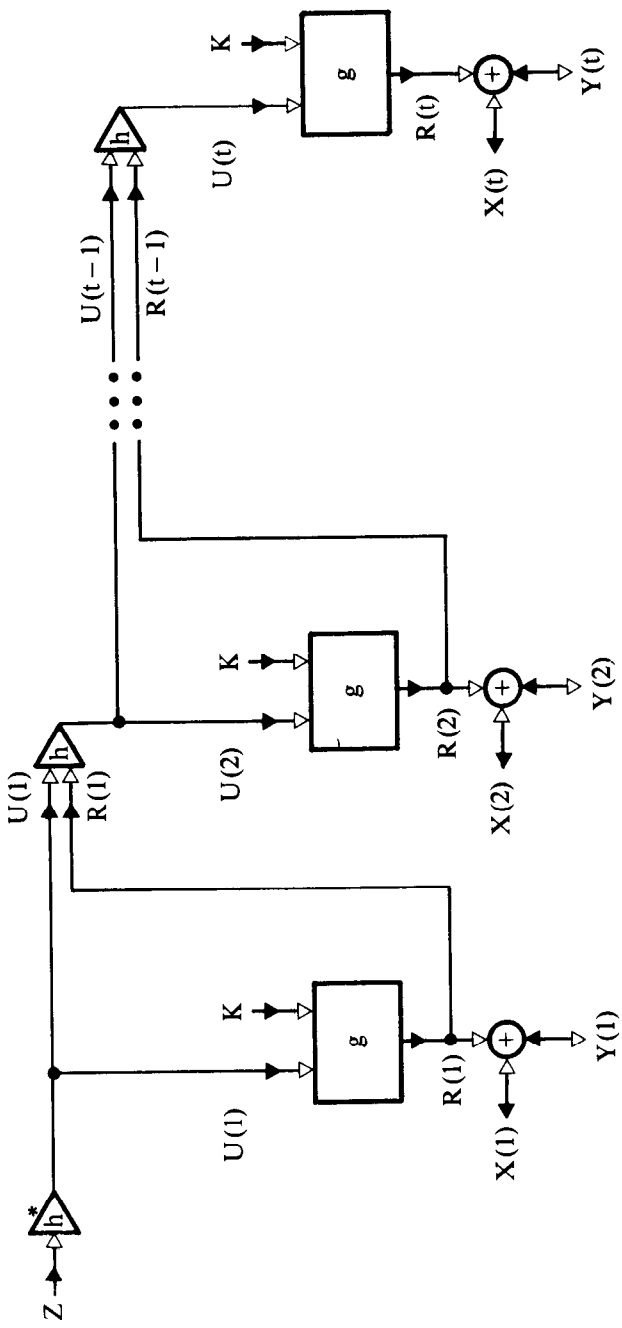
A method for generating the initial condition U(2) is considered next. Let the intermediate initializing vector at time i, U(i), be a function h of the previous initializing vector, U(i − 1), as well as an additional feedback quantity:

$$U(i) = h[U(i-1), \text{feedback quantity}]$$

However, since U(1) equals h*(Z), it follows that U(2) is given by

$$U(2) = h[h^*(Z), \text{feedback quantity}]$$

as shown in Figure 2-11. It should be understood that the intermediate



Legend: encipherment mode: ⟶, decipherment mode: ⟵

Note: Function h* determines the number of bits in each intermediate value U(i). The message is X = X(1), X(2), ..., X(t), where t can be as large as the user desires.

Figure 2-12. Key Auto-Key Cipher

initializing vectors must satisfy the same conditions as the initializing vector Z. The function h must therefore not introduce a bias into the ciphering process that could make the U values predictable.

The special case where the feedback is obtained from the cryptographic bit-stream is shown in Figure 2-12 and is defined as the *key auto-key cipher*. One property of this cipher is that an error in the ciphertext produces an error only in the corresponding bit positions of the recovered plaintext (i.e., there is no error expansion due to the ciphering process).

There are many ways to design a key auto-key cipher. The component common to all of these designs is that the feedback must be obtained from the cryptographic bit-stream. In general, the following relationships hold for a key auto-key cipher:

$$Y(i) = X(i) \oplus R(i)$$
$$X(i) = Y(i) \oplus R(i)$$

where

$$R(i) = g_K(U(i)); \quad 1 \leqslant i \leqslant t$$
$$U(i) = \begin{cases} h^*(Z); & i = 1 \\ h[U(i-1), R(i-1); & i > 1 \end{cases}$$

where $U(1) = h^*(Z)$ is the initial seed value, U(i) is the new seed at iteration $i > 1$, and h is a simple function of two arguments.

Some of the differences between block and stream ciphers can now be stated.

1. The block cipher enciphers a single block of data at one time. It requires a minimum blocksize determined by considerations of cryptographic strength. The stream cipher requires no minimum blocksize; it can be used to encipher, in the extreme case, on a bit-by-bit basis.
2. In the block cipher, every ciphertext bit is a complex function of every plaintext bit in the corresponding input block. In the stream cipher, every ciphertext bit y(i) is related to its corresponding plaintext bit x(i) by the relationship $y(i) = x(i) \oplus r(i)$.
3. The block cipher may or may not require an initializing vector (Z); it is allowed to reoriginate. This is because knowledge of plaintext and corresponding ciphertext does not reveal information in the same way that it would in the case of the stream cipher.[17] A cryptographically strong stream cipher must not reoriginate, and thus requires an initializing vector (Z).

[17] Although an initializing vector is not always a requirement for a block cipher, it is nevertheless used in block chaining. But even there the block cipher may reoriginate, since the initializing vector could be reused for a limited period of time.

BLOCK CIPHERS WITH CHAINING

The overall strength of a cryptographic system can be enhanced by using a technique known as chaining. *Chaining* is a procedure used during the ciphering process which makes an output block dependent not only on the current input block and key, but also on earlier input and/or output.

In certain applications, data to be enciphered may contain patterns that are longer than the cipher's blocksize. Such patterns in the plaintext may result in similar patterns in the ciphertext which could be exploited by an opponent. Chaining significantly reduces the presence of repetitive patterns in the ciphertext, because with chaining two identical blocks of plaintext will, upon encipherment, result in different ciphertext blocks.

Patterns Within Data

Patterns within data may occur because of a definite arrangement or interrelation between the characters or strings of characters that span a data record, that is, because of the data's *structure*. Patterns may also occur within data because only relatively few of the possible characters or strings of characters tend to repeat, that is, because of the data's *redundancy*.

The structural relationship that may exist within data is illustrated by an example of several assembler language statements punched onto 80-column cards (Figure 2-13). When this plaintext is enciphered using DES (no chaining), patterns within the ciphertext are still discernible (Figure 2-14).

Similarly, data intended for visual display may also contain patterns because of a rigidly defined format. For example, a format for medical records might well include such displayed keywords as *name, age, height, weight*, and the like. These constant portions of the displayed data could allow its overall structure to be determined, even though enciphered. Once this underlying structure is known, variable portions of the data may be further exposed to analysis.

If data are highly redundant, then encryption with a block cipher may not prevent cryptanalysis using block frequency analysis. Block frequency analysis determines the frequency of each ciphertext block from a large sample of intercepted ciphertext. By relating the observed frequencies of the ciphertext blocks to the expected frequencies of the plaintext blocks, an opponent may be able to draw certain inferences concerning the nature of the plaintext corresponding to a given ciphertext.

Data redundancy can be exploited to attack a cryptographic system by the method illustrated in the following example. Assume that a cryptographic system uses a block cipher (no chaining) to protect messages transmitted among the nodes of a communication network. Assume further that each pair of nodes shares a different cipher key for messages transmitted between them.

At each node, the cipher keys are managed by the system and the user is not aware of the ciphering operations. Most importantly, the cipher keys are kept secret from users, even though users may request that messages be enciphered and deciphered using cipher keys.

```
IKJRDF01 CSECT ,                                                        00003000
@MAINENT DS    0H                                                       00004000
         USING *,@15                                                    00005000
         B     @PROLOG                                                  00006000
         DC    AL1(16)                                                  00007000
         DC    C'IKJRDF01  75.325'                                      00008000
         DROP  @15                                                      00009000
@PROLOG  STM   @14,@12,12(@13)                                          00010000
         BALR  @09,0                                                    00011000
@PSTART  DS    0H                                                       00012000
         USING @PSTART,@09                                              00013000
         L     @00,@SIZDATD                                             00014000
         GETMAIN  R,LV=(0)                                              00015000
         LR    @07,@01                                                  00016000
         USING @DATD,@07                                                00017000
         ST    @13,@SA00001+4                                           00018000
         LM    @00,@01,20(@13)                                          00019000
         ST    @07,8(,@13)                                              00020000
         LR    @13,@07                                                  00021000
         MVC   @PC00001(12),0(@01)                                      00022000
RBPRFXLN EQU 64                                                         00023000
RBPRFLNA EQU 32                                                         00024000
PRBLEN   EQU 136                                                        00025000
SIRBLEN  EQU 200                                                        00026000
TIRBLEN  EQU 136                                                        00027000
IRBLEN   EQU 128                                                        00028000
SVRBLEN  EQU 200                                                        00029000
*                                                 WAIT NO-TIME        */00030000
*                                                                       00031000
*    /*****************************************************************/00032000
*    /*                                                               */00033000
*    /* FIRST LETS MAKE SURE WE ARE NOT ALREADY ACTIVE                */00034000
*    /*                                                               */00035000
*    /*****************************************************************/00036000
*                                                                       00037000
```

Figure 2-13. Example of Highly Structured Plaintext

Spaces represent nonprintable characters

K = Hex '85CDCB1C9BD0851A' is the parity-adjusted key used for encipherment. Hexadecimal, or "Hex" for short, is a base-sixteen system for representing numbers. The numbers 0 through 15 are represented by digits 0 through 9, and letters A through F, respectively.

Figure 2-14. Ciphertext Obtained when Plaintext in Figure 2-13 is Encrypted Using the DES (No Chaining)

Although the cryptographic system described above protects users from outsiders, it does not necessarily protect one user from another. For example, a large amount of known plaintext could be transmitted between any two selected nodes by one of the system's users. This user could then recover his own ciphertext, if necessary, by performing a wiretap. A dictionary of equivalent plaintext and ciphertext blocks could then be constructed. This dictionary would permit the user to recover portions of intercepted ciphertext transmitted by another system's user between the same pair of communication nodes.

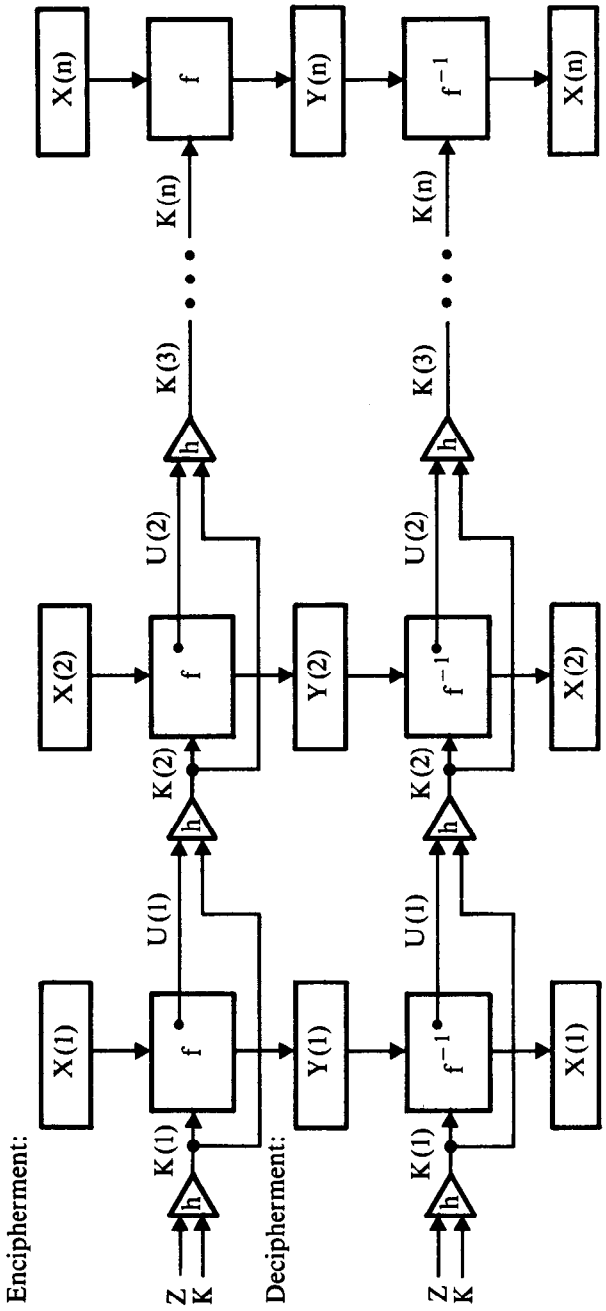
If the data normally transmitted in the communications network have enough redundancy, then the number of possible meaningful plaintext blocks will be small enough to permit a dictionary to be constructed. For example, 1 million different eight-character groups can be transmitted over a 4800 bit-per-second (baud) line in about four hours. And a dictionary of 1 million plaintext and ciphertext equivalents could easily be stored within most computer systems. Even with a dictionary of 1 million entries, it is likely that some blocks of intercepted ciphertext could be recovered directly from the dictionary. Once a few plaintext blocks have been correctly recovered, new suppositions concerning the content of adjacent blocks can be made. These hypotheses could be tested by transmitting additional blocks of plaintext and intercepting the corresponding ciphertext to determine if the suppositions were correct. Therefore, through a process of trial and error, it may be possible for additional portions of an intercepted message to be recovered.

One way to eliminate the undesirable effects of redundancy and structure within data is by Exclusive-ORing a different random or pseudo-random bit pattern Z with each block of plaintext prior to its encryption. In effect, the previously existing patterns within the data (should they occur) are canceled as a result of the noise vector Z.

However, if the values of Z (chaining values) are selected using a process that cannot be duplicated at the time decipherment takes place, then each chaining value must also be transmitted or stored with each block of ciphertext so that recovery of the plaintext is possible. This requirement is most disadvantageous since it causes the amount of information that must be transmitted or stored to be doubled.

Chaining eliminates the problem of transmitting or storing a separate Z-value for each block of ciphertext, since at each step in the ciphering process an equivalent chaining value is computed from information used within the ciphering process (such as prior plaintext, ciphertext, or key). The chaining value used at the first step in the ciphering process is called the *initial chaining value* or initializing vector (Z), and if it is to be used it must be supplied as input to the ciphering process.

In effect, the chaining value permits noise to be introduced into the ciphering process. The way in which this chaining value is derived and applied determines the type of chaining used. Several different block chaining techniques are discussed below.



U(1), U(2), ..., U(n) represent intermediate results that are identical for encipherment and decipherment.

Figure 2-15. Block Cipher with Block Chaining (Block Chaining Using Variable Key)

Block Chaining Using a Variable Key

One way to obtain block chaining is to change cipher keys internally at each step in the ciphering process. This could be accomplished by using a feedback from some intermediate value derived within the ciphering function (Figure 2-15).

It must be possible to derive the same intermediate value used for feedback during both encipherment and decipherment. For example, if DES were used, then such an intermediate value could be obtained after the eighth round of encipherment/decipherment. (See Chapter 3 for a discussion of the DES algorithm.)

The cryptographic function f defines the relationship between plaintext and ciphertext. Since the length of plaintext X equals the length of ciphertext Y, function f is one-to-one as well as onto. Function h defines how the cipher keys are changed or altered through the introduction of the initializing vector Z or the feedback vectors U(1), U(2), . . . , U(n − 1). Note that function h may be a many-to-one function since identical inputs to this function will be available during both encipherment and decipherment.

From Figure 2-15, it follows that

$$K(i) = h(K(i-1), U(i-1)); \qquad i \geqslant 1 \tag{2-15}$$

where

$$U(i) = \begin{cases} \text{an intermediate result of the ciphering} \\ \text{operation that is identical during both} \\ \text{enciphering and deciphering operations;} \end{cases} \qquad i > 0$$

$$U(0) \equiv Z$$

$$K(0) \equiv K$$

and ≡ denotes "identically equal to." Hence, encipherment and decipherment are expressed as

$$Y(i) = f_{K(i)}(X(i)); \qquad i \geqslant 1 \tag{2-16a}$$

and

$$X(i) = f^{-1}_{K(i)}(Y(i)); \qquad i \geqslant 1 \tag{2-16b}$$

respectively. Even if the initializing vector (Z) is held constant, patterns in the input data will be eliminated. This is because cipher key K(i) is different from cipher key K(j) so that ciphertext Y(i) is different from ciphertext Y(j), even if plaintext X(i) equals plaintext X(j). In contrast, *stereotyped messages* (such as may occur in a terminal-to-computer inquiry system where frequent yes and no responses are transmitted) are not masked when Z is constant, since identical messages will always result in identical cryptograms.

To eliminate the problem of stereotyped data records, a variable initializing vector (Z) must be used.

Since U(i) is an intermediate result of the encipherment of block X(i), it can be expressed as a function h_1 of K(i) and X(i):

$$U(i) = h_1(K(i), X(i)); \quad i \geqslant 1 \tag{2-17a}$$

Similarly, U(i) is an intermediate result of the decipherment of block Y(i), and so it can also be expressed as a function h_2 of K(i) and Y(i):

$$U(i) = h_2(K(i), Y(i)); \quad i \geqslant 1 \tag{2-17b}$$

But, by the recursive nature of Equations 2-15 and 2-17a, it follows that there exist functions $\phi_1, \phi_2, \ldots, \phi_i$ such that

$$K(i) = \phi_i(K, X(0), X(1), \ldots, X(i-1)); \quad i \geqslant 1 \tag{2-18a}$$

where $X(0) \equiv Z$. Likewise from Equations 2-15 and 2-17b, it follows that there exist functions $\psi_1, \psi_2, \ldots, \psi_i$ such that

$$K(i) = \psi_i(K, Y(0), Y(1), \ldots, Y(i-1)); \quad i \geqslant 1 \tag{2-18b}$$

where $Y(0) \equiv Z$.

But from Equations 2-16a and 2-18a, it follows that there exist functions $H_1, H_2, \ldots, H_i$ such that the generated ciphertext, Y(i), is given by

$$Y(i) = H_i(K, X(0), X(1), \ldots, X(i)); \quad i \geqslant 1 \tag{2-19a}$$

and from Equations 2-16b and 2-17b, it follows that there exist functions $G_1, G_2, \ldots, G_i$ such that the recovered plaintext, X(i), is given by

$$X(i) = G_i(K, Y(0), Y(1), \ldots, Y(i)); \quad i \geqslant 1 \tag{2-19b}$$

where $X(0) \equiv Y(0) \equiv Z$.

Equation 2-19a enables us to determine the most general block cipher. Since the ciphering process is entirely deterministic, an output ciphertext block at time i, Y(i), can depend only on the inputs to the ciphering process from time 1 through time i, namely the cipher key (K), the initializing vector (Z), and all plaintext blocks X(1) through X(i). It follows, therefore, that Equation 2-19a represents the most general relation that could be established for a block cipher. Moreover, since ciphertext block Y(i) depends on the initial conditions established at the beginning of the ciphering process, namely at time 1, it is said that Y(i) is *origin-dependent*.

For similar reasons, it follows that a recovered plaintext block at time i, X(i), can depend only on the cipher key (K), the initializing vector (Z), and all ciphertext blocks Y(1) through Y(i). Equation 2-19b, therefore, represents the most general relation that could be established for a block cipher. In like manner, X(i) is also origin-dependent.

A block cipher which satisfies the general relations expressed in Equations 2-19a and 2-19b is defined as a *general block cipher*. A block cipher for which every bit in the recovered plaintext block X(i) is a function of every bit in ciphertext blocks Y(1) through Y(i) is said to have the property of *error propagation*. Since the corruption of only a single bit of ciphertext may cause each subsequent bit of recovered plaintext to be in error, error propagation can be used as a means for detecting the occurrence of such errors (see Cryptographic Message Authentication Using Chaining Techniques).

Since strong intersymbol dependence is one property of a block cipher, it follows that error propagation is automatically achieved in a general block cipher. However, since a bit in output block (i) does not depend on bits within input blocks (i + 1), (i + 2), . . . , the dependence is not defined as strong intersymbol dependence but rather as *intersymbol dependence*.

Block Chaining Using Plaintext and Ciphertext Feedback

Another way to obtain block chaining is to hold the cipher key constant and modify the input plaintext by making it a function of both the previous block of plaintext and the previous block of ciphertext (Figure 2-16). In this case, encipherment and decipherment are given by

$$Y(i) = f_K(X(i) \oplus U(i)); \quad 1 \geqslant 1 \tag{2-20a}$$

and

$$X(i) = f_K^{-1}(Y(i)) \oplus U(i); \quad i \geqslant 1 \tag{2-20b}$$

respectively, where

$$U(i) = \begin{cases} Z; & i = 1 \\ h(X(i-1), Y(i-1)); & i > 1 \end{cases} \tag{2-21}$$

Suppose that h is simple addition modulo 2^{64}.

$$U(i) = X(i-1) + Y(i-1) \bmod 2^{64}; \quad i > 1$$

Then, from Equation 2-20a, it follows that

$$Y(i) = f_K(X(i) \oplus (X(i-1) + Y(i-1) \bmod 2^{64})); \quad i > 1$$

and so, there exist functions $H_1, H_2, \ldots, H_i$ such that

$$Y(i) = H_i(K, X(0), X(1), \ldots, X(i)); \quad i \geqslant 1 \tag{2-22a}$$

where $X(0) \equiv Z$. Similarly, from Equation 2-20b, it follows that

$$X(i) = \begin{cases} f_K^{-1}(Y(i)) \oplus Z; & i = 1 \\ f_K^{-1}(Y(i)) \oplus (Y(i-1) + X(i-1) \bmod 2^{64}) & i > 1 \end{cases}$$

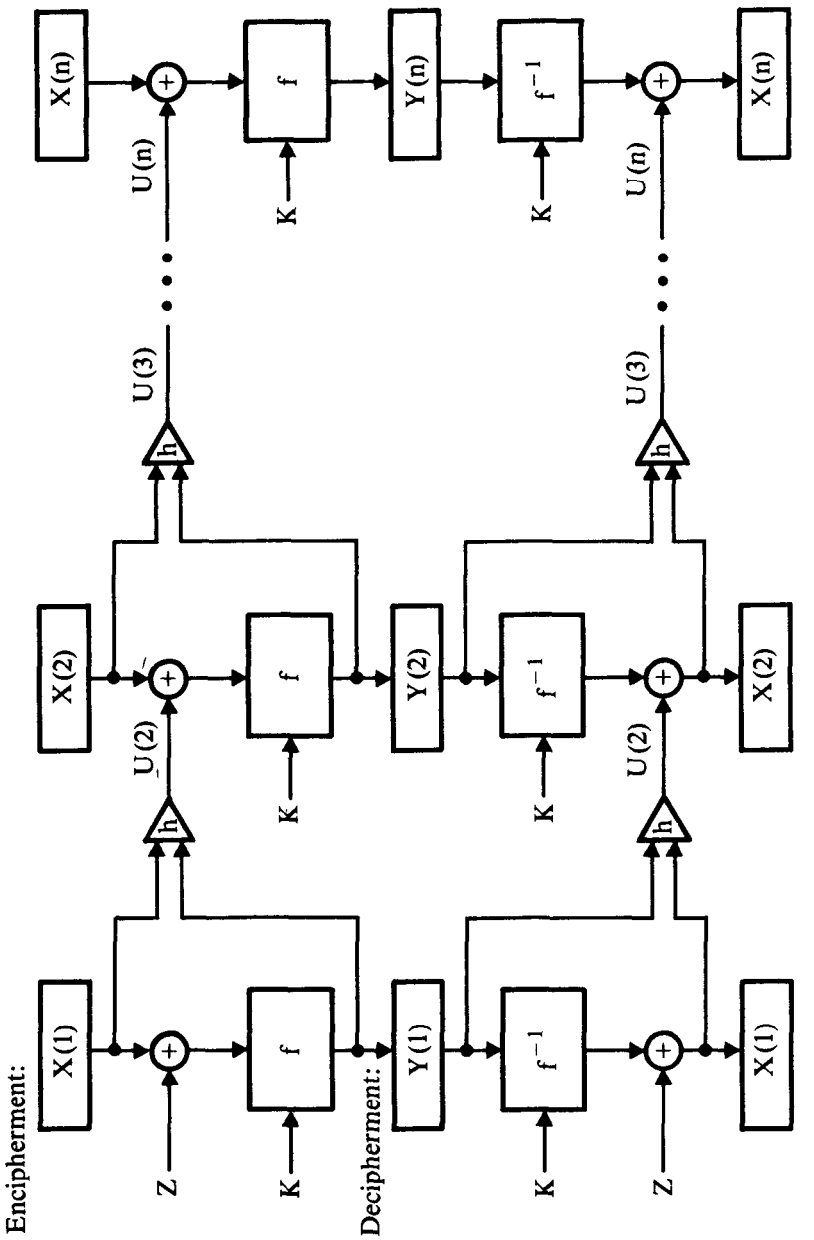


Figure 2-16. Block Cipher with Block Chaining (Block Chaining Using Plaintext-Ciphertext Feedback)

and so, there exist functions $G_1, G_2, \ldots, G_i$ such that

$$X(i) = G_i(K, Y(0), Y(1), \ldots, Y(i)); \quad i \geqslant 1 \qquad (2\text{-}22b)$$

where $Y(0) \equiv Z$. From Equations 2-22a and 2-22b, it can be seen that origin-dependence has been achieved for X(i) and Y(i), and that Equations 2-19a and 2-19b have been satisfied for the general block cipher.

A Self-Synchronizing Scheme Using Ciphertext Feedback

A cryptographic procedure or device is said to be *self-synchronizing* if after an error has occurred the ciphering operation automatically corrects itself (i.e., all plaintext can be recovered correctly except the portion affected by the error). Consider the case of two cryptographic devices that produce identical outputs for identical inputs. Suppose that an error is now introduced into the ciphering process of one device, so that the outputs of the two devices are different. If after some period of time the outputs again become equal, then the devices are said to be self-synchronizing.

A self-synchronizing block chaining scheme can be obtained by omitting the plaintext feedback in Figure 2-16 (referred to as Cipher Block Chaining, CBC [26]). Mathematically, this can be expressed by defining function h in Equation 2-21 as follows.

$$h(X(i-1), Y(i-1)) = Y(i-1); \quad i \geqslant 1$$

Hence, encipherment and decipherment can be expressed by

$$Y(i) = f_K(X(i) \oplus Y(i-1)); \quad i \geqslant 1$$

and

$$X(i) = f_K^{-1}(Y(i)) \oplus Y(i-1); \quad i \geqslant 1$$

where $X(0) \equiv Y(0) \equiv Z$ (see Figure 2-17).

Again, it follows that there exist functions $H_1, H_2, \ldots, H_i$ and $G_1, G_2, \ldots, G_i$ such that

$$Y(i) = H_i(K, X(0), X(1), \ldots, X(i)); \quad i \geqslant 1 \qquad (2\text{-}22c)$$

and

$$X(i) = G_i(K, Y(i-1), Y(i)); \quad i \geqslant 1 \qquad (2\text{-}22d)$$

From Equations 2-22c and 2-22d, it follows that patterns within the input data are masked since ciphertext block Y(i) depends on plaintext blocks X(1), X(2), . . . , X(i). However, since the recovered plaintext block X(i) does not depend on all ciphertext blocks Y(1), Y(2), . . . , Y(i), the scheme is not a general block cipher.

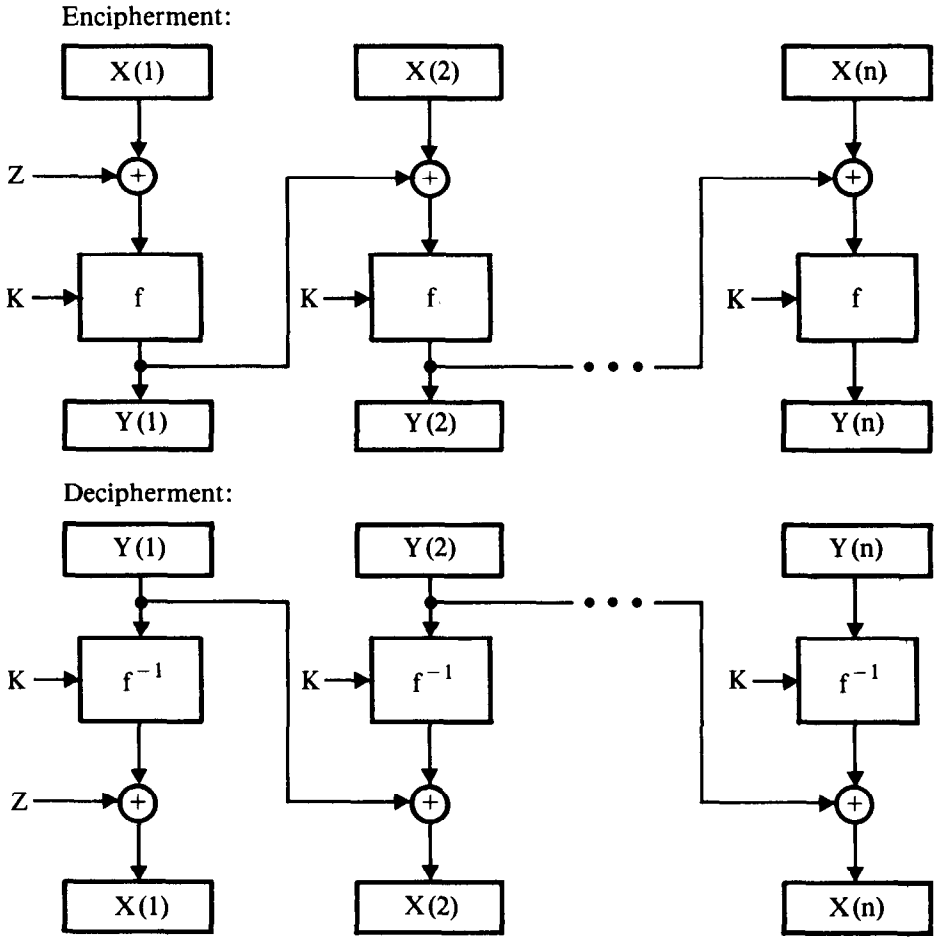


Figure 2-17. Block Cipher with Block Chaining
(Block Chaining Using Ciphertext Feedback)

An error occurring in ciphertext block Y(i − 1) can affect every bit in the recovered plaintext block X(i − 1), but it will affect only the corresponding bit positions in the recovered plaintext block X(i). In other words, if the seventh and thirteenth bits in Y(i − 1) are in error, then the seventh and thirteenth bits in X(i) are in error. None of the bits in the recovered plaintext blocks X(i + 1), X(i + 2), and so forth, will be affected by an error occurring in ciphertext block Y(i − 1). Since most of the plaintext can be recovered, even when an error occurs in the ciphertext, the scheme is said to be self-synchronizing.

However, since an error in ciphertext does not propagate, the scheme cannot be directly used for message authentication (see Cryptographic Message Authentication Using Chaining Techniques). Hence the choice of a block chaining method must involve the weighing of the benefit of direct cryptographic authentication against that of self-synchronization.

A practical application for the self-synchronizing approach is the protection of stored data. When cryptography is used for communication security, one can recover from an error in transmission simply by retransmitting the original message. When a file is encrypted, recovery from an error must be effected with ciphertext alone. If a ciphering procedure with error propagation is used for file security, subsequent inability to read a portion of the ciphertext, because of damage either to the physical medium or to the recorded bits, may prevent all following ciphertext from being deciphered. In certain applications for cryptography, therefore, a self-synchronizing approach may be the most desirable.

Examples of Block Chaining

To illustrate how block chaining can be used to eliminate patterns within data, the plaintext in Figure 2-13 was enciphered using block chaining with ciphertext feedback (Figure 2-17). Figure 2-18 illustrates the situation where each 80-character line or 80-column card is enciphered as a separate data record so that only the blocks within each line or card are chained together. Figure 2-19 illustrates the case where the entire text is enciphered as a single data record so that all blocks are chained together.

Short Block Encryption

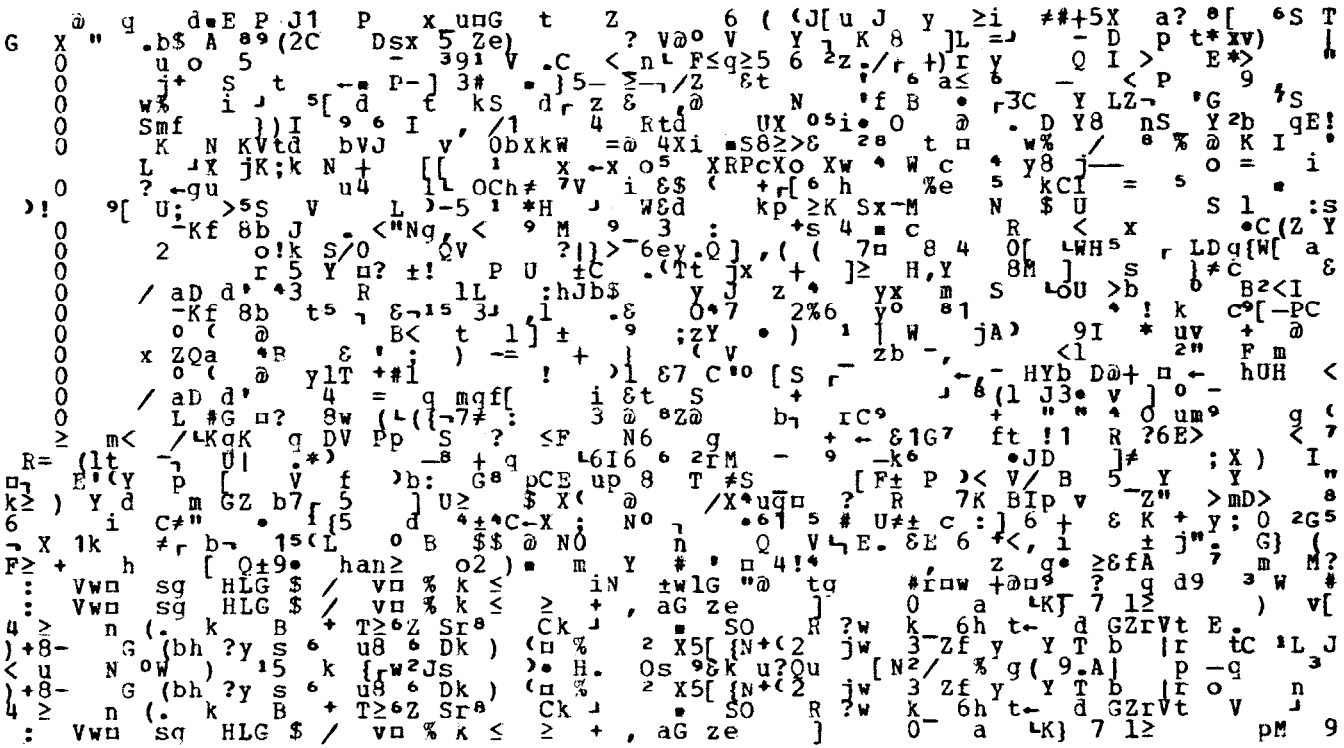
Since a block cipher enciphers and deciphers only blocks of bits at a time, it is important to know how a block cipher can cope with data whose length is not an integral multiple of the cipher's blocksize. A block whose length is less than the cipher's blocksize is called a *short block*, whereas a block whose length is equal to the blocksize is called a *block* or *standard block*.

A short block will always occur as the last block of data when the data's length is not an integral multiple of the cipher's blocksize. A short block will also occur as the first (and only) block of data when the data's length is less than the cipher's blocksize.

If a short block is first padded with enough additional bits to produce a standard block, it is always possible to encipher a short block in a secure way using a block cipher (see Effects of Padding and Initializing Vectors). *Padding* is the operation of appending additional data bits (or bytes) to plaintext so that its length becomes a multiple of the cipher's blocksize. For security purposes, it is best if pad characters are produced by a random process, although in most cases a pseudo-random process is sufficient. If pad characters could be predicted by an opponent, then, in terms of the work factor, the blocksize would be effectively reduced. The technique of short block encryption using padding is illustrated below (Figure 2-20).

Generally, when cryptography is used for communication security, padding is an acceptable solution for handling messages that may be variable in length. This, however, is not always the case when cryptography is used for file security, because padding bits may cause overflow of secondary storage.

When ciphering operations are not length-preserving, it may no longer be convenient, practical, or even possible to substitute ciphertext freely for

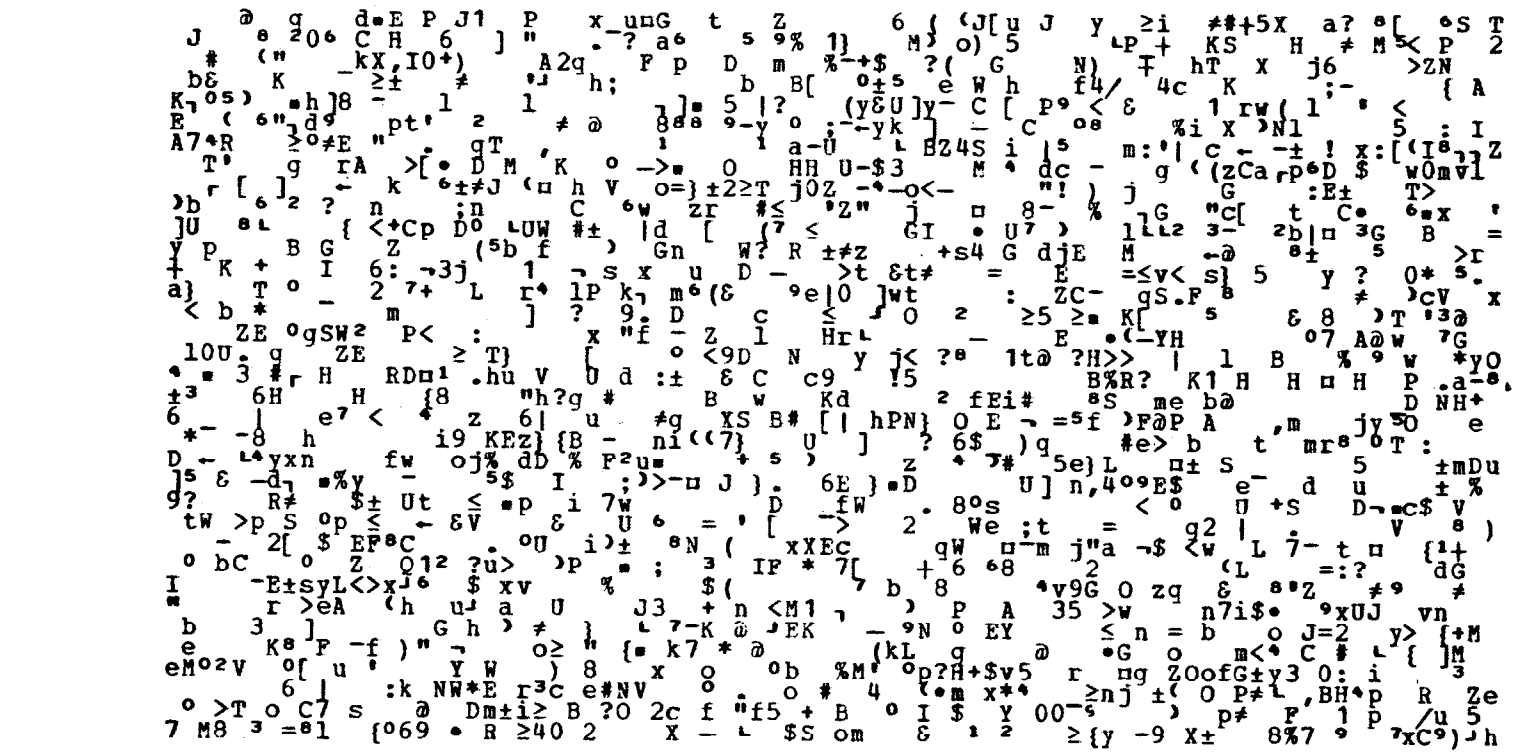


Spaces represent nonprintable characters.

Z = Hex '5555555555555555' is the constant initializing vector, which is the same for each line of plaintext to be encrypted.

K = Hex '85CDCB1C9BD0851A' is the parity-adjusted key used for encipherment.

Figure 2-18. Ciphertext Obtained when the Plaintext in Figure 2-13 is Encrypted Line-By-Line Using the DES Block Cipher with Ciphertext Feedback



Spaces represent nonprintable characters.

Z = Hex ' 5555555555555555 ' is the initializing vector.

K = Hex ' 85CDCB1C9BD0851A ' is the parity-adjusted key used for encipherment.

Figure 2-19. Ciphertext Obtained when the Plaintext in Figure 2-13 is Encrypted as a Single Aggregate Message Using the DES Block Cipher with Ciphertext Feedback

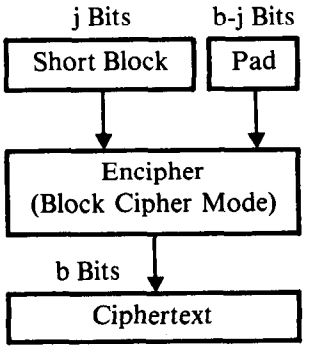


Figure 2-20. Encipherment of a Short Data Block Using Block Cipher Mode

plaintext within a computer data base. Expanded ciphertext may cause a file to overflow the physical boundaries of the recording medium. Encipherment of selected fields within records, or of selected records within files may require that record formats be redefined and may in turn require existing files to be restructured. Such dependencies between the encryption algorithm and stored data are undesirable.

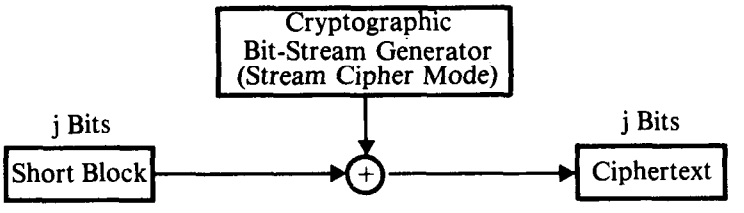
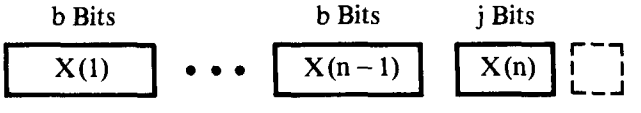
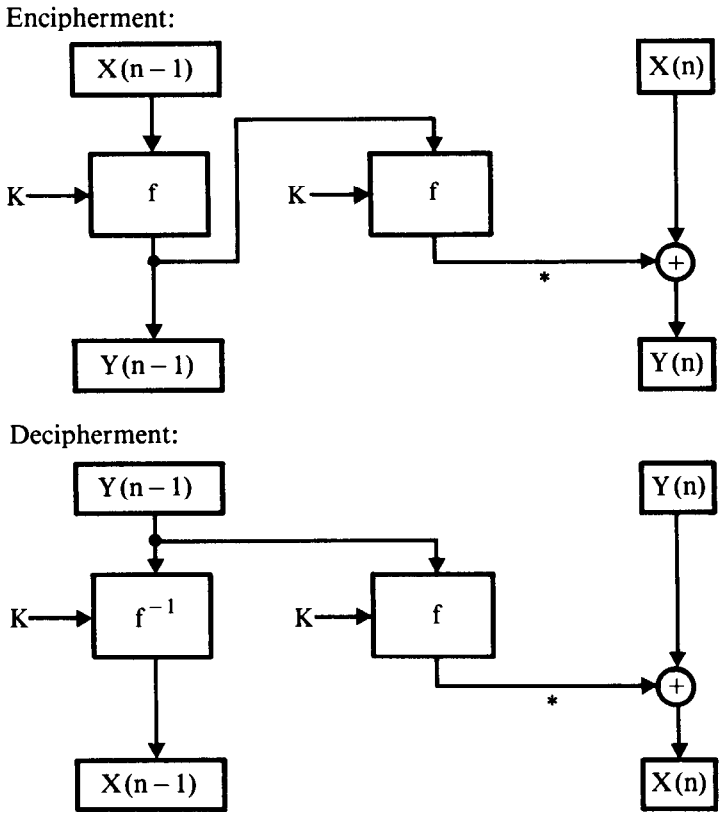


Figure 2-21. Encipherment of a Short Data Block Using Stream Cipher Mode

One way to avoid data expansion would be to use the stream cipher mode of operation to handle the special situations of short blocks (Figure 2-21). In this mixed mode of operation, the block cipher mode is used for ciphering standard blocks and the stream cipher mode is used for ciphering short blocks. One way to implement the stream cipher mode is to generate the cryptographic bit-stream by reenciphering the previous block of ciphertext or, in the case of the first block, by enciphering the initializing vector. This scheme is shown in Figure 2-22.

Consider the following plaintext whose length is greater than, but not an integral multiple of, the cipher's blocksize.





* | X(n) | bits are used to encipher X(n), where | X(n) | is the length of X(n) in bits. If n = 1, then the initializing vector (Z) is enciphered under K to produce the cryptographic bit-stream.

Figure 2-22. Stream Cipher Mode for Encipherment of Short Blocks

where

$$b = \text{blocksize}$$

$$1 \leqslant j < b$$

Instead of using a mixed mode of operation (i.e., a block cipher for standard blocks and a stream cipher for short blocks) one can use a block cipher to encipher short blocks provided that the data's length is greater than the cipher's blocksize.

Another approach for enciphering a short block, X(n), is to make use of a technique called *ciphertext-stealing mode* (Figure 2-23). In this mode, the short block, X(n), is first padded by stealing (removing) just enough bits from the ciphertext Y(n − 1) to make the length of X(n) equal to the cipher's blocksize. This results in Y(n − 1) becoming a short block and Y(n)

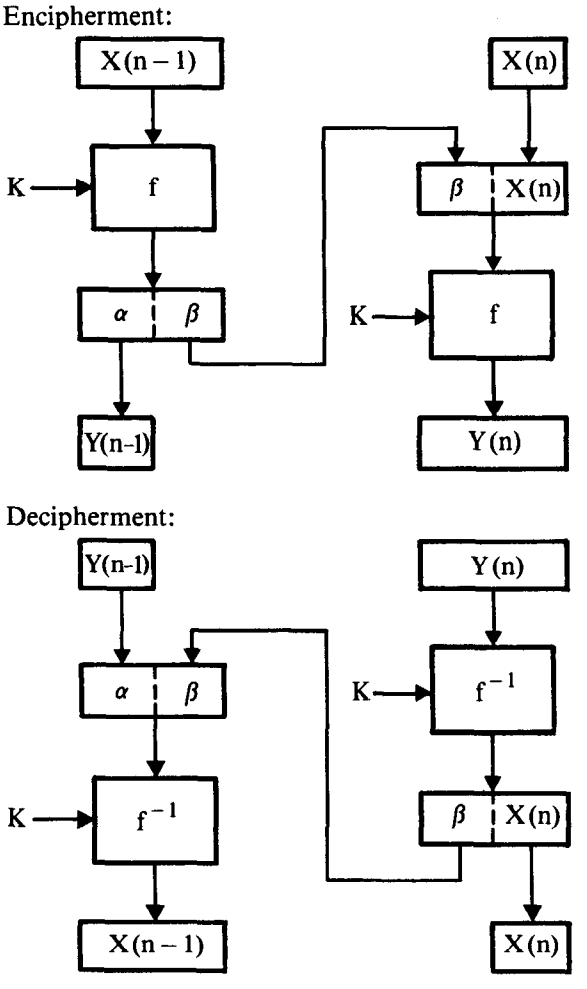


Figure 2-23. Ciphertext-Stealing Mode for Encipherment of Short Blocks

becoming a standard block. Since the number of bits removed from Y(n − 1) equals the number of bits added to X(n), no expansion occurs. The enciphering process is reversed by deciphering Y(n) prior to Y(n − 1) and recovering the original stolen bits from Y(n − 1). The reconstructed value of Y(n − 1) is then deciphered.

Both the stream cipher mode and the ciphertext-stealing mode display a certain awkwardness in the manner in which short blocks are handled. With the stream cipher mode, the cryptographic bit-stream is generated by encipherment of Y(n − 1) regardless of whether encipherment or decipherment is taking place. Complete symmetry between encipherment and decipherment is therefore lost. With the ciphertext-stealing mode, the serial fashion in which blocks are normally enciphered or deciphered is not preserved. Here,

the two trailing ciphertext blocks are deciphered in reverse order. Again, complete symmetry between encipherment and decipherment is lost.

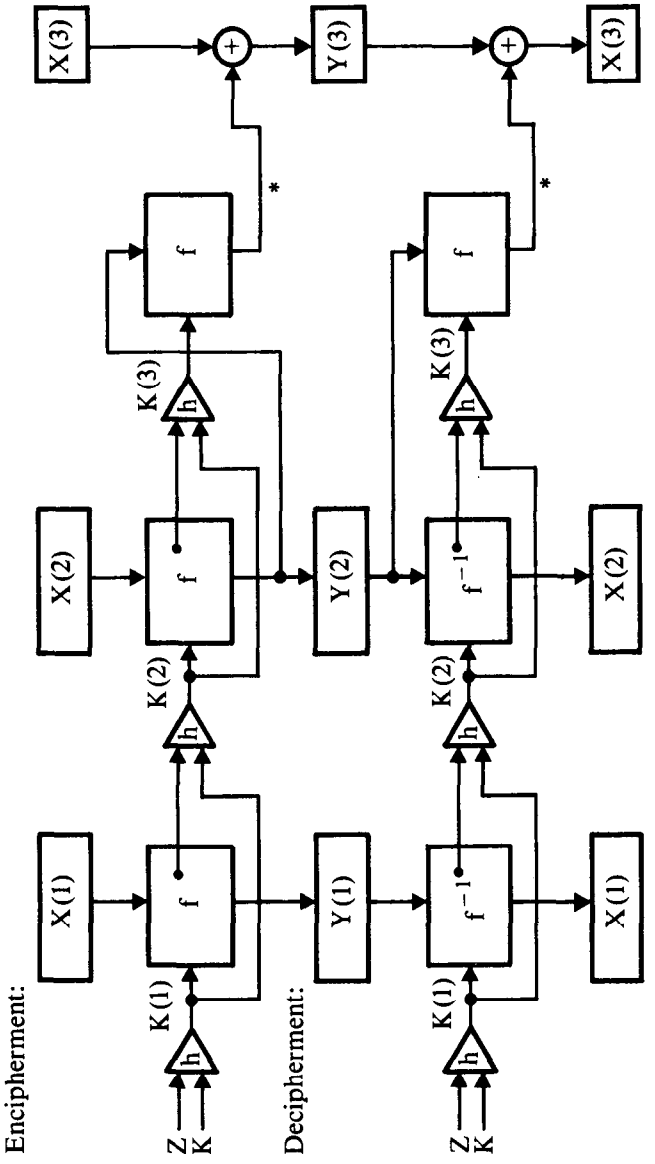
The encryption of short blocks also affects error propagation. In the stream cipher mode (Figure 2-22), a bit change in Y(n − 1) will affect all bits in the recovered plaintext, (X(n − 1), X(n)), whereas a bit change in Y(n) will cause only a corresponding bit change in the recovered plaintext, X(n). Hence the error propagation property discussed earlier for the general block cipher is lost as far as the last short block of ciphertext is concerned. (There is no strong intersymbol dependence between plaintext and ciphertext in the last block.) In the ciphertext-stealing mode (Figure 2-23), any bit change in the short ciphertext block Y(n − 1) will affect only the recovery of plaintext block X(n − 1), but will not affect the recovery of plaintext short block X(n). Hence, any error in ciphertext block Y(n − 1) will not propagate, and again the error propagation property is affected.

When implemented properly, the stream cipher and ciphertext-stealing modes provide equivalent cryptographic strength, although a somewhat unlikely set of circumstances can be found in which these two techniques are not equivalent. Suppose that these two techniques are implemented in a cryptographic system which uses a block cipher with no chaining. Assume further that it is possible for an opponent to request enciphering operations but not deciphering operations, and that the cipher keys are managed by the system (i.e., unknown to the system's users). If the stream cipher mode is used, the short block X(n) can be recovered by intercepting Y(n − 1) and Y(n) via a wiretap, retransmitting Y(n − 1) as text in a second message, intercepting the ciphered version of Y(n − 1) via a second wiretap, and finally, Exclusive-ORing Y(n) with the ciphered version of Y(n − 1). This attack could be prevented either by using $X(n-1) + Y(n-1) \bmod 2^{64}$ instead of Y(n − 1) as the value to be ciphered, or by using chaining.

Figures 2-24 through 2-27 illustrate how the stream cipher and ciphertext-stealing modes can be used in conjunction with the block chaining schemes previously discussed. Without loss of generality, only two full blocks are shown, X(1) and X(2), respectively. Block X(3) is a short block. Generally speaking, all of these schemes are equivalent in cryptographic strength provided that the basic cryptographic algorithm is strong (i.e., an algorithm comparable in strength to DES is used).

In each case (Figure 2-24 and 2-26), the stream cipher mode is implemented in such a way that it is not possible for an opponent to recover X(3) by intercepting Y(2) and Y(3) via a wiretap, retransmitting Y(2) as data, intercepting the encipherment of Y(2) via a second wiretap, and finally Exclusive-ORing the enciphered version of Y(2) with Y(3). In Figure 2-24, the attack is not possible because Y(2) is enciphered with a variable key that is chained back to the origin. In Figure 2-26, the attack is not possible because the cryptographic bit-stream used to encipher X(3) via the Exclusive-OR operation is a function of both plaintext and ciphertext.

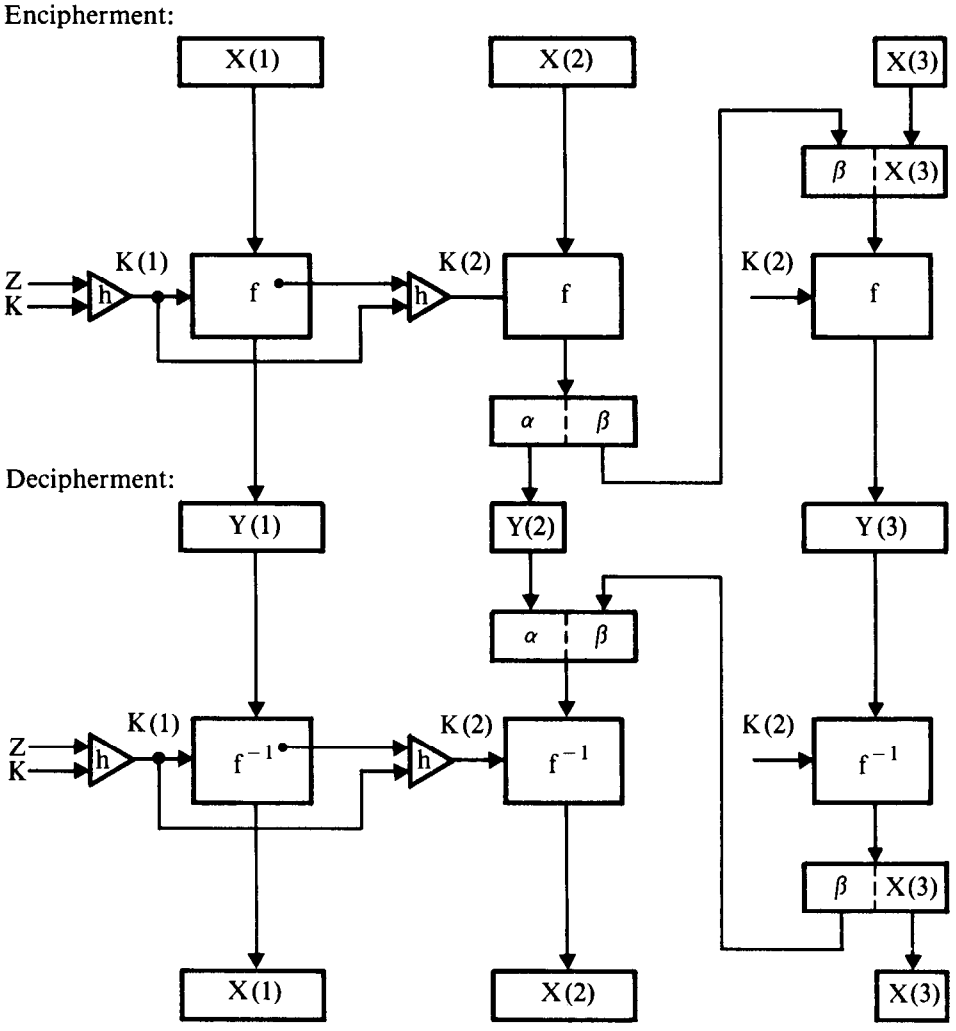
For all practical purposes, the ciphertext-stealing mode is implemented in such a way that a frequency analysis on the short block is not possible. In Figure 2-25, observe that the variable key K(2) used to encipher X(2) is the same key used to encipher the quantity β concatenated with X(3), and that



* Number of bits used equals | X(3) |

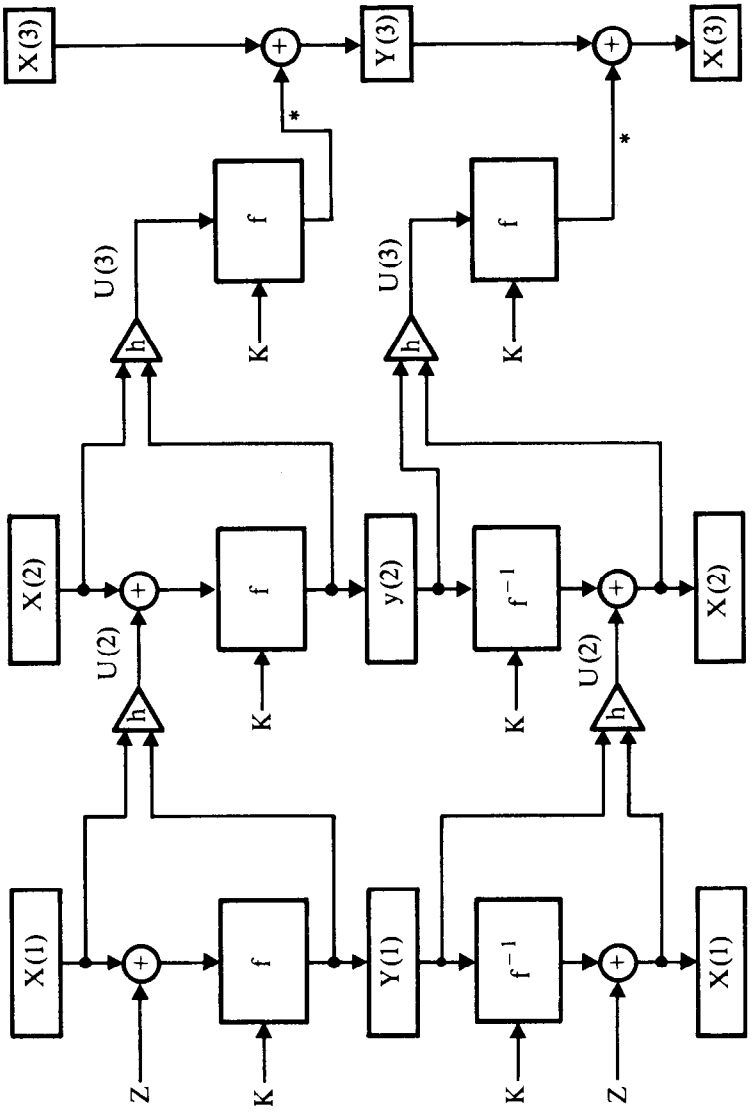
Note: Function h could be an Exclusive-OR operation in an actual implementation. A bit change in Y(3) results only in a corresponding bit change in X(3) and hence is predictable by an opponent. A change of any other ciphertext bit is propagated, and the effect on the recovered plaintext is unpredictable.

Figure 2-24. Block Cipher with Block Chaining
(Block Chaining Using Variable Key, and Stream Cipher Mode for Short Blocks)



Note: Function h could be an Exclusive-OR operation in an actual implementation. A change in Y(2) affects only the recovery of X(2), and hence the error does not propagate in that case. A change of any other ciphertext bit is propagated, and the effect on the recovered plaintext is unpredictable.

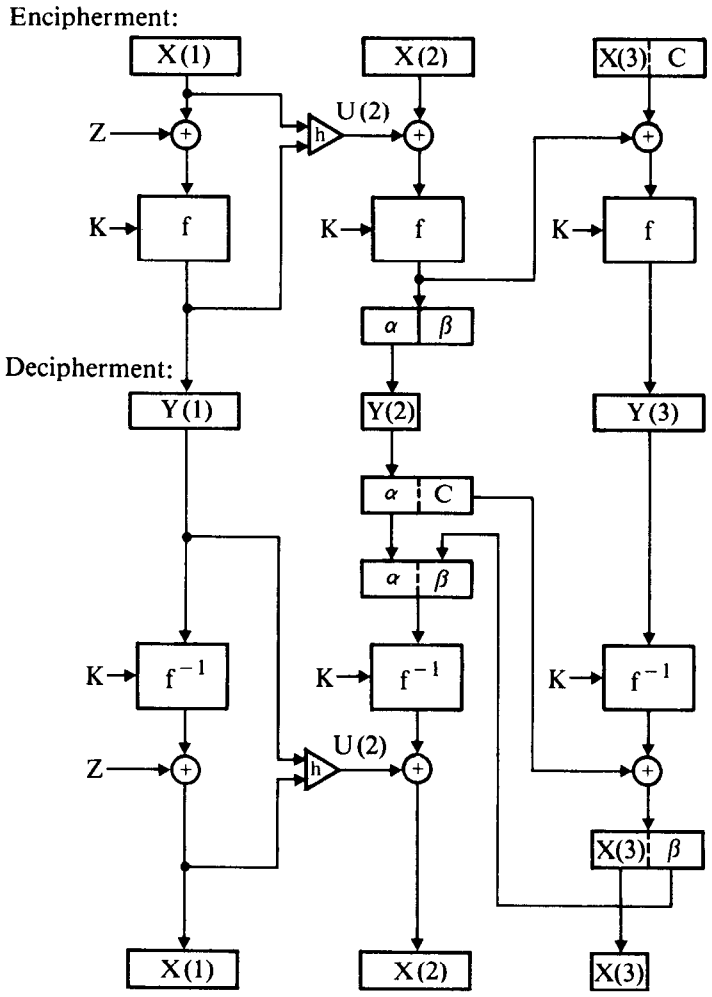
Figure 2-25. Block Cipher with Block Chaining (Block Chaining Using Variable Key, and Ciphertext-Stealing Mode for Short Blocks)



* Number of bits used equals | X(3) |

Note: Function h could be addition modulo 2^{64} in an actual implementation. A bit change in Y(3) results only in a corresponding bit change in X(3) and hence is predictable by an opponent. A change of any other ciphertext bit is propagated, and the effect on the recovered plaintext is unpredictable.

Figure 2-26. Block Cipher with Block Chaining (Block Chaining Using Plaintext-Ciphertext Feedback, and Stream Cipher Mode for Short Blocks)



Note: (1) C is a constant, say all zero bits.

(2) Function h could be addition modulo 2^{64} in an actual implementation; or a self-synchronizing system could be obtained by making h depend only on the ciphertext feedback.
A change in Y(2) affects only the recovery of X(2), and hence the error does not propagate in that case. A change of any other ciphertext bit is propagated, and the effect on the recovered plaintext is unpredictable.

Figure 2-27. Block Cipher with Block Chaining (Block Chaining Using Plaintext-Ciphertext Feedback and Ciphertext-Stealing Mode for Short Blocks)

K(2) is chained to the origin. Since X(2) is enciphered prior to X(3) and Y(3) is deciphered prior to Y(2), the same cipher key, K(2), is used for both the encipherment and decipherment of the second and third blocks of plaintext and ciphertext, respectively. In Figure 2-27, observe that the short block is chained using only a ciphertext feedback. Since the last two blocks must be treated in reverse order, using a feedback from the plaintext would prevent recovery.

Observe that the method for implementing the ciphertext-stealing mode in Figure 2-27 is slightly different from that shown in Figure 2-23. In addition to stealing ciphertext and concatenating it with short block X(3), ciphertext is also stolen and Exclusive-ORed with X(3). This extra step is important because for all practical purposes it prevents a successful block frequency analysis on Y(3). With the technique shown in Figure 2-23, this is not necessarily true when the number of bits concatenated to X(n) is small, (e.g., if β were only one bit and the number of plaintext combinations for X(n) were small).

As pointed out in Figure 2-22, the encipherment of data consisting of a single short block of $j < b$ bits can be accomplished by Exclusive-ORing the first j bits of the encipherment of the initializing vector Z. However, such a procedure would be weak if the same Z were used repeatedly to encipher a sequence of short blocks, e.g., a file of records in which each record consists of a single short block. In effect, each short block would be protected using the same cryptographic bit-stream, which would thus allow the plaintext associated with these short blocks to be recovered via a simple cryptanalysis.

In the case of file security, a strong procedure for the encipherment of data records consisting of repeated short blocks is to use a method of *record chaining* [24,25]. Here, the value of Z is a variable that changes for each record to be enciphered. In the recommended approach, Z_i (the initializing vector for the ith record) is specified as the rightmost 64 bits ("right64") of the concatenation (||) of Z_{i-1} and the just-constructed ciphertext (the ciphertext of record $i - 1$), i.e.,

$$Z_i = \text{right64}[Z_{i-1} \,||\, \text{ciphertext of record } i - 1]$$

Thus, the first record is enciphered using the initial value of Z (defined as Z_1). All subsequent records are enciphered using a computed value of Z, as described above.

With record chaining, the chaining process continues across record boundaries. To correctly decipher a record (given the key), only that record and the preceding 64 bits of ciphertext (and possibly the initial value of Z if less than 64 bits of ciphertext are present) are required. Record chaining is ideally suited for sequentially organized files. For nonsequential files, block chaining is best suited. But for block chaining to be strong, each record must have its own (unique or randomly selected) 64-bit initializing vector, Z.

The inherent cryptographic weaknesses associated with the encryption of groups of short data blocks in a sequentially organized file can also be avoided via the implementation. For example, data blocks (of any length)

can be temporarily joined (concatenated) into an "artificial" data unit (or *cipher unit*) which is then enciphered as if it were one, large record or collection of data.

In the case of communication security, padding is the preferred technique for short block encryption. Here, the physical boundary limitations that can lead to data overflow, which apply to file security, do not exist.

STREAM CIPHERS WITH CHAINING

In a block cipher, chaining can be used to acquire two important properties. First, it can mask repetitive patterns within data by making each block of ciphertext, Y(i), dependent upon all prior blocks of plaintext, X(1), X(2), . . . , X(i − 1), as well as on the present plaintext block X(i). In a sense, this chaining technique extends the effective blocksize of the cipher. Second, it can extend error propagation across block boundaries by making each block of recovered plaintext, X(i), dependent upon all prior blocks of ciphertext, Y(1), Y(2), . . . , Y(i − 1), as well as on the present ciphertext block, Y(i).

In a stream cipher, patterns occurring within the input plaintext are automatically eliminated as a consequence of Exclusive-ORing the plaintext with the cryptographic bit-stream (Figure 2-10). The cryptographic bit-stream introduces pseudo-random noise into the ciphering process and hence eliminates exploitable statistics associated with the plaintext. The changing initializing vector Z assures that stereotyped messages (if they occur) will result in different ciphertext. Thus chaining is not needed in a stream cipher to mask patterns within the data or to mask stereotyped messages. It can, however, be useful in a stream cipher to achieve either the property of error propagation, if one desires secrecy and authentication in one operation, or self-synchronization, in which case the system does not have to be reinitialized after an error condition occurs.

In the stream cipher, it can be assumed that the cryptographic bit-stream is produced as a series of blocks:

$$R(1), R(2), \ldots, R(t)$$

where

$$R(i) = (r_1(i), r_2(i), \ldots, r_b(i))$$

is a block of b bits generated at iteration i, and b is the blocksize. Encipherment and decipherment are defined as

$$Y(i) = X(i) \oplus R(i); \quad i \geqslant 1 \tag{2-23a}$$

and

$$X(i) = Y(i) \oplus R(i); \quad i \geqslant 1 \tag{2-23b}$$

A Chaining Method with the Property of Error Propagation

Error propagation is present in a block cipher whenever each bit in the recovered plaintext block X(i) is a function of every bit in ciphertext blocks Y(1) through Y(i). In a stream cipher, however, because of the modulo 2 addition shown in Equation 2-23b, the jth bit in the recovered plaintext block X(i) depends on the jth bit in the ciphertext block Y(i), but not on any other bits in ciphertext block Y(i). At best, a scheme could be devised where the jth bit in X(i) is a function of every bit in Y(1) through Y(i − 1). If this were the case, then an error occurring in any of the ciphertext blocks Y(1) through Y(i − 1) could propagate to the recovered plaintext block X(i).

To achieve this dependence, a feedback could be provided from either the plaintext X, the initializing vector Z, or a combination of both, in addition to the feedback from the ciphertext. A stream cipher with the property of error propagation is shown in Figure 2-28.

Note, however, that error propagation due to corruption of the ciphertext could be obtained by providing a feedback only from the plaintext. That is, encipherment is expressed by

$$\begin{aligned} Y(1) &= X(1) \oplus g_K(Z); \quad i = 1 \\ Y(i) &= X(i) \oplus g_K(X(i-1)); \quad i > 1 \end{aligned}$$

and decipherment is expressed by

$$\begin{aligned} X(1) &= Y(1) \oplus g_K(Z); \quad 1 = 1 \\ X(i) &= Y(i) \oplus g_K(X(i-1)); \quad i > 1 \end{aligned} \tag{2-24}$$

It follows that each bit in the recovered plaintext block X(i) depends on each bit in the initializing vector (Z) and on each bit in the ciphertext blocks Y(1) through Y(i − 1), by the recursive relation shown in Equation 2-24.

If a feedback from plaintext were used, patterns in the plaintext would result in patterns in the ciphertext. This is because Y(i) is not origin-dependent. Recall that patterns were destroyed in the key auto-key cipher (Figure 2-12) because the feedback was taken from the cryptographic bit-stream.

From Figure 2-28 it follows that encipherment and decipherment can be expressed as

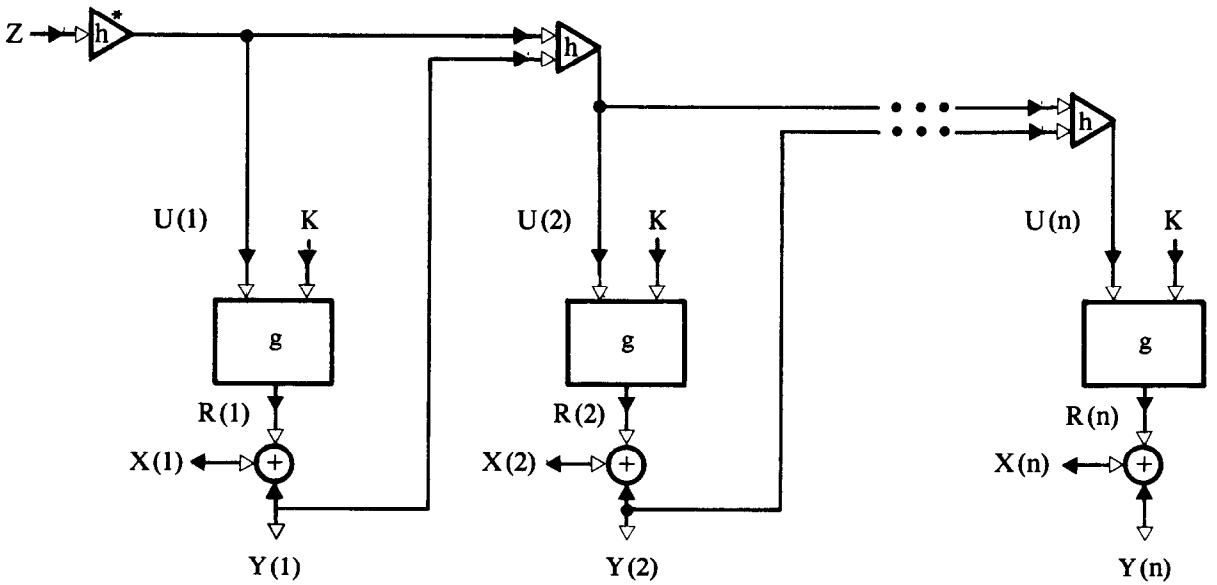
$$Y(i) = X(i) \oplus g_K(U(i)); \quad i \geqslant 1 \tag{2-25a}$$

and

$$X(i) = Y(i) \oplus g_K(U(i)); \quad 1 \geqslant 1 \tag{2-25b}$$

respectively, where

$$U(i) = \begin{cases} h^*(Z); & i = 1 \\ h(U(i-1), Y(i-1)); & i > 1 \end{cases} \tag{2-26}$$



Legend: encipherment mode: ⟶▷ , decipherment mode: ⟶

Note: Z, U, R, X and Y are blocks of n bits; h could be an Exclusive-OR function

Figure 2-28. Stream Cipher with Error Propagation

Function h* is again introduced to allow Z to be different in length from U.

From the recursive nature of Equation 2-26, it follows that there exist functions $G_1, G_2, \ldots, G_i$ and $H_1, H_2, \ldots, H_i$ such that

$$Y(i) = X(i) \oplus H_i(K, X(0), X(1), \ldots, X(i-1)); \quad i \geqslant 1 \qquad (2\text{-}27a)$$

and

$$X(i) = Y(i) \oplus G_i(K, Y(0), Y(1), \ldots, Y(i-1)); \quad i \geqslant 1 \qquad (2\text{-}27b)$$

where $X(0) \equiv Y(0) \equiv Z$.

Using the same arguments that led to the definition of a general block cipher (see Block Chaining Using a Variable Key), the reader can see that Equations 2-27a and 2-27b represent the most general relation that can be established for a stream cipher. Whenever such relations hold for a stream cipher, it is called a *general stream cipher*.

It follows (Equation 2-27a) that the jth bit in ciphertext block Y(i) is affected by only the jth bit in plaintext block X(i), whereas it is potentially affected by every bit in plaintext blocks X(1) through X(i − 1). In like manner, it follows (Equation 2-27b) that the jth bit in the recovered plaintext block X(i) is affected by only the jth bit in ciphertext block Y(i), whereas it is potentially affected by every bit in ciphertext blocks Y(1) through Y(i − 1).

Since the recovered plaintext block X(i) is potentially affected by every bit in ciphertext blocks Y(1) through Y(i − 1), error propagation is achieved. However, because the jth bit in the recovered plaintext block X(i) depends only on the jth bit in ciphertext block Y(i), the following statements may be made. *For the general stream cipher, intersymbol dependence can be achieved for all but the final block. For the general block cipher, there is intersymbol dependence throughout all blocks.* This is an important difference between block ciphers and stream ciphers.

A Chaining Method with the Property of Self-Synchronization

A self-synchronizing stream cipher can be obtained from Figure 2-28 by defining function h as

$$h(U(i-1), Y(i-1)) = Y(i-1); \quad i > 1$$

that is, by feeding back the ciphertext as input to the algorithm. By defining $Y(0) \equiv Z$, it follows that

$$g_K(U(i)) = g_K(Y(i-1)); \quad i \geqslant 1$$

and so, from Equations 2-25a and 2-25b, encipherment and decipherment can be expressed as

$$Y(i) = X(i) \oplus g_K(Y(i-1)); \quad i \geqslant 1 \qquad (2\text{-}28a)$$

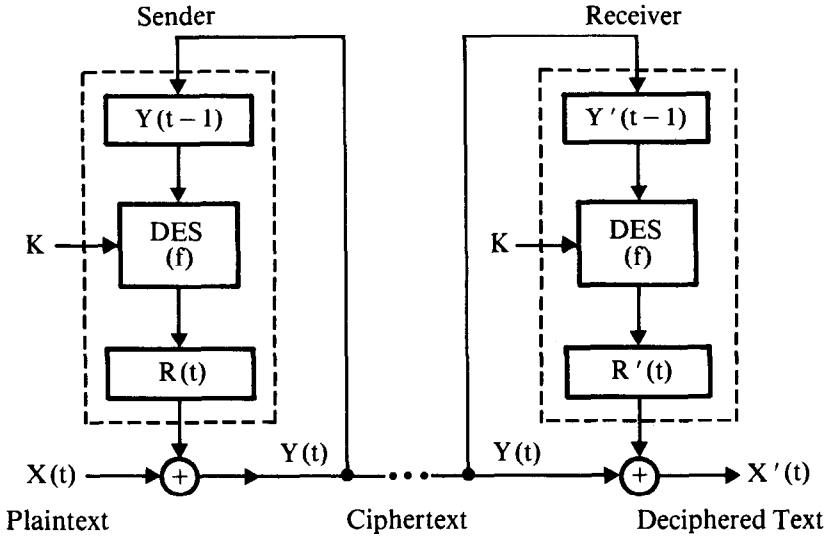
and

$$X(i) = Y(i) \oplus g_K(Y(i-1)); \quad i \geqslant 1 \qquad (2\text{-}28b)$$

respectively.

It follows (Equation 2-28b) that an error in ciphertext block Y(i − 1) can potentially affect every bit in the computed quantity $g_K(Y(i-1))$, and hence can cause every bit in the recovered plaintext block X(i) to be in error. Moreover, it follows (Equation 2-28b) that an error in ciphertext block Y(i − 1) will cause the corresponding bit positions in the recovered plaintext block X(i − 1) to be in error. That is, if the third, fifth, and eleventh ciphertext bits in Y(i − 1) are in error, then the third, fifth, and eleventh recovered plaintext bits in X(i − 1) will be in error. Finally, it follows (Equation 2-28b) that an error in ciphertext block Y(i − 1) will at most affect only the recovery of plaintext blocks X(i − 1) and X(i), but it will not affect the recovery of subsequent plaintext blocks X(i + 1), X(i + 2), and so forth. Hence the scheme is self-synchronizing.

A specific example of a self-synchronizing stream cipher, the ciphertext auto-key cipher, is shown in Figure 2-29. The cryptographic bit-stream is produced in blocks of 64 bits by enciphering the contents of a 64-bit input register, denoted by Y for the sender and Y′ for the receiver, and storing the result in a 64-bit output register, denoted by R for the sender and R′ for the



At t = 1: $X(t) \equiv Z$; $Y(t-1) \neq Y'(t-1)$

At t > 1: $Y(t-1) = Y'(t-1)$ implies in synchronization
$Y(t-1) \neq Y'(t-1)$ implies out of synchronization

Figure 2-29. Ciphertext Auto-Key Cipher

receiver. The contents of registers Y, Y′, R, and R′ at time t are denoted by Y(t), Y′(t), R(t), and R′(t), respectively.

Before communication within the system is possible, the sender and receiver must be synchronized. This is necessary since registers Y and Y′ are assumed to be volatile (i.e., stored information is lost when power to the cryptographic device is turned off). Therefore, it is assumed that at time $t = 0$

$$Y(0) \neq Y'(0)$$

At time $t = 1$, synchronization is accomplished by transmitting a 64-bit initializing vector Z, instead of the usual block of plaintext. This causes the same block of ciphertext to be gated into registers Y and Y′ so that Y(1) equals Y′(1), and hence synchronization is achieved. However, unlike subsequent blocks of transmitted plaintext, the first block (the initializing vector) is not presented to the user at the receiving end.

At time $t > 0$, the input $Y(t-1)$ is enciphered using key K to obtain

$$R(t) = f_K(Y(t-1))$$

R(t) is Exclusive-ORed with the data block X(t) to obtain

$$Y(t) = R(t) \oplus X(t)$$

Y(t) is then transmitted to the receiving end where the input $Y'(t-1)$ is enciphered using key K to obtain

$$R'(t) = f_K(Y'(t-1))$$

R′(t) is Exclusive-ORed with the ciphertext block Y(t) to obtain the recovered plaintext, X′(t).

$$X'(t) = Y(t) \oplus f_K(Y'(t-1))$$

If the first data block X(1) is defined as the initializing vector (Z) (i.e., $X(1) \equiv Z$) then the following may be said:

1. At $t = 1$, $Y(0) \neq Y'(0)$ implies that $X(1) \neq X'(1)$ even though X′(1) is not presented to the user. However, if Y(1) is received without error, then the sender and receiver are in synchronization.
2. At $t > 1$, $Y(t-1) \neq Y'(t-1)$ implies that $X(t) \neq X'(t)$, (i.e., the receiver obtains incorrect plaintext). However, if Y(t) is received without error, then the sender and receiver are in synchronization. $Y(t-1) = Y'(t-1)$ implies that $X(t) = X'(t)$ (i.e., the receiver obtains correct plaintext).

If errors on the transmission line (bit changes, but not bit additions or deletions) cause sender and receiver to get out of synchronization ($Y(t) \neq$

Y'(t)), then error-free transmission of another block of ciphertext will cause sender and receiver to come back into synchronization (Y(t + 1) = Y'(t + 1)). Generally, the ciphertext blocksize can be less than 64 bits in length if desired. In this case, only the necessary bits from R are Exclusive-ORed with plaintext, and the feedback will affect fewer bits in Y.

For all practical purposes, a 48-bit initializing vector is enough to provide adequate cryptographic strength. If the sender and receiver are able to sense when they are resynchronizing, a protocol can be established whereby only a 48-bit block of ciphertext is sent for this purpose. Both sender and receiver pad this 48-bit block of ciphertext with a designated constant (all zeros), so that the values placed into registers Y and Y' are the same. Furthermore, attacks on initializing vectors should be prevented by generation of these quantities within the secure area of the cryptographic device.

Cipher Feedback Stream Cipher

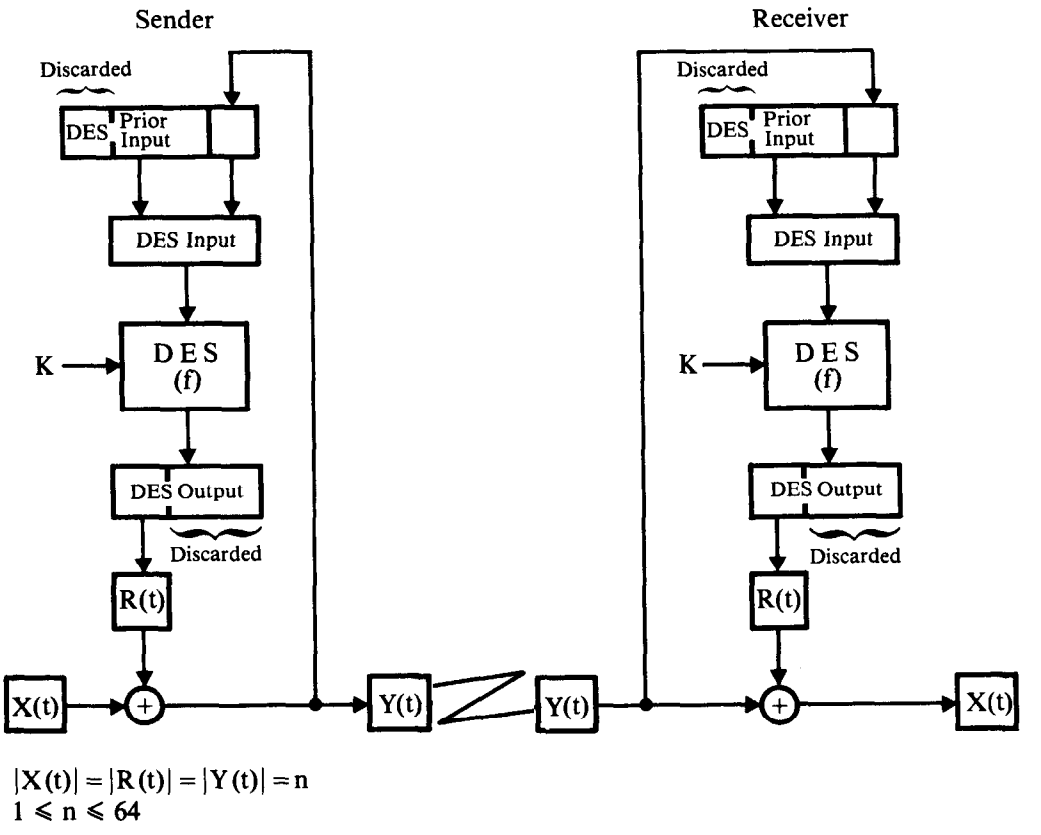
By definition, a ciphertext auto-key cipher produces its cryptographic bitstream using feedback from ciphertext. The various algorithms differ in the way this ciphertext is manipulated before being used. One such algorithm, mentioned in the proposed U.S. Federal Standard 1026 [26], is called *cipher feedback* (Figure 2-30). In this approach, the leftmost n bits of the DES output are Exclusive-ORed with n bits of plaintext to produce n bits of ciphertext, where n is the number of bits enciphered at one time ($1 \leqslant n \leqslant 64$). These n bits of ciphertext are fed back into the algorithm by shifting the current DES input n bits to the left, and then appending the n bits of ciphertext to the right side of the shifted input to produce a new DES input used for the next interaction of the algorithm.

A seed value, which must be the same for both sender and receiver, is used as an initial input to DES in order to generate the cryptographic bitstream. Standard 1026 allows seed length to vary from 8 to 64 bits, but to ensure compatability among users it requires that all cipher feedback implementations must be capable of using a 48-bit seed. Both the sender and receiver are synchronized by right justification of the seed in the input to DES and setting the remaining bits equal to 0.

An Example of Seed Generation

One method of producing seed values is to use the DES algorithm as a generator of pseudo-random numbers. (See Stream Ciphers for a discussion of requirements for initializing vectors, or seed values.) The seed values are unpredictable because the key used by DES to produce the cryptographic bitstream is also used to produce the seed values.

During an initialization phase, a nonsecret quantity, such as the ID of the device in which the algorithm is installed, is placed in a nonvolatile storage that can be accessed (read) only by the cryptographic algorithm. A seed is produced by enciphering this initial quantity with the installed cryptographic key and using the leftmost m bits ($m \leqslant 64$) from the DES output. The entire 64-bit DES output, however, is used to replace the initial quantity in nonvolatile storage. The content of this nonvolatile



$|X(t)| = |R(t)| = |Y(t)| = n$
$1 \leqslant n \leqslant 64$

Figure 2-30. Cipher Feedback

storage continually changes and, in addition, cannot be manipulated by external means.

Let IC_0 be the starting value supplied by the user and placed in non-volatile storage, where IC stands for *initial condition*. During the first seed generation, IC_1 is produced by encrypting IC_0

$$f_K(IC_0) = IC_1$$

and IC_1 replaces IC_0. During the second seed generation, IC_2 is produced by encrypting IC_1 and IC_2 replaces IC_1. The process continues in this manner. The method of generating and using a seed in cipher feedback is illustrated in Figure 2-31.

The Cipher Feedback approach is self-synchronizing, since any bit change occurring in the ciphertext during transmission gets shifted out of the DES input after 64 additional ciphertext bits are sent and received. If, for example, 8 bits are enciphered at one time, as shown in Figure 2-30, and a bit is altered in $Y(t_1)$, changing it to $Y^*(t_1)$, then the DES inputs at sender and receiver are as shown in Figure 2-32, where the 5-byte seed is defined as S1, S2, . . . , S5. In this case, the blocks of ciphertext, given by $Y^*(t_1)$, $Y(t_2)$, . . . , $Y(t_8)$,

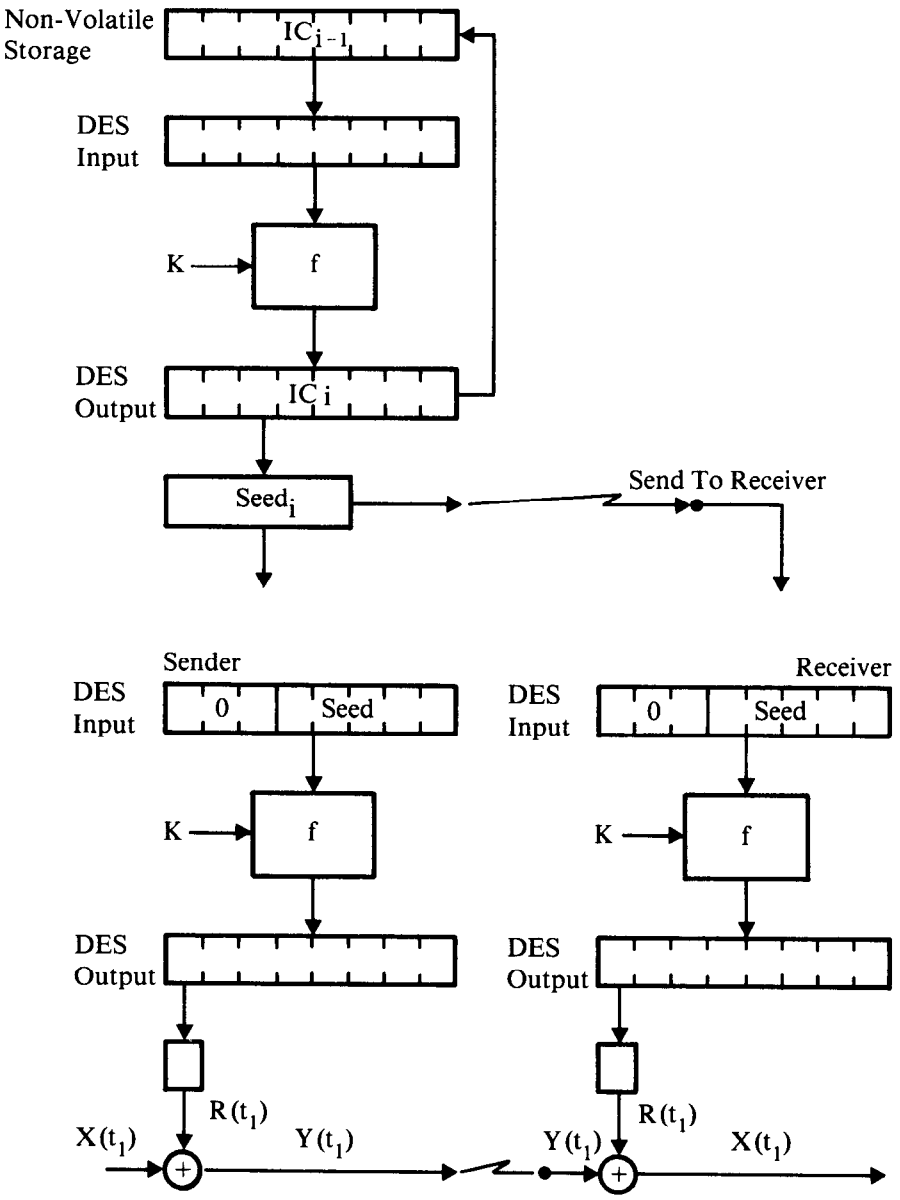


Figure 2-31. Cipher Feedback with a 40-bit Seed and 8-bit Plaintext

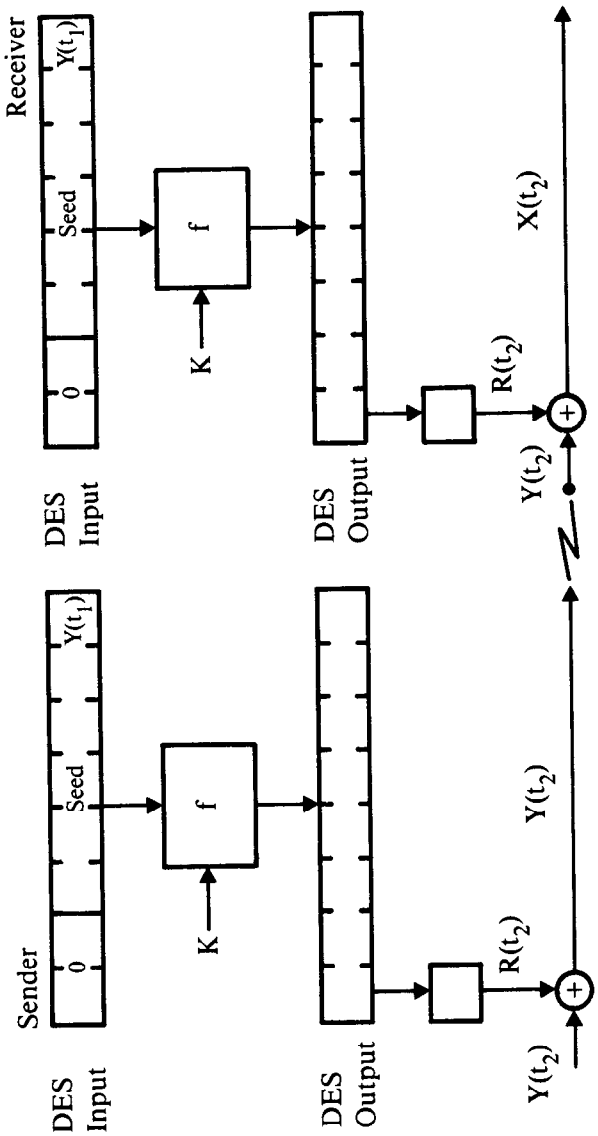


Figure 2-31 (cont'd). Cipher Feedback with a 40-bit Seed and 8-bit Plaintext

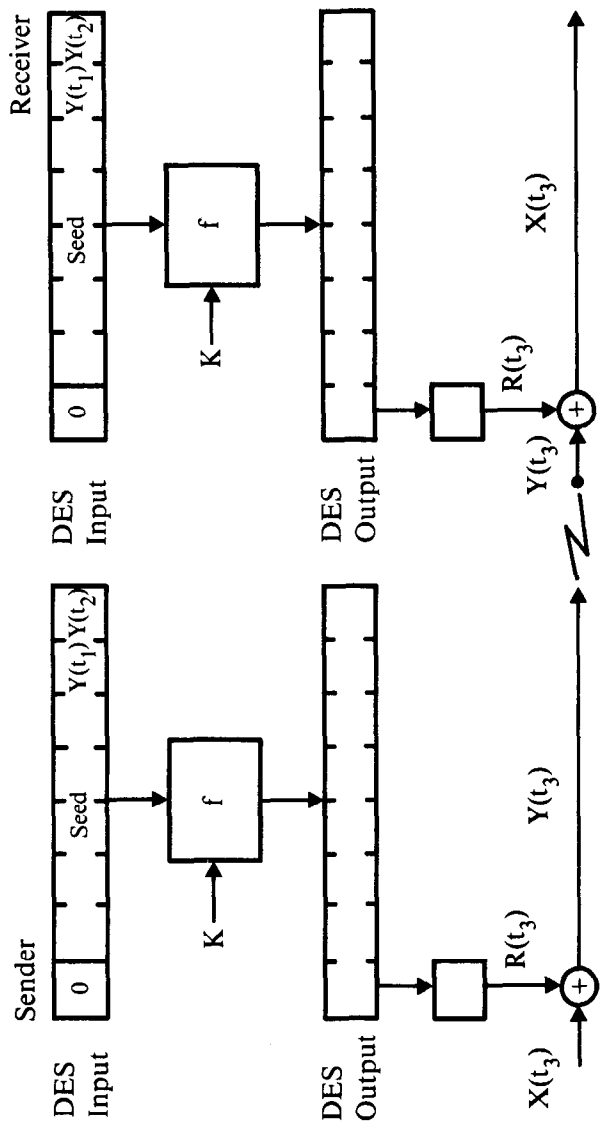


Figure 2-31 (cont'd). Cipher Feedback with a 40-bit Seed and 8-bit Plaintext

Iteration	DES Input At Sender	DES Input At Receiver
0	0 0 0 S1 S2 S3 S4 S5	0 0 0 S1 S2 S3 S4 S5
1	0 0 S1 S2 S3 S4 S5 $Y(t_1)$	0 0 S1 S2 S3 S4 S5 $Y^*(t_1)$
2	0 S1 S2 S3 S4 S5 $Y(t_1)$ $Y(t_2)$	0 S1 S2 S3 S4 S5 $Y^*(t_1)$ $Y(t_2)$
3	S1 S2 S3 S4 S5 $Y(t_1)$ $Y(t_2)$ $Y(t_3)$	S1 S2 S3 S4 S5 $Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$
4	S2 S3 S4 S5 $Y(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$	S2 S3 S4 S5 $Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$
5	S3 S4 S5 $Y(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$	S3 S4 S5 $Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$
6	S4 S5 $Y(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$	S4 S5 $Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$
7	S5 $Y(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$	S5 $Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$
8	$Y(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$ $Y(t_8)$	$Y^*(t_1)$ $Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$ $Y(t_8)$
9	$Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$ $Y(t_8)$ $Y(t_9)$	$Y(t_2)$ $Y(t_3)$ $Y(t_4)$ $Y(t_5)$ $Y(t_6)$ $Y(t_7)$ $Y(t_8)$ $Y(t_9)$

Figure 2-32. Self-Synchronizing Feature in Cipher Feedback

will be correctly deciphered at the receiver only by chance, since the DES input in each case is incorrect. After eight blocks of uncorrupted ciphertext have been received, given by $Y(t_2), \ldots, Y(t_9)$, both the sender's and receiver's cryptographic devices will have equal DES inputs again.

In general, any bit change in an n-bit block of ciphertext can cause a change in any of the corresponding n bits of recovered plaintext and in any of the 64 bits of recovered plaintext immediately following. However, one should realize that a permanent out-of-synch condition will result if a ciphertext bit is added or dropped, since the integrity of the block boundary is lost. To recover from such an error, the sender and receiver would have to have a way to establish the beginning and end of blocks of bits that are enciphered at one time (n = 8 bits in the given example). On the other hand, if enciphering takes place on a bit-by-bit basis (n = 1), then the property of self-synchronization is maintained even when bits are lost or added. This is because blocks are bits, and therefore the block boundary cannot be disturbed. (Note that in the example where n = 8, self-synchronization would be maintained if bits were dropped or added in blocks of 8 bits.)

Examples of Cipher Feedback

Figures 2-33 and 2-34 illustrate two examples of cipher feedback using 8-bit blocks and the described method of seed generation.

Key (external) = 1 3 3 4 5 7 7 9 9 B B C D F F 1

Key (internal) = F 0 C C A A F 5 5 6 6 7 8 F

IC_{i-1} = 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4

IC_i = 9 9 A D F 9 4 D 9 C E 6 3 0 C 7

Plaintext = 0 1 2 3 4 5 6 7 8 9 A B C D E F

Seed Length	Seed (underlined) Followed by Ciphertext
16 bits	99AD190E35C419F818AA
24 bits	99ADF938B80C2CF1E1F7CC
32 bits	99ADF94DB347FC9D5F21D142
40 bits	99ADF94D9C8C266744C539AA59
48 bits	99ADF94D9CE6E88A57084C7A0E57

Figure 2-33. Cipher Feedback—Example 1

Key (external) = 49BC26469EBA7304

Key (internal) = 3249667 7C9E332

IC_{i-1} = 285BC74684BCD734

IC_i = 73B35D20EE034A73

Plaintext = FEDCBA9876543210

Seed Length	Seed (underlined) Followed by Ciphertext
16 bits	73B3554CE44CA6A60601
24 bits	73B35DDE02A95F890D110E
32 bits	73B35D202B4EC26CCD9A882B
40 bits	73B35D20EE56852B80C35CB1AF
48 bits	73B35D20EE032E69D50427AB6B27

Figure 2-34. Cipher Feedback—Example 2

EFFECTS OF PADDING AND INITIALIZING VECTORS

When the block cipher is used for communication security, padding is generally the easiest and most straightforward way to handle short blocks. The small amount of message expansion which results from padding can normally be tolerated within the communication network.

When padding is used, an additional character called the *pad count* must be included as part of the pad characters. The pad count specifies the number of pad characters, including itself, which have been appended to the block. This procedure works well for short blocks, but it creates a problem for standard blocks. Strictly speaking, an extra block of pad characters must be appended to a message whenever its length is a multiple of the blocksize. This allows the procedure to be applied uniformly to all transmissions.

The problem of performance degradation, which may result from adding an extra block of pad characters when the block is already a multiple of the cipher's blocksize, can be greatly reduced by using a *pad indicator bit*. (For example, a 0 indicates no padding and a 1 indicates padding.) The pad indicator bit is transmitted with each message as part of the message's header.

As a measure of the amount of message expansion caused by padding, the *cryptographic throughput factor* (ξ) is defined as

$$\xi = \frac{N_x}{N_y}$$

where

N_x = the length in bits of message X

N_y = the length in bits of cryptogram Y (Y is enciphered from X)

If $N_y = nb$, where n represents the number of blocks in Y and b the block-size in bits, and n_p represents the number of required bits of padding, then

$$\xi = \frac{(nb - n_p)}{nb} = 1 - \frac{n_p}{nb}$$

Figure 2-35 shows a plot of ξ versus message length (in characters), when the DES block cipher is used.

When the stream cipher is used for communication security, practically no throughput degradation results from transmission of the initializing vector Z, provided that this is done only at sign-on or power-up time. In most situations, however, this assumption is not justified. In practice, each communication node is required to multiplex its transmissions among several different nodes which are all competing for the right to transmit data. This requires that the last initial state be stored and saved for each temporarily inactive session. Generally speaking, this procedure is less desirable than transmitting a new initializing vector each time communication is reactivated. In addition to multiplexing problems, line errors may create an out-of-synch condition between a pair of communicating nodes. This problem appears to

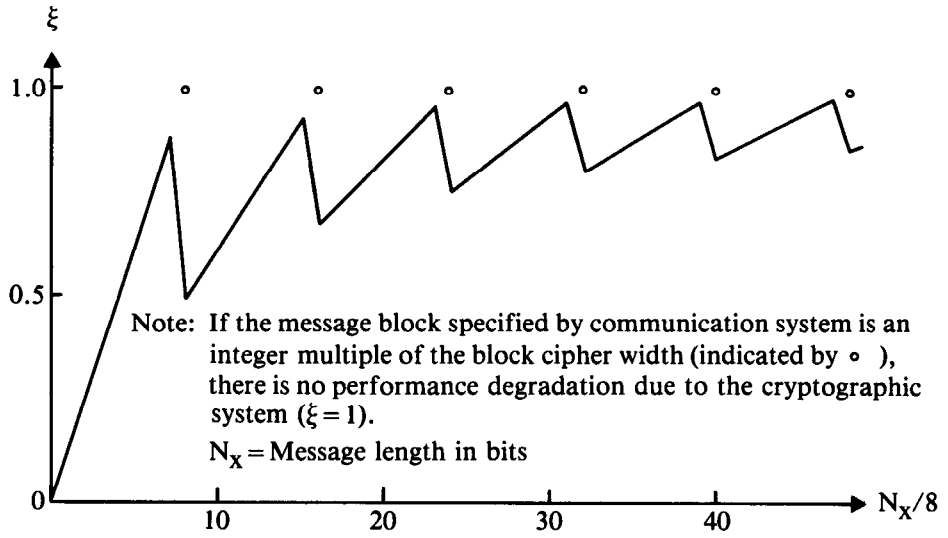


Figure 2-35. Degradation of Throughput in Block Ciphers due to Padding

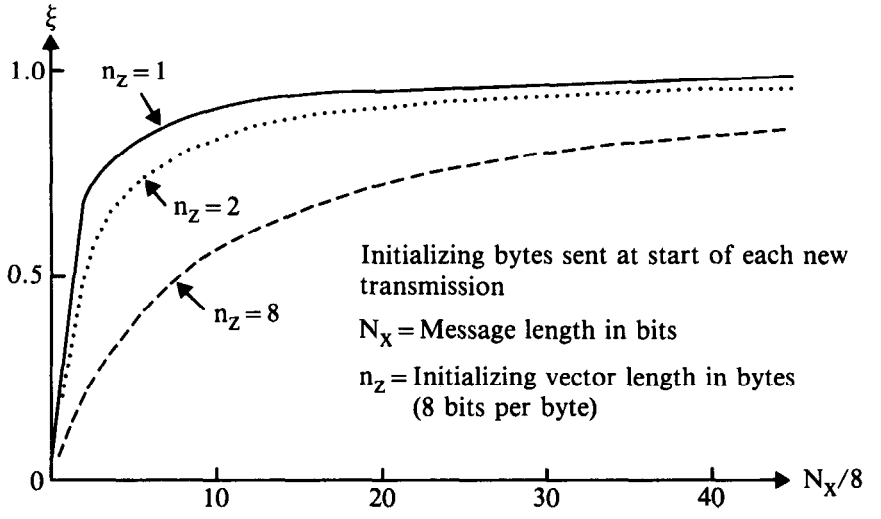


Figure 2-36. Degradation of Throughput in Stream Ciphers due to Initializing Vector

be best resolved by sending another initializing vector. Let n_z represent the number of bytes in Z. Figure 2-36 shows several plots ($n_z = 1, 2, \ldots, 6$) of ξ versus message length (in characters) when the stream cipher is used.

CRYPTOGRAPHIC MESSAGE AUTHENTICATION USING CHAINING TECHNIQUES

The authentication technique described below permits one to determine with a high level of confidence whether a string of text (plaintext or ciphertext) has been altered (accidentally or intentionally). When enciphered data are transmitted or stored within a computing system, either the ciphertext or the recovered plaintext may be authenticated, depending on which is more convenient for the particular application. For example, in the case of enciphered keys used by a cryptographic system's key manager, it may be impractical to decipher these keys to authenticate them. In contrast, authenticating plaintext may be useful in situations where the data are not confidential (i.e., where data are transmitted or stored within the computing system in unenciphered form).

Authentication is accomplished by verifying a bit pattern, called the *authentication code* (AC), that has been computed and appended to the text (plaintext or ciphertext) at a prior time when the data was assumed or known to be correct (see Figure 2-37).

The AC must be a function (ϕ) of the text (TXT), and should have the following properties.

1. It should be computationally infeasible for an opponent to compute $\phi(\text{TXT}')$ for a different text, $\text{TXT}' \neq \text{TXT}$. Otherwise, an opponent could replace TXT and $\phi(\text{TXT})$ with TXT′ and $\phi(\text{TXT}')$, respectively.

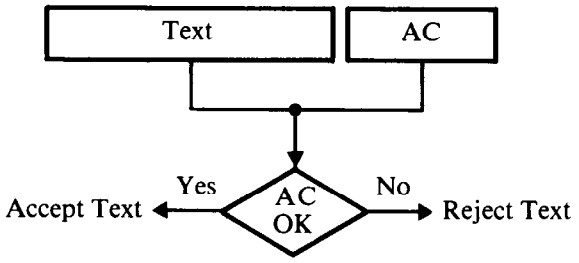


Figure 2-37. Text Checking Procedure Using an Authentication Code

2. It should be computationally infeasible for an opponent to find a different text, TXT′, such that ϕ(TXT′) equals ϕ(TXT). Otherwise, an opponent could replace TXT with TXT′, while leaving ϕ(TXT) unchanged.
3. ϕ(TXT) should be uniformly distributed in the sense that for TXT′ ≠ TXT, the probability that ϕ(TXT′) = ϕ(TXT) is $1/2^c$, where c is the number of bits in AC. In that case, there is only a small chance ($1/2^c$) that a different text (changed either deliberately or accidentally) will be accepted as genuine when C is chosen large enough.

One way to implement an authentication scheme is to exploit the error propagation property obtained with certain chaining techniques. This idea can be explained by reference to the block chaining method that incorporates a plaintext and ciphertext feedback (Figure 2-16). It has the advantage that secrecy and authentication can be achieved in one operation.

For purposes of discussion, function h is assumed to be addition modulo 2^{64}, and the length of the initializing vector Z is equal to the cipher's blocksize. (Using an Exclusive-OR for function h has been shown to be weak, see Authentication by an Encryption Method Without the Property of Error Propagation, Chapter 8.) Let X(1), X(2), . . . , X(n) denote plaintext blocks to be enciphered with key K and initializing vector Z, and let Y(1), Y(2), . . . , Y(n) denote the resulting ciphertext.

$$Y(1) = f_K(X(1) \oplus Z)$$
$$Y(2) = f_K(X(2) \oplus (Y(1) + X(1) \bmod 2^{64}))$$
$$\vdots$$
$$Y(n) = f_K(X(n) \oplus (Y(n-1) + X(n-1) \bmod 2^{64}))$$

The AC could then be defined as follows.

$$AC = f_K(Z \oplus (Y(n) + X(n) \bmod 2^{64}))$$

Note that an additional plaintext block, X(n + 1), is appended to the end of the text to permit the computation of Y(n + 1) = AC. In the example, the additional block is defined to be equal to the initializing vector Z. In another

approach it could be a designated constant, say all zero bits, or it could be a repetition of the first block, X(1).

Assume that the length of the AC equals the cipher's blocksize. Upon decipherment, the recovered plaintext is given by

$$X(1) = f_K^{-1}(Y(1)) \oplus Z$$
$$X(2) = f_K^{-1}(Y(2)) \oplus (Y(1) + X(1) \bmod 2^{64})$$
$$\vdots$$
$$X(n) = f_K^{-1}(Y(n)) \oplus (Y(n-1) + X(n-1) \bmod 2^{64})$$
$$X(n+1) = f_K^{-1}(Y(n+1)) \oplus (Y(n) + X(n) \bmod 2^{64})$$

where X(n + 1) was originally defined to be equal to the initializing vector Z.

By comparing X(n + 1) and Z for equality, a decision can be made to accept or reject the message (Figure 2-38). The receiver accepts the message if X(n + 1) equals Z, since only the sender who knows the secret key K could have properly created Y(n + 1) in the first place. Otherwise, the message is rejected.

It is also possible to authenticate a message by reconstructing the AC (Figure 2-39) instead of deciphering Y(n + 1) and recovering Z. In that case, Y(1) through Y(n) are deciphered as before, but then Z, X(n), and Y(n) are combined to form Z (X(n) + Y(n) mod 2^{64}) and this quantity is enciphered with the secret key K to produce AC. The receiver accepts the message if Y(n + 1) equals AC, otherwise, the message is rejected.

The latter method has the advantage that the AC does not have to be a full block (i.e., one could use c bits for AC, where $c < b$ and b is the number of bits in a block). The probability of accepting a message as genuine when it is not is in that case $1/2^c$ (provided that no error cancellation occurs). Thus, the value of c depends on the risk one is willing to take in accepting a forged or corrupted message as genuine.

To analyze the effects of error propagation, let ciphertext block Y(i) be the last corrupted block. The decipherment of ciphertext block Y(i) is shown in Figure 2-40. If Y(i) is the only ciphertext block in error, then the following is true. The only case in which the error is not propagated all the way through to the recovered plaintext block X(n + 1) occurs when the corrupted ciphertext block Y(i)′ and the deciphered value of Y(i)′ under key K are such that

$$Y(i)' + (f_K^{-1}(Y(i)') \oplus Q) = Y(i) + (f_K^{-1}(Y(i)) \oplus Q)$$

where Q is the value produced at point 2 (i.e., the feedback value at point 1 in Figure 2-40 is unchanged). (Note that the input at point 2 in Figure 2-40 is unchanged and hence cancelled out.) Assuming that an error in Y(i) causes each bit in $f_K^{-1}(Y(i)')$ to differ from its corresponding bit in $f_K^{-1}(Y(i))$ with a probability approximately equal to 0.5, it follows that the probability of the event that error cancellation occurs is approximately equal to $1/2^b$.

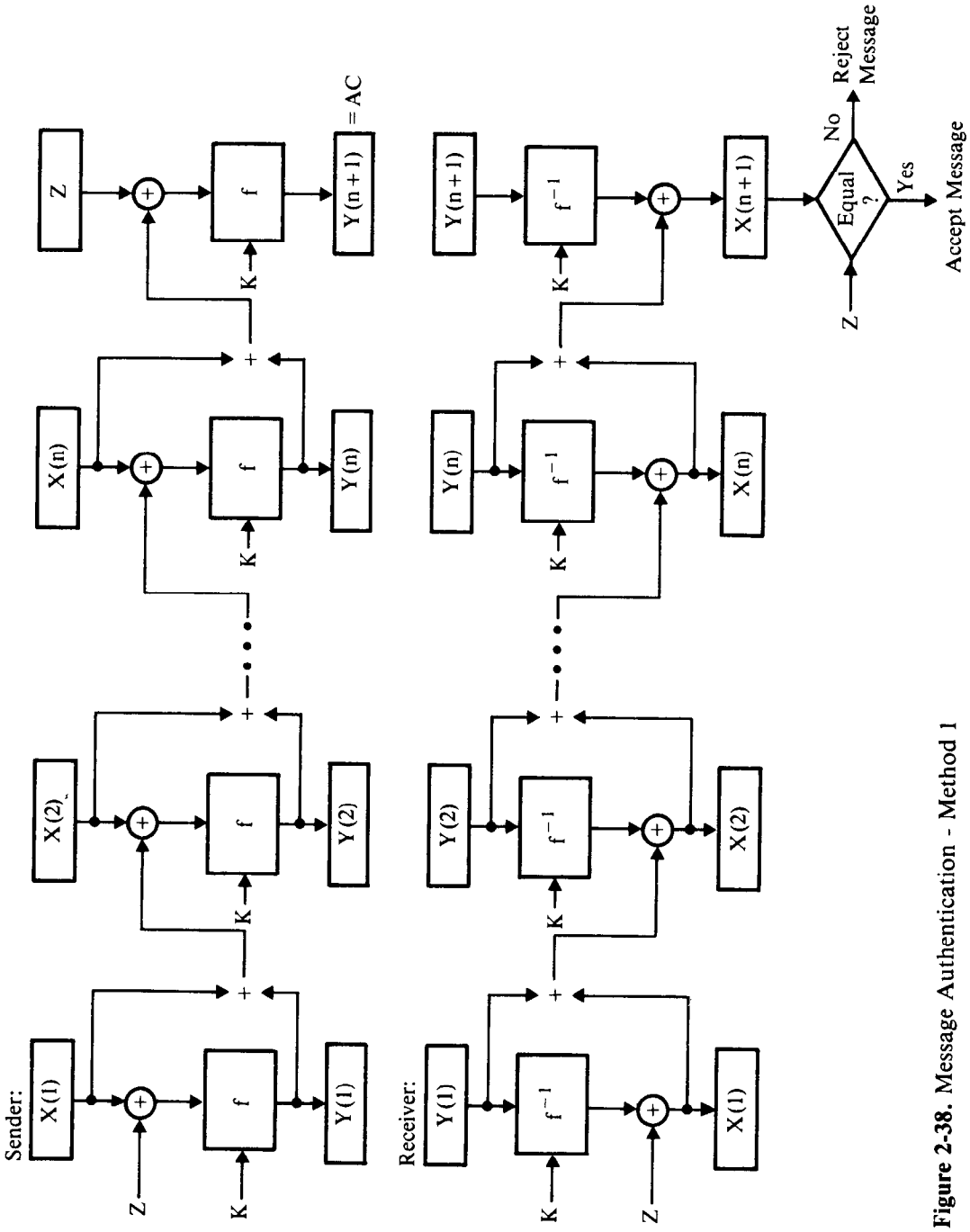


Figure 2-38. Message Authentication - Method 1

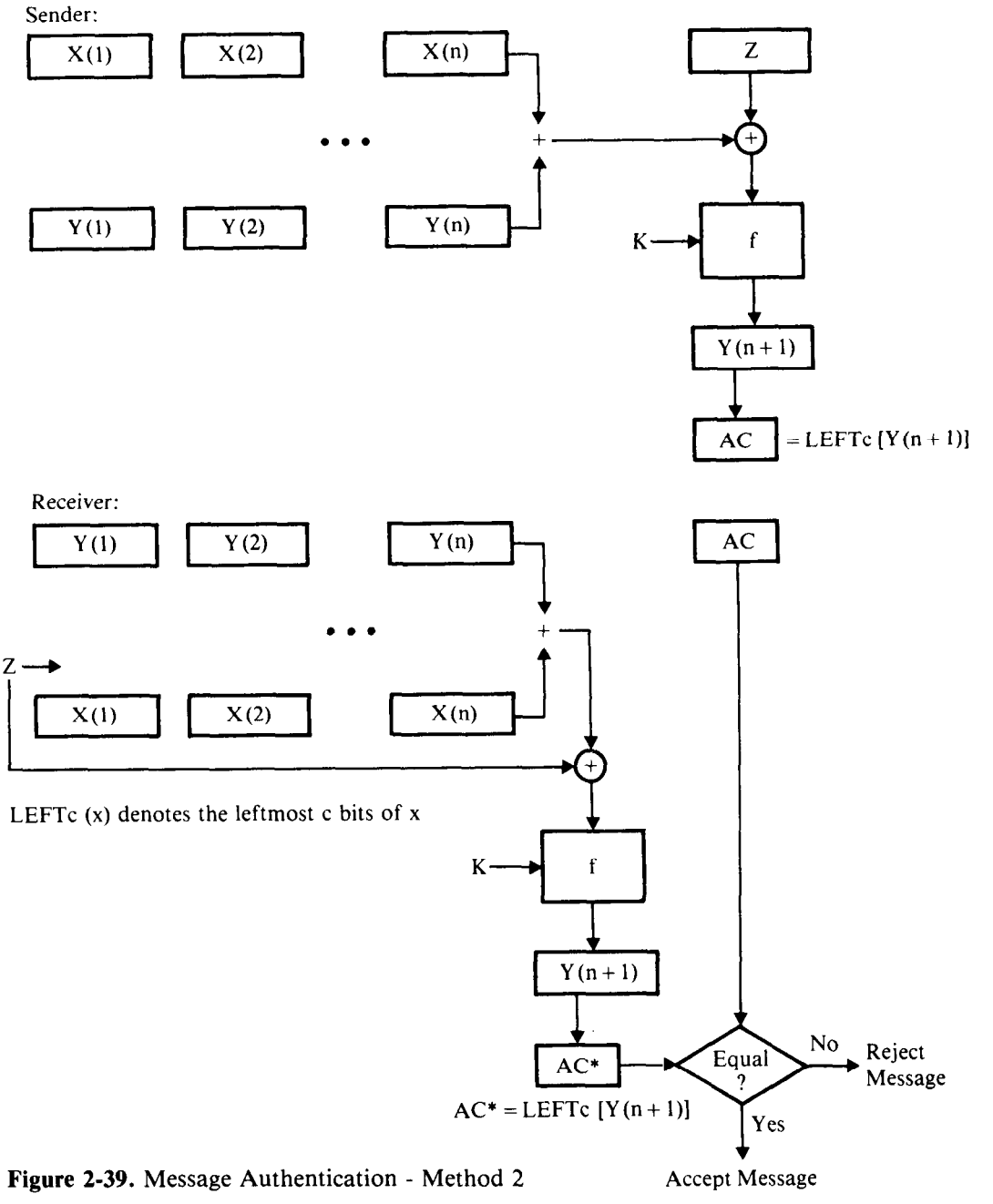


Figure 2-39. Message Authentication - Method 2

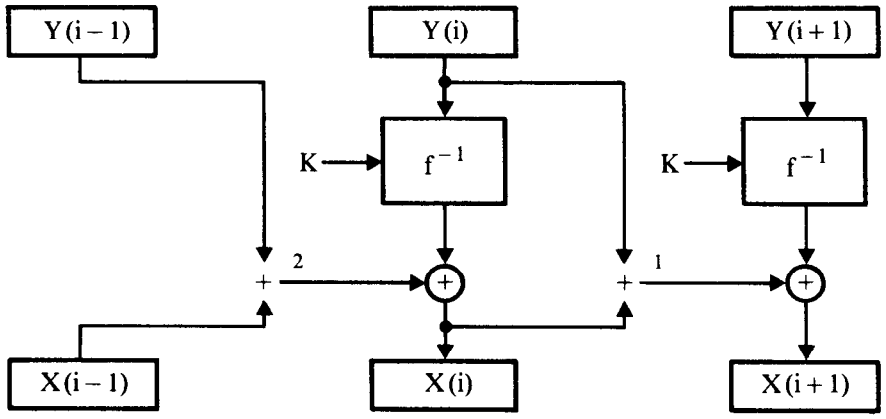


Figure 2-40. Decipherment of Ciphertext Block Y(i)

If, in addition, there are also errors in blocks of ciphertext preceding ciphertext block Y(i), then the following is true. The only case in which the error is not propagated all the way through to the recovered plaintext block X(n + 1) occurs when the corrupted ciphertext block Y(i)′ is such that the deciphered value of Y(i)′ under key K cancels the errors both in Y(i)′ and in the feedback value at point 2 in Figure 2-40 (i.e., the feedback value at point 1 is unchanged).

$$Y(i)' + X(i)' = Y(i) + X(i) \text{ Mod } 2^{64}$$

Again, for all practical purposes, the probability that Y(i)′ will give rise to a $f_K^{-1}(Y(i)')$ that in turn produces an X(i)′ that is self-canceling is approximately equal to $1/2^b$.

Properties one and two for authentication, given above, are satisfied because, under normal conditions an opponent will not know the secret cipher key K, and without knowledge of this key it is computationally infeasible to compute the authentication code or to make systematic changes to the ciphertext that would escape detection. Property three is satisfied if the length c of the AC is large enough.

So far, examples have been given to authenticate messages enciphered with a block cipher. Similar techniques exist for stream ciphers but are not shown here.

COMPARISON OF BLOCK CIPHERS AND STREAM CIPHERS

In the discussion on block ciphers, it was shown that a block cipher need not use an initializing vector (the block cipher could reoriginate). It was also shown that cryptographic strength depends on a minimum required blocksize (the blocksize for DES is 64 bits).

The discussion on stream ciphers showed that a stream cipher must always use an initializing vector (the stream cipher must not reoriginate). A minimum blocksize is not required for the stream cipher, although it was shown that (if desired) the cryptographic bit-stream could be generated in blocks of bits (up to 64 bits per block for DES), and that encipherment could be performed on a block-by-block basis.

In certain applications where highly redundant or structured data are enciphered using a block cipher, patterns in the input stream can be masked or hidden through the use of chaining techniques (block chaining). Stereotyped messages are also masked if an initializing vector is used in conjunction with block chaining. Again, by way of contrast, chaining is not needed in a stream cipher to mask patterns in the input stream, since this is automatically accomplished by the cryptographic bit-stream. Stereotyped messages are also masked, since initializing vectors are always required in a stream cipher.

In certain applications involving authentication, it is desirable for the ciphering technique to have the property of error propagation, that is, an error in the ciphertext causes the recovered plaintext (measured from the point of the error to the end of the recovered plaintext) to be in error. Error propagation can be achieved within block ciphers and stream ciphers through the use of chaining techniques.

A block cipher for which there exist functions $G_1, G_2, \ldots, G_i$ and $H_1, H_2, \ldots, H_i$ such that

$$Y(i) = H_i(K, X(0), X(1), \ldots, X(i)); \quad i \geqslant 1$$

and

$$X(i) = G_i(K, Y(0), Y(1), \ldots, Y(i)); \quad i \geqslant 1$$

where

$$X(0) \equiv Y(0) \equiv Z$$

was defined to be a general block cipher. Two examples of general block ciphers are the block cipher using a variable key (Figure 2-15), and the block cipher using plaintext-ciphertext feedback (Figure 2-16). The self-synchronizing scheme using ciphertext feedback (Figure 2-17) is not a general block cipher. This is because the general relation for X(i) is given by $X(i) = G_i(K, Y(i-1), Y(i))$ (see equation 2-22d) rather than $X(i) = G_i(K, Y(0), Y(1), \ldots, Y(i))$, as shown above.

A stream cipher for which there exist functions $G_1, G_2, \ldots, G_i$ and $H_1, H_2, \ldots, H_i$ such that

$$Y(i) = X(i) \oplus H_1(K, X(0), X(1), \ldots, X(i-1)); \quad i \geqslant 1$$

and

$$X(i) = Y(i) \oplus G_i(K, Y(0), Y(1), \ldots, Y(i-1)); \quad i \geqslant 1$$

where

$$X(0) \equiv Y(0) \equiv Z$$

was defined to be a general stream cipher. An example of the general stream cipher is shown in Figure 2-28. The self-synchronizing scheme using ciphertext feedback (Figure 2-29) is not a general stream cipher. This is because the general relation for X(i) as derived from Equation 2-25b is given by $X(i) = Y(i) \oplus G_i(K, Y(i-1))$ rather than $X(i) = Y(i) \oplus G_i(K, Y(0), Y(1), \ldots, Y(i-1))$, as shown above.

There are two important differences between the general block cipher and the general stream cipher.

1. In the general block cipher, an intersymbol dependence can exist for all blocks. In the general stream cipher, an intersymbol dependence can only exist for all but the last block.
2. The initializing vectors used with the general block cipher need not be frequently changed. They could, for example, be held constant for the duration of a terminal-to-computer communication session lasting the entire day. Hence, the block cipher reoriginates during that session. Since stream ciphers must not reoriginate,the initializing vectors used with the general stream cipher can only be used once.

Instead of frequent generation and transmission of initializing vectors within a communication system, the initializing vectors, or their equivalent condition, could be stored within each system node. Recall that in the key autokey cipher (Figure 2-12), the vectors U(1), U(2), . . . , U(t) represent states that could be used for the purpose of initialization. This is true of stream ciphers in general. The initializing vector Z determines the initial state of the system ($Z \equiv U(1)$). Once the initial state of the system has been set, only the current state of the system need be remembered to maintain synchronization. For example, suppose at the beginning of a session, node A sends vector Z to node B. Transmission of message X from node A to node B leaves both node A and node B in the same state, say U(i). Hence at this point either node A or node B can continue to transmit using U(i) as the initialization vector. A new Z is not required. Although this method works very well in a system with two communication nodes, it becomes extremely complex when several nodes are involved. In practice, this is avoided by generating a new initializing vector each time a cryptographic device changes from a decryption mode into an encryption mode.

A block cipher has the problem of coping with a short block. A record whose length is less than the cipher's blocksize can be enciphered only after it has been padded with enough bits to make it a standard blocksize. However, the last short block within a record whose length is greater than the cipher's blocksize can be enciphered with no data expansion using a technique known as ciphertext-stealing mode.

With a stream cipher, there is no problem enciphering a short block since only as many bits from the cryptographic bit-stream are used as are

	General Block Cipher	General Stream Cipher
Encipherment	$Y(i) = H_i(K, X(0), \ldots, X(i))$	$Y(i) = X(i) \oplus H_i(K, X(0), \ldots, X(i-1))$
Decipherment	$X(i) = G_i(K, Y(0), \ldots, Y(i))$	$X(i) = Y(i) \oplus G_i(K, Y(0), \ldots, Y(i-1))$
	The jth bit of Y(i) depends	
Intersymbol Dependence Within Ciphertext	on every bit in X(1), X(2), ..., X(i)	only on the jth bit in X(i), but, on every bit in X(1), X(2), ..., X(i-1)
	The jth bit of X(i) depends	
Intersymbol Dependence Within Recovered Plaintext	on every bit in Y(1), Y(2), ..., Y(i)	only on the jth bit in Y(i), but, on every bit in Y(1), Y(2), ..., Y(i-1)

Table 2-5. Comparison between a General Block Cipher and a General Stream Cipher

	General Block Cipher	General Stream Cipher
Initializing Vector Z	Not mandatory, but highly desirable. If used, Z can be constant over a session. Z should be 64 bits for DES.	Required for all applications. In the practical case, Z must be frequently changed. To achieve maximum security, Z should contain 48 bits for the DES.
Number of bits which may be enciphered at a single time	Equal to the blocksize of the block cipher (64 bits for the DES). It is not possible to encipher short blocks in a secure way. A short block preceded by a complete block can be enciphered securely, but this will affect error propagation.	Any number from 1 to the maximum determined by design (64 if the DES is used).
Implementation Considerations	Straightforward when Z is not used. Slightly more complicated if Z is infrequently generated.	More complicated because of the frequent generation of Z.

Table 2-5 (cont'd). Comparison between a General Block Cipher and a General Stream Cipher

needed to encipher the plaintext. Hence as an alternative to enciphering short blocks using block cipher mode, these special cases could be handled using stream cipher mode.

When the length of a data record is less than the cipher's blocksize, enciphering the record causes data expansion—regardless of whether block cipher mode or stream cipher mode is used. In a block cipher, these extra bits are used for padding, while in a stream cipher they manifest themselves in the form of a required initializing vector.

With a stream cipher, an initializing vector of 48 bits is sufficient for most applications whenever DES is used. If a weaker system can be tolerated, a 40-bit initializing vector may be sufficient. With fewer than 40 bits, however, the system may be considerably weakened as far as the protection of the first part of the enciphered message is concerned, and an analysis should be performed to determine if this situation can be tolerated. (In applications where data travel over low-speed communication lines, it could very well happen that 16 bits are sufficient.)

Table 2-5 summarizes the similarities and differences between block ciphers and stream ciphers. Figure 2-41 illustrates the effect that a single bit change in ciphertext can have on recovered plaintext, when several different ciphering protocols are considered. Figure 2-42 illustrates the effect that a single bit change in the plaintext can have on the resulting ciphertext (during encipherment).

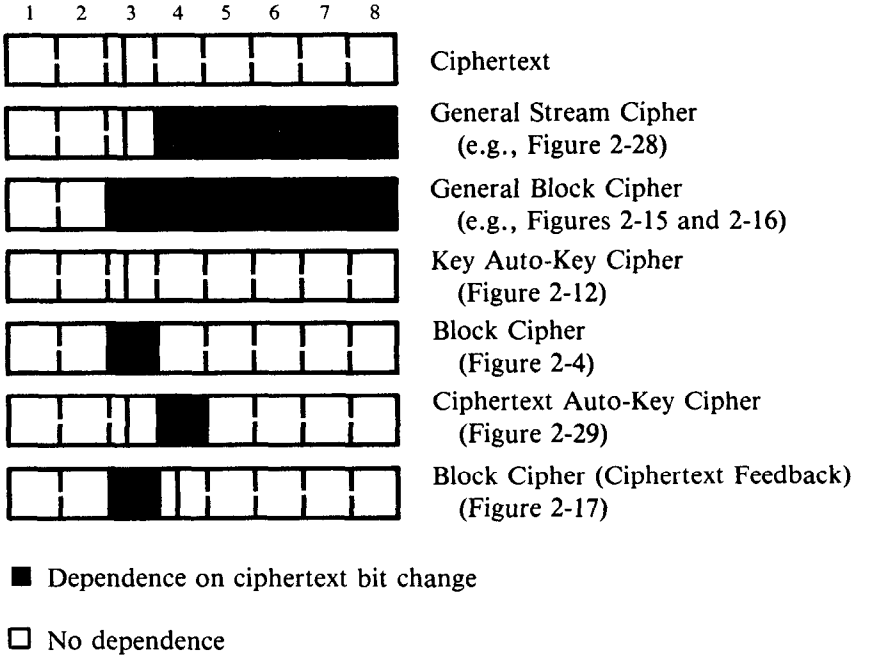


Figure 2-41. Effect on Recovered Plaintext for a One-Bit Ciphertext Change

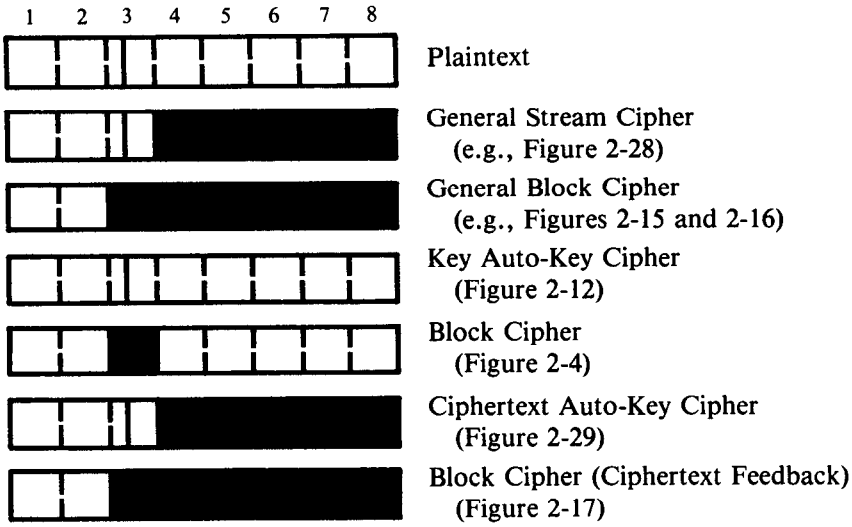


Figure 2-42. Effect on Produced Ciphertext for a One-Bit Plaintext Change

REFERENCES

1. Kahn, D., *The Codebreakers,* Macmillan, New York, 1972.
2. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal,* **28**, 656–715 (1949).
3. Diffie, W., and Hellman, M., "New directions in cryptography," *IEEE Transactions on Information Theory,* **22**, 644–645 (November 1976).
4. Merkle, R., and Hellman, M., "Hiding Information and Receipts in Trap Door Knapsacks," *IEEE Transactions on Information Theory,* **24**, 525–530 (September 1978).
5. Rivest, R. L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM,* **21**, No. 2, 120–126 (1978).
6. McEliece, R. J., "A Public-Key Cryptosystem Based on Algebraic Coding Theory," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, *DSN Progress Report, 42–44,* 114–116 (January-February 1978).
7. Ore, O., *Number Theory and its History*, McGraw-Hill, New York, 1948.
8. Solovay, R., and Strassen, V., "A Fast Monte-Carlo test for primality, *SIAM Journal on Computing,* **6**, 84–85 (March 1977).
9. Miller, G. L., "Reimann's hypothesis and tests for primality," *Proceedings Seventh Annual ACM Symposium on the Theory of Computing,* Albuquerque, New Mexico, 234–239, May 1975. Extended version available as Research Report CS-75-27, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (October 1975).
10. Rabin, M. O., "Probabilistic algorithms," In J. F. Traub, Ed., *Algorithms and Complexity,* Academic Press, New York, 21–40 (1976).

11. Pollard, J. M., "Theorems on factorization and primality testing," *Cambridge Philosophical Society Proceedings,* **76**, 521–528 (1974).
12. Niven, I., and Zuckerman, H. S., *An Introduction to the Theory of Numbers,* Wiley, New York, 1972.
13. Simmons, G. J., and Norris, J. N., "Preliminary comments on the M.I.T. public-key cryptosystem," *Cryptologia,* **1**, No. 4, 406–414 (1977).
14. Rivest, R. L., "Remarks on a proposed cryptanalytic attack on the M.I.T. Public-Key Cryptosystem," *Cryptologia,* **2**, No. 1, 62–65 (1978).
15. Williams, H. C., and Schmid, B., "Some Remarks Concerning the M.I.T. Public-Key Cryptosystem," *Science Report No. 91,* Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada (1979).
16. Davies, D. W., Price, W. L., and Parkin, G. I., "An Evaluation of Public-Key Cryptosystems," *NPL Report CTU 1,* National Physical Laboratory, Teddington, Middlesex TW11 OLW, UK (1979).
17. Herlestam, T., "Critical Remarks on Some Public-Key Cryptosystems," *BIT,* **18**, 493–496 (1978).
18. Rivest, R. L., "Critical Remarks on 'Some Critical Remarks on Public-Key Cryptosystems' by Tore Herlestam," *BIT,* **19**, 1–3 (1978).
19. Blakley, B., and Blakley, G. R., "Security of Number Theoretic Public Key Cryptosystems Against Random Attack, I," *Cryptologia,* **2**, No. 4, 305–321 (1978).
20. Blakely, B., and Blakley, G. R., "Security of Number Theoretic Public Key Cryptosystems Against Random Attack, II," *Cryptologia,* **3**, No. 1, 29–42 (1979).
21. Blakley, B., and Blakley, G. R., "Security of Number Theoretic Public Key Cryptosystems Against Random Attack, III," *Cryptologia,* **3**, No. 2, 105–118 (1979).
22. Shamir, A., "A Polynomial Time Algorithm for Breaking Merkle-Hellman Cryptosystems," (extended abstract) Applied Mathematics, The Weizmann Institute, Rehovot, Israel (April 1982).
23. Vernam, G. S., "Cipher Printing Telegraphy Systems for Secret Wire and Radio Telegraphic Communications," *Journal of the AIEE,* **45**, 109–115 (February 1926).
24. Matyas, S. M., Meyer, C. H., and Tuckerman, L. B., "Method and Apparatus for Enciphering Blocks Which Succeed Short Blocks in a Key-Controlled Block-Cipher Cryptographic System," U.S. Patent No. 4,229,818 (October 21, 1980).
25. Konheim, A. G., Mack, M. H., McNeill, R. K., Tuckerman, B., and Waldbaum, G., "The IPS Cryptographic Programs," *IBM Systems Journal,* **19**, No. 2, 253–283 (1980).
26. Proposed Federal Standard 1026, *Telecommunications: Interoperability and Security Requirements for Use of the Data Encryption Standard in the Physical and Data Link Layers of Data Communications,* General Services Administration, Washington, D.C., Draft (January 21, 1982).

Other Publications of Interest

27. Ryska, N. and Herda, S., *Kryptographische Verfahren in der Datenverarbeitung,* Springer Verlag, Berlin, also New York, 1980.
28. Denning, D. E., *Cryptography and Data Security,* Addison-Wesley, Reading, 1982.