PART 1: CRYPTOGRAPHY

CHAPTER 2

SYMMETRIC ENCRYPTION AND MESSAGE CONFIDENTIALITY

All the afternoon Mungo had been working on Stern's code, principally with the aid of the latest messages which he had copied down at the Nevin Square drop. Stern was very confident. He must be well aware London Central knew about that drop. It was obvious that they didn't care how often Mungo read their messages, so confident were they in the impenetrability of the code.

—*Talking to Strange Men,* Ruth Rendell

Amongst the tribes of Central Australia every man, woman, and child has a secret or sacred name which is bestowed by the older men upon him or her soon after birth, and which is known to none but the fully initiated members of the group. This secret name is never mentioned except upon the most solemn occasions; to utter it in the hearing of men of another group would be a most serious breach of tribal custom. When mentioned at all, the name is spoken only in a whisper, and not until the most elaborate precautions have been taken that it shall be heard by no one but members of the group. The native thinks that a stranger knowing his secret name would have special power to work him ill by means of magic.

—*The Golden Bough,* Sir James George Frazer

Symmetric encryption, also referred to as conventional encryption, secret-key, or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the late 1970s.[1] It remains by far the most widely used of the two types of encryption.

This chapter begins with a look at a general model for the symmetric encryption process; this will enable us to understand the context within which the algorithms are used. Then we look at three important block encryption algorithms: DES, triple DES, and AES. This is followed by a discussion of random and pseudorandom number generation. Next, the chapter introduces symmetric stream encryption and describes the widely used stream cipher RC4. Finally, we look at the important topic of block cipher modes of operation.

2.1 SYMMETRIC ENCRYPTION PRINCIPLES

A **symmetric encryption** scheme has five ingredients (Figure 2.1):

- **Plaintext:** This is the original message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.

[1]Public-key encryption was first described in the open literature in 1976; the National Security Agency (NSA) claims to have discovered it some years earlier.

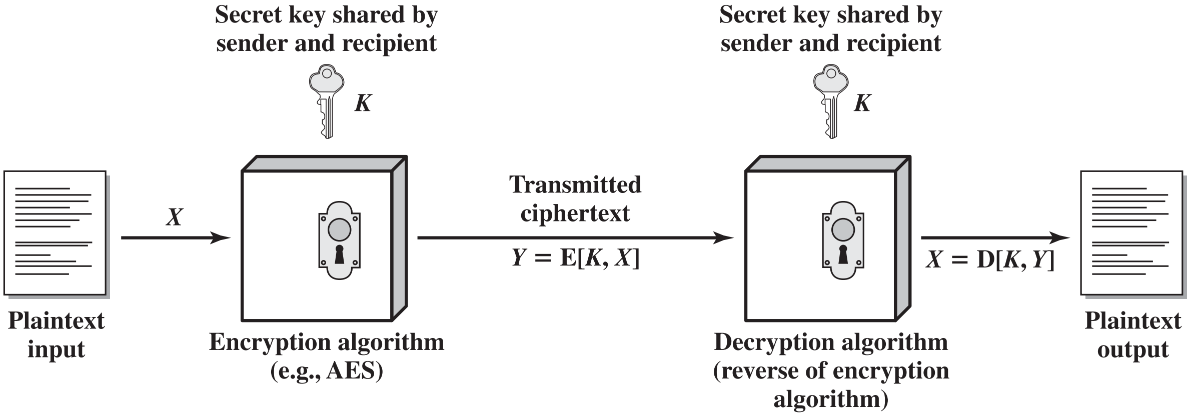


Figure 2.1 Simplified Model of Symmetric Encryption

- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the same secret key and produces the original plaintext.

There are two requirements for secure use of symmetric encryption:

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

It is important to note that the security of symmetric encryption depends on the secrecy of the key, not the secrecy of the algorithm. That is, it is assumed that it is impractical to decrypt a message on the basis of the ciphertext *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret.

This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Cryptography

Cryptographic systems are generically classified along three independent dimensions:

1. **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations be reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.
2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each use a different key, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **The way in which the plaintext is processed.** A **block cipher** processes the input one block of elements at a time, producing an output block for each input block. A **stream cipher** processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis

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

Table 2.1 summarizes the various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general, we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the ciphertext itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an EXE file, a Java source listing, an accounting file, and so on.

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All of these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general

Table 2.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext to be decoded
Known plaintext	• Encryption algorithm • Ciphertext to be decoded • One or more plaintext–ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by a corporation might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 2.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

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

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

Unfortunately, it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully. However, assuming there are no inherent mathematical weaknesses in the algorithm, then a brute-force approach is indicated, and here we can make some reasonable estimates about costs and time.

A brute-force approach involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 2.2 shows how much time is involved for various key sizes. The 56-bit key size is used with the DES (Data Encryption Standard) algorithm. For each key size, the results are shown assuming that it takes 1 μs to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater. The final column of Table 2.2 considers the results for a system that can process 1 million keys per microsecond. As you can see, at this performance level, DES no longer can be considered computationally secure.

Feistel Cipher Structure

Many symmetric block encryption algorithms, including DES, have a structure first described by Horst Feistel of IBM in 1973 [FEIS73] and shown in Figure 2.2. The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K. The plaintext block is divided into two halves, LE_0 and RE_0. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs LE_{i-1} and RE_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys K_i are different from K and from each other and are generated from the key by a subkey generation algorithm. In Figure 2.2, 16 rounds are used, although any number of rounds could be implemented. The right-hand side of Figure 2.2 shows the decryption process.

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

Table 2.2 Average Time Required for Exhaustive Key Search

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

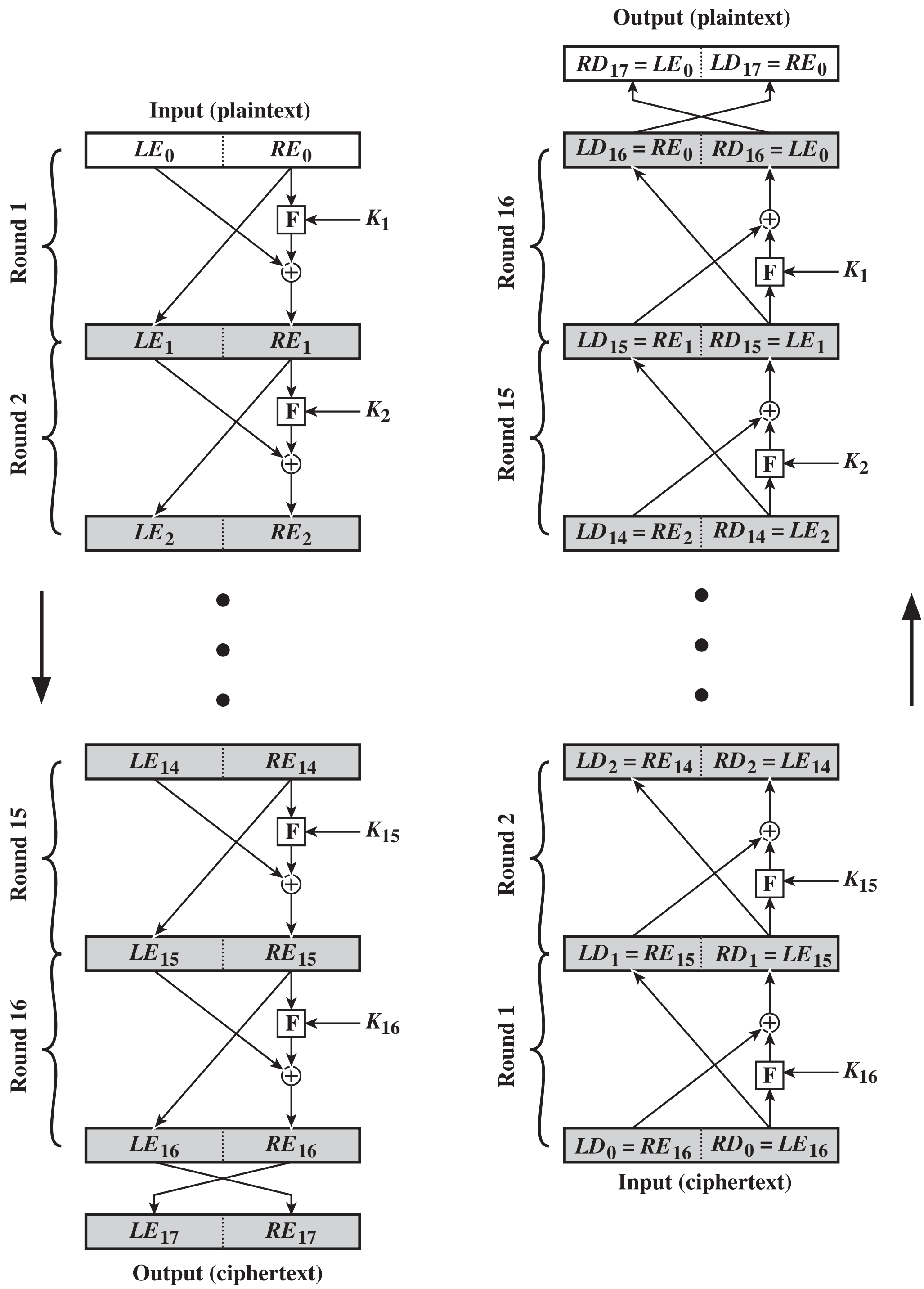


Figure 2.2 Feistel Encryption and Decryption (16 rounds)

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

The **Feistel structure** is a particular example of the more general structure used by all symmetric block ciphers. In general, a symmetric block cipher consists of a sequence of rounds, with each round performing substitutions and permutations conditioned by a secret key value. The exact realization of a symmetric block cipher depends on the choice of the following parameters and design features.

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed. A block size of 128 bits is a reasonable tradeoff and is nearly universal among recent block cipher designs.
- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The most common key length in modern algorithms is 128 bits.
- **Number of rounds:** The essence of a symmetric block cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function:** Again, greater complexity generally means greater resistance to cryptanalysis.

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

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

Decryption with a symmetric block cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on until K_1 is used in the last round. This is a nice feature, because it means we need not implement two different algorithms—one for encryption and one for decryption.

2.2 SYMMETRIC BLOCK ENCRYPTION ALGORITHMS

The most commonly used symmetric encryption algorithms are block ciphers. A **block cipher** processes the plaintext input in fixed-sized blocks and produces a block of ciphertext of equal size for each plaintext block. This section focuses on the three most important symmetric block ciphers: the Data Encryption Standard (DES), triple DES (3DES), and the Advanced Encryption Standard (AES).

Data Encryption Standard

The most widely used encryption scheme is based on the **Data Encryption Standard (DES)** issued in 1977, as Federal Information Processing Standard 46 (FIPS 46) by the National Bureau of Standards, now known as the National Institute of Standards and Technology (NIST). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).[2]

Description of the Algorithm The plaintext is 64 bits in length and the key is 56 bits in length; longer plaintext amounts are processed in 64-bit blocks. The DES structure is a minor variation of the Feistel network shown in Figure 2.2. There are 16 rounds of processing. From the original 56-bit key, 16 subkeys are generated, one of which is used for each round.

The process of decryption with DES is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the DES algorithm, but use the subkeys K_i in reverse order. That is, use K_{16} on the first iteration, K_{15} on the second iteration, and so on until K_1 is used on the 16th and last iteration.

The Strength of DES Concerns about the strength of DES fall into two categories: concerns about the algorithm itself and concerns about the use of a 56-bit key. The first concern refers to the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. Over the years, there have been numerous attempts to find and exploit weaknesses in the algorithm, making DES the most-studied encryption algorithm in existence. Despite numerous approaches, no one has so far succeeded in discovering a fatal weakness in DES.[3]

A more serious concern is key length. With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16} keys. Thus, on the face of it, a brute-force attack appears impractical. Assuming that on average half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years (see Table 2.2) to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker [EFF98]. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force

---

[2]The terminology is a bit confusing. Until recently, the terms *DES* and *DEA* could be used interchangeably. However, the most recent edition of the DES document includes a specification of the DEA described here plus the triple DEA (3DES) described subsequently. Both DEA and 3DES are part of the Data Encryption Standard. Furthermore, until the recent adoption of the official term *3DES*, the triple DEA algorithm was typically referred to as *triple DES* and written as 3DES. For the sake of convenience, we will use 3DES.

[3]At least, no one has publicly acknowledged such a discovery.

approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

A final point: If the only form of attack that could be made on an encryption algorithm is brute force, then the way to counter such attacks is obvious: use longer keys. To get some idea of the size of key required, let us use the EFF cracker as a basis for our estimates. The EFF cracker was a prototype, and we can assume that with today's technology a faster machine is cost effective. If we assume that a cracker can perform one million decryptions per μs, which is the rate used in Table 2.2, then a DES code would take about 10 hours to crack. This is a speed-up of approximately a factor of 7 compared to the EFF result. Using this rate, Figure 2.3 shows how long it would take to crack a DES-style algorithm as a function of key size. For example, for a 128-bit key, which is common among contemporary algorithms, it would take over 10^{18} years to break the code using the EFF cracker. Even if we managed to speed up the cracker by a factor of 1 trillion (10^{12}), it would still take over 1 million years to break the code. So a 128-bit key is guaranteed to result in an algorithm that is unbreakable by brute force.

Triple DES

Triple DES (3DES) was first standardized for use in financial applications in ANSI standard X9.17 in 1985. 3DES was incorporated as part of the Data Encryption Standard in 1999 with the publication of FIPS 46-3.

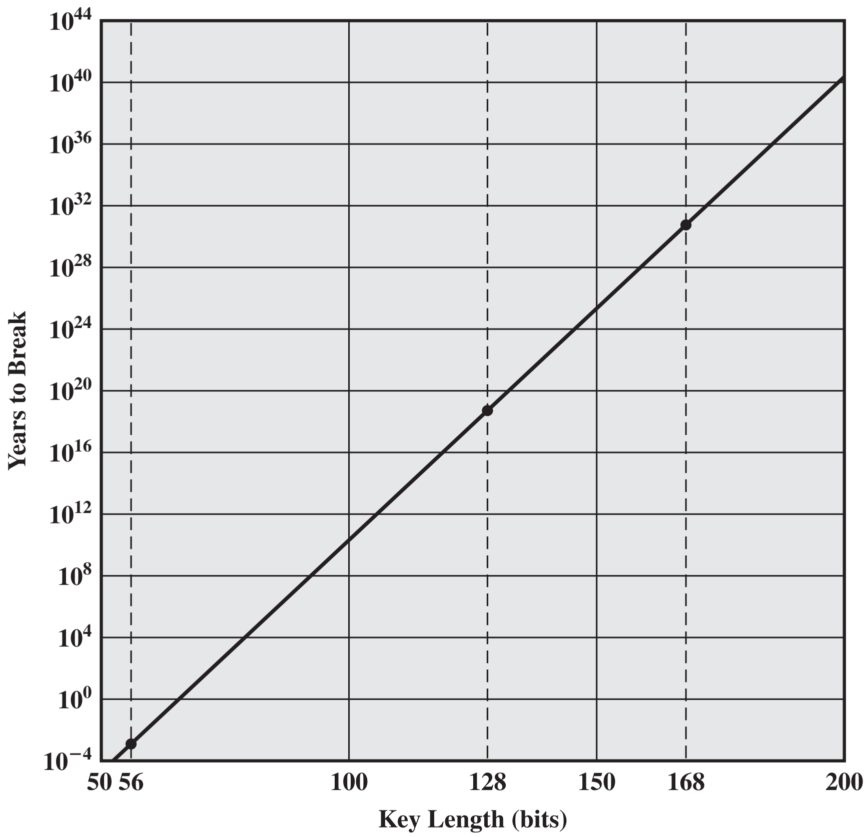


Figure 2.3 Time to Break a Code (assuming 10^6 decryptions/μs)

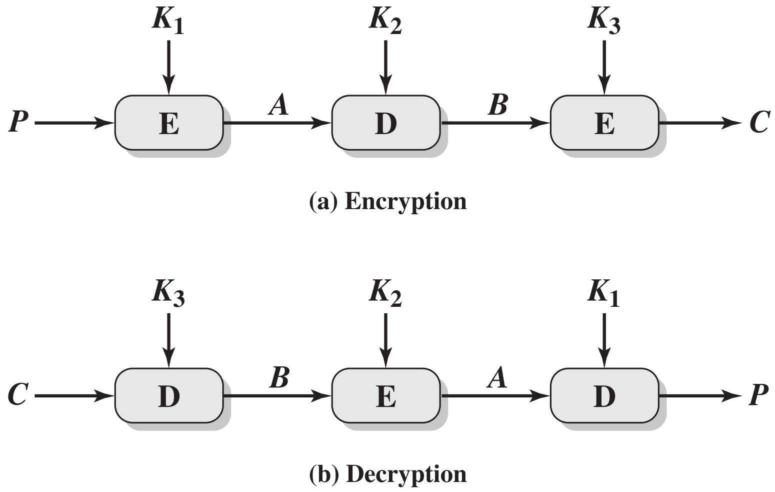


Figure 2.4 Triple DES

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

$$C = \mathrm{E}(K_3, \mathrm{D}(K_2, \mathrm{E}(K_1, P)))$$

where

$$\begin{aligned} C &= \text{ciphertext} \\ P &= \text{plaintext} \\ \mathrm{E}[K, X] &= \text{encryption of } X \text{ using key } K \\ \mathrm{D}[K, Y] &= \text{decryption of } Y \text{ using key } K \end{aligned}$$

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

$$P = \mathrm{D}(K_1, \mathrm{E}(K_2, \mathrm{D}(K_3, C)))$$

There is no cryptographic significance to the use of decryption for the second stage of 3DES encryption. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$$C = \mathrm{E}(K_1, \mathrm{D}(K_1, \mathrm{E}(K_1, P))) = \mathrm{E}[K, P]$$

With three distinct keys, 3DES has an effective key length of 168 bits. FIPS 46-3 also allows for the use of two keys, with $K_1 = K_3$; this provides for a key length of 112 bits. FIPS 46-3 includes the following guidelines for 3DES.

- 3DES is the FIPS approved symmetric encryption algorithm of choice.
- The original DES, which uses a single 56-bit key, is permitted under the standard for legacy systems only. New procurements should support 3DES.
- Government organizations with legacy DES systems are encouraged to transition to 3DES.
- It is anticipated that 3DES and the Advanced Encryption Standard (AES) will coexist as FIPS-approved algorithms, allowing for a gradual transition to AES.

It is easy to see that 3DES is a formidable algorithm. Because the underlying cryptographic algorithm is DEA, 3DES can claim the same resistance to cryptanalysis based on the algorithm as is claimed for DEA. Furthermore, with a 168-bit key length, brute-force attacks are effectively impossible.

Ultimately, AES is intended to replace 3DES, but this process will take a number of years. NIST anticipates that 3DES will remain an approved algorithm (for U.S. government use) for the foreseeable future.

Advanced Encryption Standard

3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DEA. Second, the underlying encryption algorithm in 3DES is the same as in DEA. This algorithm has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Accordingly, there is a high level of confidence that 3DES is very resistant to cryptanalysis. If security were the only consideration, then 3DES would be an appropriate choice for a standardized encryption algorithm for decades to come.

The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DEA was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DEA, is correspondingly slower. A secondary drawback is that both DEA and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.

Because of these drawbacks, 3DES is not a reasonable candidate for long-term use. As a replacement, NIST in 1997 issued a call for proposals for a new **Advanced Encryption Standard (AES)**, which should have a security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. Evaluation criteria included security, computational efficiency, memory requirements, hardware and software suitability, and flexibility.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to five algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.

Overview of the Algorithm AES uses a block length of 128 bits and a key length that can be 128, 192, or 256 bits. In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Figure 2.5 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words: each word is four bytes and the total key schedule is 44 words for the 128-bit key. The ordering of bytes within a

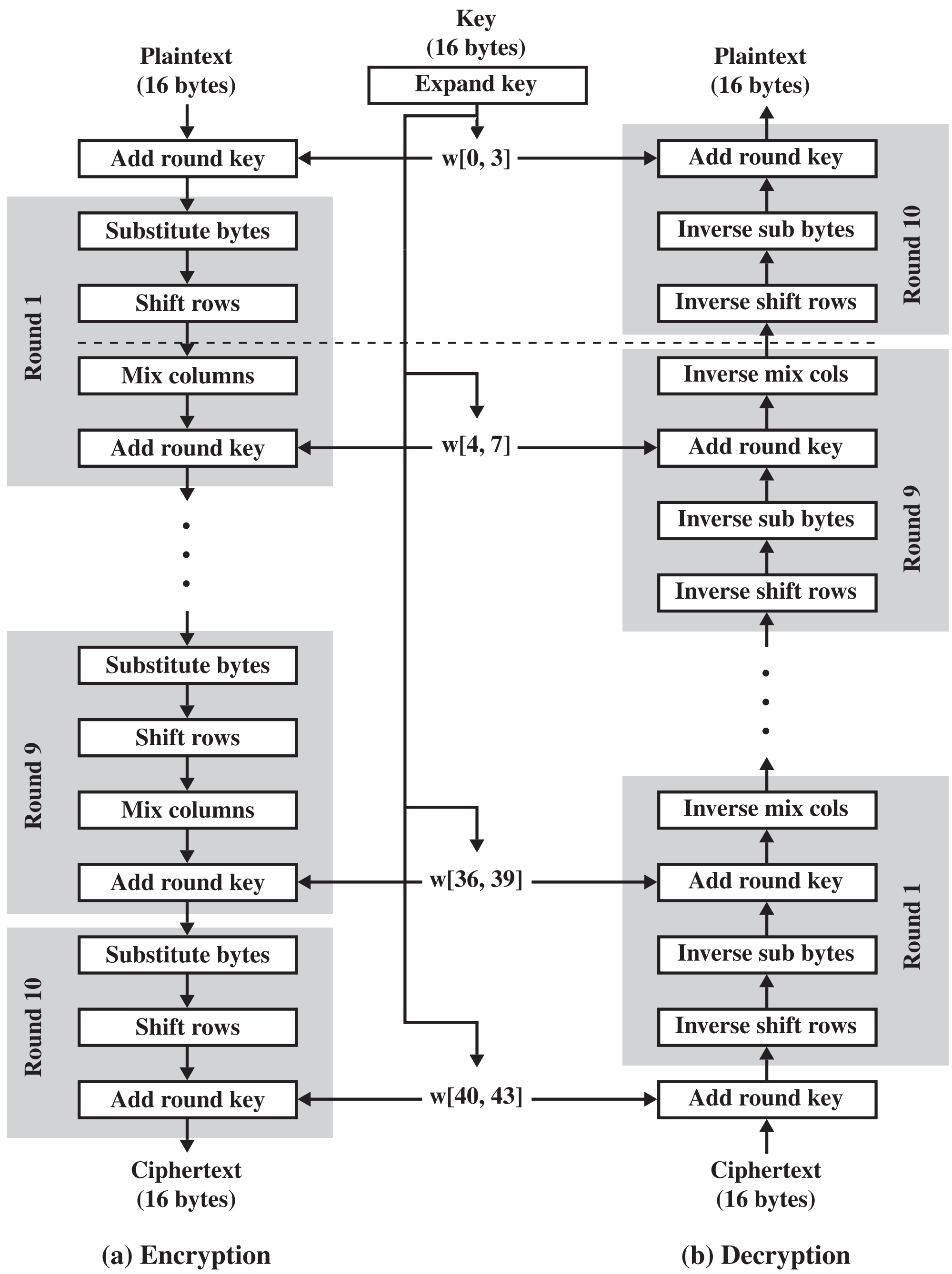


Figure 2.5 AES Encryption and Decryption

matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

The following comments give some insight into AES.

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. AES does not use

a Feistel structure but processes the entire data block in parallel during each round using substitutions and permutation.

2. The key that is provided as input is expanded into an array of forty-four 32-bit words, **w**[i]. Four distinct words (128 bits) serve as a round key for each round.
3. Four different stages are used, one of permutation and three of substitution:
   - **Substitute bytes:** Uses a table, referred to as an S-box,[4] to perform a byte-by-byte substitution of the block.
   - **Shift rows:** A simple permutation that is performed row by row.
   - **Mix columns:** A substitution that alters each byte in a column as a function of all of the bytes in the column.
   - **Add round key:** A simple bitwise XOR of the current block with a portion of the expanded key.
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an Add Round Key stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 2.6 depicts the structure of a full encryption round.
5. Only the Add Round Key stage makes use of the key. For this reason, the cipher begins and ends with an Add Round Key stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The Add Round Key stage by itself would not be formidable. The other three stages together scramble the bits, but by themselves, they would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (Add Round Key) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
7. Each stage is easily reversible. For the Substitute Byte, Shift Row, and Mix Columns stages, an inverse function is used in the decryption algorithm. For the Add Round Key stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus B \oplus B = A$.
8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.
9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 2.5 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), **State** is the same for both encryption and decryption.
10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

---

[4]The term *S-box*, or substitution box, is commonly used in the description of symmetric ciphers to refer to a table used for a table-lookup type of substitution mechanism.

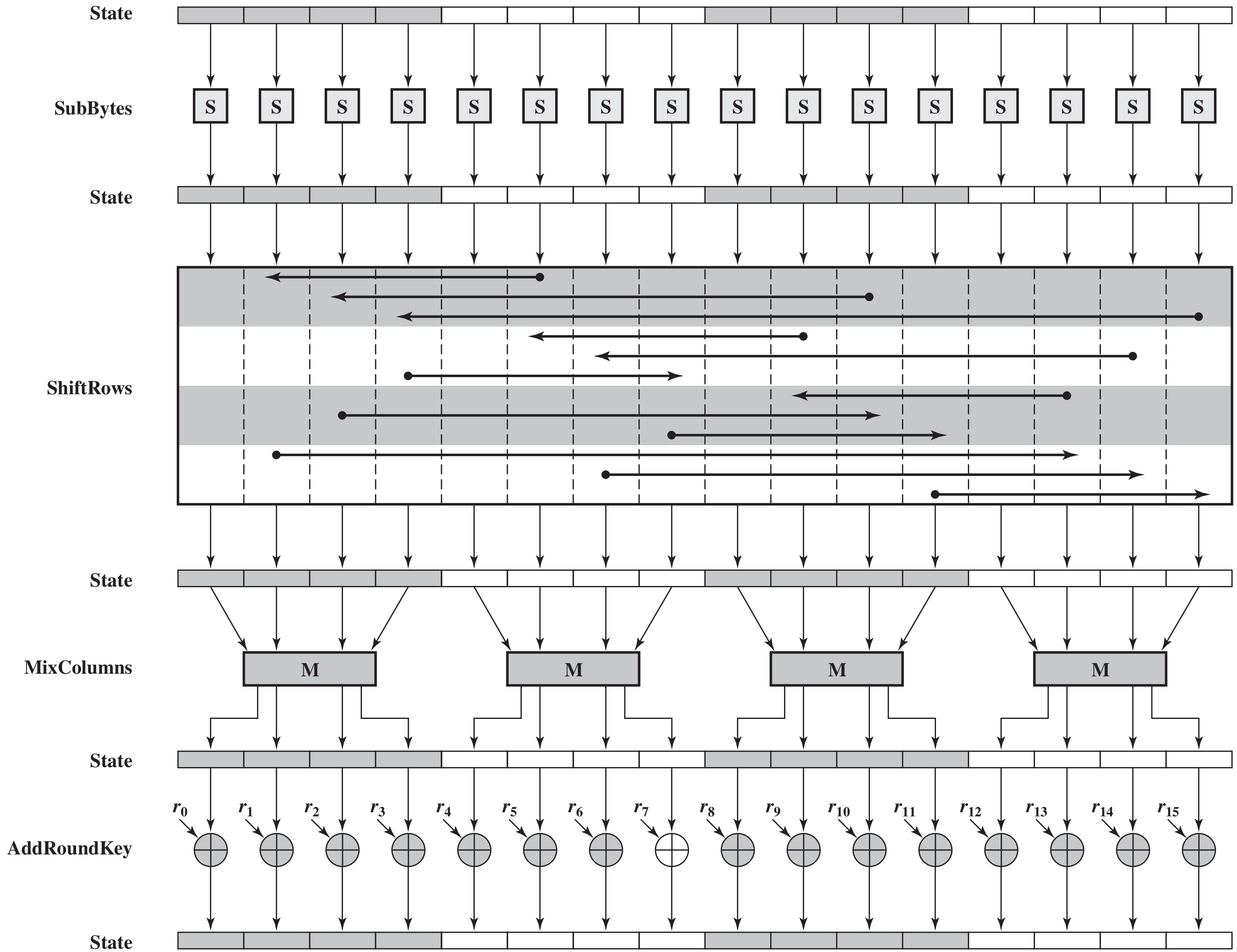


Figure 2.6 AES Encryption Round

2.3 RANDOM AND PSEUDORANDOM NUMBERS

Random numbers play an important role in the use of encryption for various network security applications. We provide an overview in this section. The topic is examined in more detail in Appendix E.

The Use of Random Numbers

A number of network security algorithms based on cryptography make use of random numbers. For example,

- Generation of keys for the RSA public-key encryption algorithm (described in Chapter 3) and other public-key algorithms.
- Generation of a stream key for symmetric stream cipher (discussed in the following section).
- Generation of a symmetric key for use as a temporary session key. This function is used in a number of networking applications, such as Transport Layer Security (Chapter 5), Wi-Fi (Chapter 6), e-mail security (Chapter 7), and IP security (Chapter 8).
- In a number of key distribution scenarios, such as Kerberos (Chapter 4), random numbers are used for handshaking to prevent replay attacks.

These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability.

RANDOMNESS Traditionally, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well-defined statistical sense. The following criteria are used to validate that a sequence of numbers is random.

- **Uniform distribution:** The distribution of bits in the sequence should be uniform; that is, the frequency of occurrence of ones and zeros should be approximately the same.
- **Independence:** No one subsequence in the sequence can be inferred from the others.

Although there are well-defined tests for determining that a sequence of numbers matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

In the context of our discussion, the use of a sequence of numbers that appear statistically random often occurs in the design of algorithms related to cryptography. For example, a fundamental requirement of the RSA public-key encryption scheme discussed in Chapter 3 is the ability to generate prime numbers. In general, it is difficult to determine if a given large number N is prime. A brute-force approach would be to divide N by every odd integer less than $\sqrt{N}$. If N is on the order, say, of 10^{150} (a not uncommon occurrence in public-key cryptography), such a brute-force approach is beyond the reach of human analysts and their computers. However, a number of effective algorithms exist that test the primality of a number by using a sequence of randomly chosen integers as input to relatively simple computations. If the sequence is sufficiently long (but far, far less than $\sqrt{10^{150}}$), the primality of a number can be determined with near certainty. This type of approach, known as randomization, crops up frequently in the design of algorithms. In essence, if a problem is too hard or time-consuming to solve exactly, a simpler, shorter approach based on randomization is used to provide an answer with any desired level of confidence.

Unpredictability In applications such as reciprocal authentication and session key generation, the requirement is not so much that the sequence of numbers be statistically random but that the successive members of the sequence are unpredictable. With "true" random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable. However, as is discussed shortly, true random numbers are not always used; rather, sequences of numbers that appear to be random are generated by some algorithm. In this latter case, care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements.

TRNGs, PRNGs, and PRFs

Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers**.

You may be somewhat uneasy about the concept of using numbers generated by a deterministic algorithm as if they were random numbers. Despite what might be called "philosophical" objections to such a practice, it generally works. As one expert on probability theory puts it [HAMM91],

> For practical purposes we are forced to accept the awkward concept of "relatively random" meaning that with regard to the proposed use we can see no reason why they will not perform as if they were random (as the theory usually requires). This is highly subjective and is not very palatable to purists, but it is what statisticians regularly appeal to when they take "a random sample"—they hope that any results they use will have approximately the same properties as a complete counting of the whole sample space that occurs in their theory.

Figure 2.7 contrasts a **true random number generator (TRNG)** with two forms of pseudorandom number generators. A TRNG takes as input a source that is effectively random; the source is often referred to as an **entropy source**. In essence, the entropy source is drawn from the physical environment of the computer and could include things such as keystroke timing patterns, disk electrical activity, mouse movements, and instantaneous values of the system clock. The source, or combination of sources, serves as input to an algorithm that produces

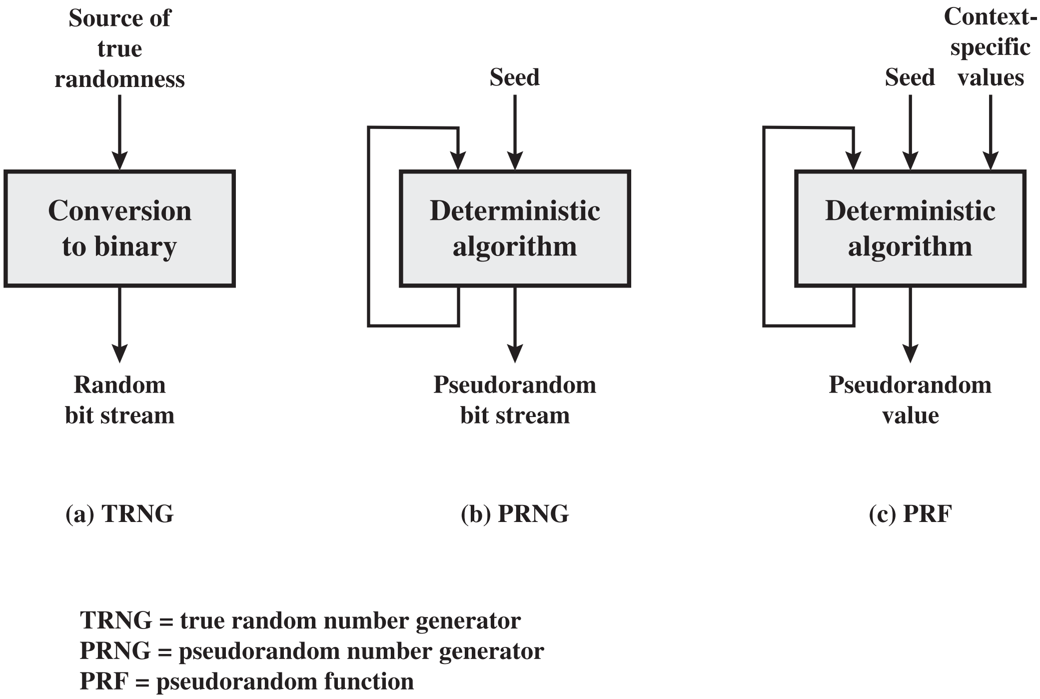


Figure 2.7 Random and Pseudorandom Number Generators

random binary output. The TRNG may simply involve conversion of an analog source to a binary output. The TRNG may involve additional processing to overcome any bias in the source.

In contrast, a PRNG takes as input a fixed value, called the **seed**, and produces a sequence of output bits using a deterministic algorithm. Typically, as shown in Figure 2.7, there is some feedback path by which some of the results of the algorithm are fed back as input as additional output bits are produced. The important thing to note is that the output bit stream is determined solely by the input value or values, so that an adversary who knows the algorithm and the seed can reproduce the entire bit stream.

Figure 2.7 shows two different forms of PRNGs, based on application.

- **Pseudorandom number generator:** An algorithm that is used to produce an open-ended sequence of bits is referred to as a PRNG. A common application for an open-ended sequence of bits is as input to a symmetric stream cipher, as discussed in the following section.
- **Pseudorandom function (PRF):** A PRF is used to produce a pseudorandom string of bits of some fixed length. Examples are symmetric encryption keys and nonces. Typically, the PRF takes as input a seed plus some context specific values, such as a user ID or an application ID. A number of examples of PRFs will be seen throughout this book.

Other than the number of bits produced, there is no difference between a PRNG and a PRF. The same algorithms can be used in both applications. Both require a seed and both must exhibit randomness and unpredictability. Furthermore, a PRNG application may also employ context-specific input.

Algorithm Design

Cryptographic PRNGs have been the subject of much research over the years, and a wide variety of algorithms have been developed. These fall roughly into two categories:

- **Purpose-built algorithms:** These are algorithms designed specifically and solely for the purpose of generating pseudorandom bit streams. Some of these algorithms are used for a variety of PRNG applications; several of these are described in the next section. Others are designed specifically for use in a stream cipher. The most important example of the latter is RC4, described in the next section.
- **Algorithms based on existing cryptographic algorithms:** Cryptographic algorithms have the effect of randomizing input. Indeed, this is a requirement of such algorithms. For example, if a symmetric block cipher produced ciphertext that had certain regular patterns in it, it would aid in the process of cryptanalysis. Thus, cryptographic algorithms can serve as the core of PRNGs. Three broad categories of cryptographic algorithms are commonly used to create PRNGs:

  —**Symmetric block ciphers**

  —**Asymmetric ciphers**

  —**Hash functions and message authentication codes**

Any of these approaches can yield a cryptographically strong PRNG. A purpose-built algorithm may be provided by an operating system for general use. For applications that already use certain cryptographic algorithms for encryption or authentication, it makes sense to re-use the same code for the PRNG. Thus, all of these approaches are in common use.

2.4 STREAM CIPHERS AND RC4

A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time as it goes along. Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate. Examples are given subsequently in this book. In this section, we look at perhaps the most popular symmetric stream cipher, RC4. We begin with an overview of stream cipher structure, and then examine RC4.

Stream Cipher Structure

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 2.8 is a representative diagram of stream cipher structure. In this structure, a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. A pseudorandom stream is one that is unpredictable without knowledge of the input key and which has an apparently random character. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR)

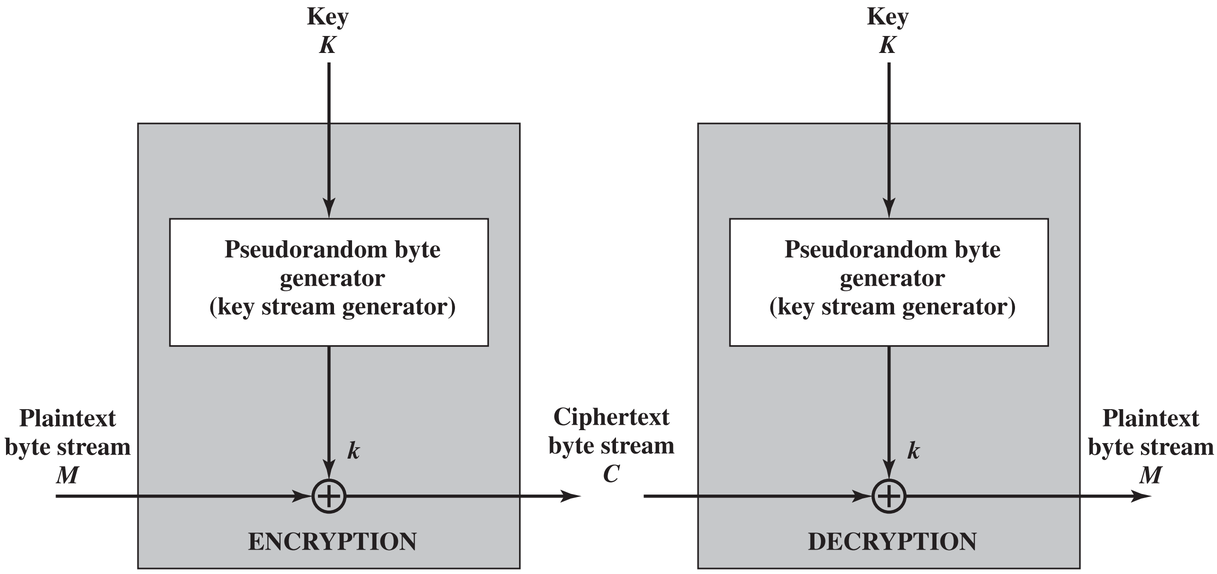


Figure 2.8 Stream Cipher Diagram

operation. For example, if the next byte generated by the generator is `01101100` and the next plaintext byte is `11001100`, then the resulting ciphertext byte is

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

Decryption requires the use of the same pseudorandom sequence:

```
  10100000  ciphertext
⊕ 01101100  key stream
  11001100  plaintext
```

[KUMA97] lists the following important design considerations for a stream cipher.

1. The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat, the more difficult it will be to do cryptanalysis.
2. The keystream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.
3. Note from Figure 2.8 that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The example in this section, RC4, can be implemented in just a few lines of code. Table 2.3, using data from [RESC01], compares execution times of RC4 with three well-known symmetric block ciphers. The advantage of a block cipher is that you can reuse keys. However, if two plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis is often quite simple [DAWS96]. If the two ciphertext streams are XORed together, the result is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis may be successful.

For applications that require encryption/decryption of a stream of data (such as over a data-communications channel or a browser/Web link), a stream cipher might be the better alternative. For applications that deal with blocks of data (such as file transfer, e-mail, and database), block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

The RC4 Algorithm

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100} [ROBS95a]. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers. It is also used in the Wired Equivalent Privacy (WEP) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

The RC4 algorithm is remarkably simple and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1], . . ., S[255]. At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte k (see Figure 2.8) is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of k is generated, the entries in S are once again permuted.

Table 2.3 Speed Comparisons of Symmetric Ciphers on a Pentium II

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	Variable	0.9
RC4	Variable	45

Initialization of S To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is, S[0] = 0, S[1] = 1, . . ., S[255] = 255. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K, and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as:

```
/* Initialization */
for i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];
```

Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255] and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
  j = (j + S[i] + T[i]) mod 256;
  Swap (S[i], S[j]);
```

Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

Stream Generation Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i] and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0]:

```
/* Stream Generation */
i, j = 0;
while (true)
  i = (i + 1) mod 256;
  j = (j + S[i]) mod 256;
  Swap (S[i], S[j]);
  t = (S[i] + S[j]) mod 256;
  k = S[t];
```

To encrypt, XOR the value *k* with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext.

Figure 2.9 illustrates the RC4 logic.

Strength of RC4 A number of papers have been published analyzing methods of attacking RC4 (e.g., [KNUD98], [MIST98], [FLUH00], [MANT01], [PUDO02], [PAUL03], [PAUL04]). None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in

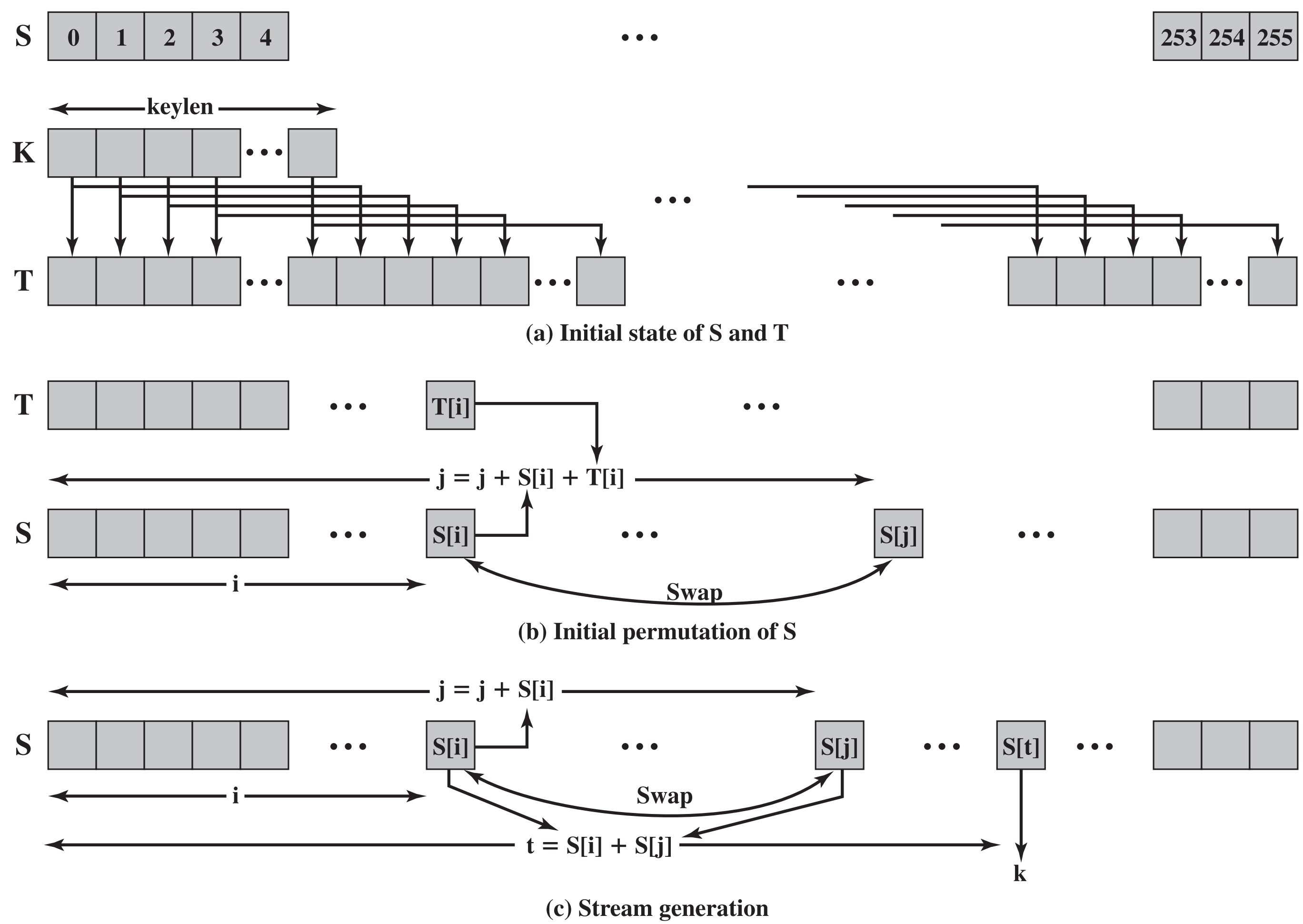


Figure 2.9 RC4

[FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

2.5 CIPHER BLOCK MODES OF OPERATION

A symmetric block cipher processes one block of data at a time. In the case of DES and 3DES, the block length is $b = 64$ bits; for AES, the block length is $b = 128$ bits. For longer amounts of plaintext, it is necessary to break the plaintext into b-bit blocks (padding the last block if necessary). To apply a block cipher in a variety of applications, five **modes of operation** have been defined by NIST (Special Publication 800-38A). The five modes are intended to cover virtually all of the possible applications of encryption for which a block cipher could be used. These modes are intended for use with any symmetric block cipher, including triple DES and AES. The most important modes are described briefly in the remainder of this section.

Electronic Codebook Mode

The simplest way to proceed is using what is known as **electronic codebook (ECB) mode**, in which plaintext is handled b bits at a time and each block of plaintext is encrypted using the same key. The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, one can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

With ECB, if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext. Because of this, for lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext–ciphertext pairs to work with. If the message has repetitive elements with a period of repetition a multiple of b bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks.

Cipher Block Chaining Mode

In the **cipher block chaining (CBC) mode** (Figure 2.10), the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function

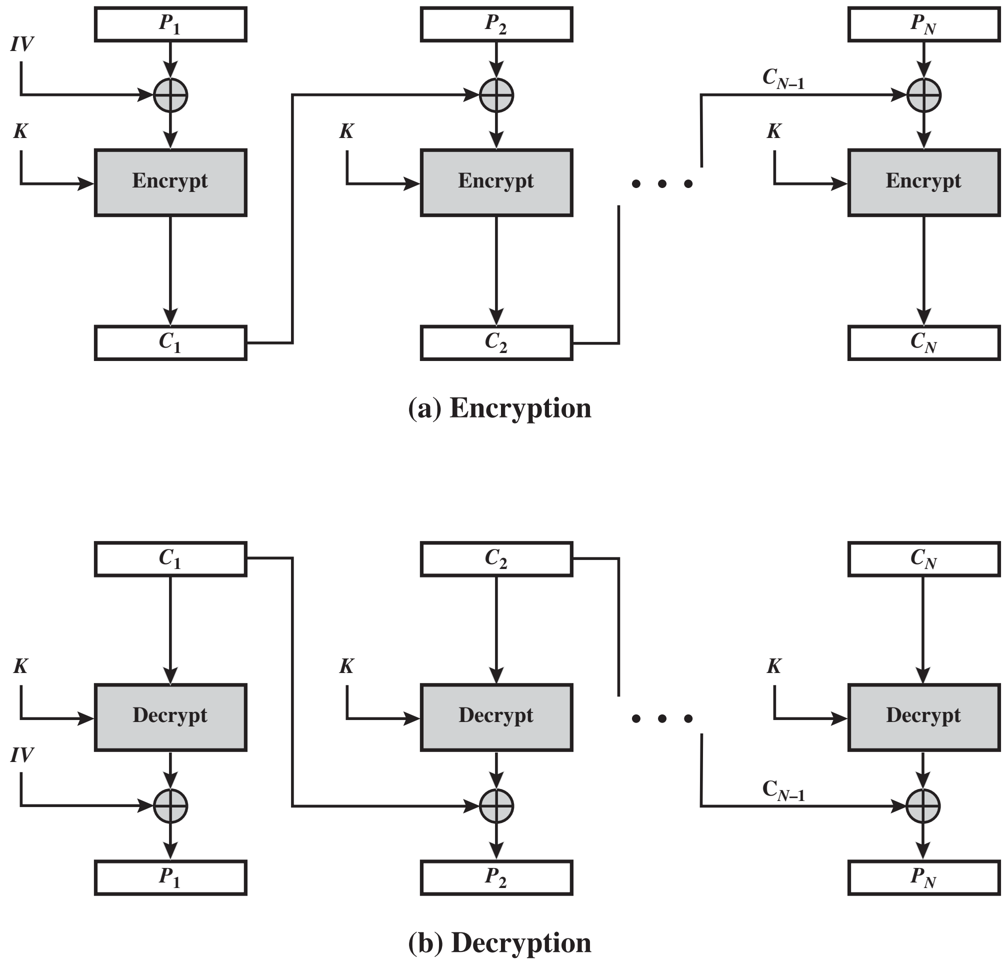


Figure 2.10 Cipher Block Chaining (CBC) Mode

for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_j = \mathrm{E}(K, [C_{j-1} \oplus P_j])$$

where E[K, X] is the encryption of plaintext X using key K, and $\oplus$ is the exclusive-OR operation. Then

$$\mathrm{D}(K, C_j) = \mathrm{D}(K, \mathrm{E}(K, [C_{j-1} \oplus P_j]))$$
$$\mathrm{D}(K, C_j) = C_{j-1} \oplus P_j$$
$$C_{j-1} \oplus \mathrm{D}(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

which verifies Figure 2.10b.

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext.

The IV must be known to both the sender and receiver. For maximum security, the IV should be protected as well as the key. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider the following:

$$C_1 = \mathrm{E}(K, [\mathrm{IV} \oplus P_1])$$
$$P_1 = \mathrm{IV} \oplus \mathrm{D}(K, C_1)$$

Now use the notation that $X[j]$ denotes the jth bit of the b-bit quantity X. Then

$$P_1[i] = \mathrm{IV}[i] \oplus \mathrm{D}(K, C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = \mathrm{IV}[i]' \oplus \mathrm{D}(K, C_1)[i]$$

where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed.

Cipher Feedback Mode

It is possible to convert any block cipher into a stream cipher by using the **cipher feedback (CFB) mode**. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted using 8 bits. If more than 8 bits are used, transmission capacity is wasted.

Figure 2.11 depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is $s = 8$. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext.

First, consider encryption. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first unit of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits, and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let $\mathrm{S}_s(X)$ be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus \mathrm{S}_s[\mathrm{E}(K, \mathrm{IV})]$$

Therefore,

$$P_1 = C_1 \oplus \mathrm{S}_s[\mathrm{E}(K, \mathrm{IV})]$$

The same reasoning holds for subsequent steps in the process.

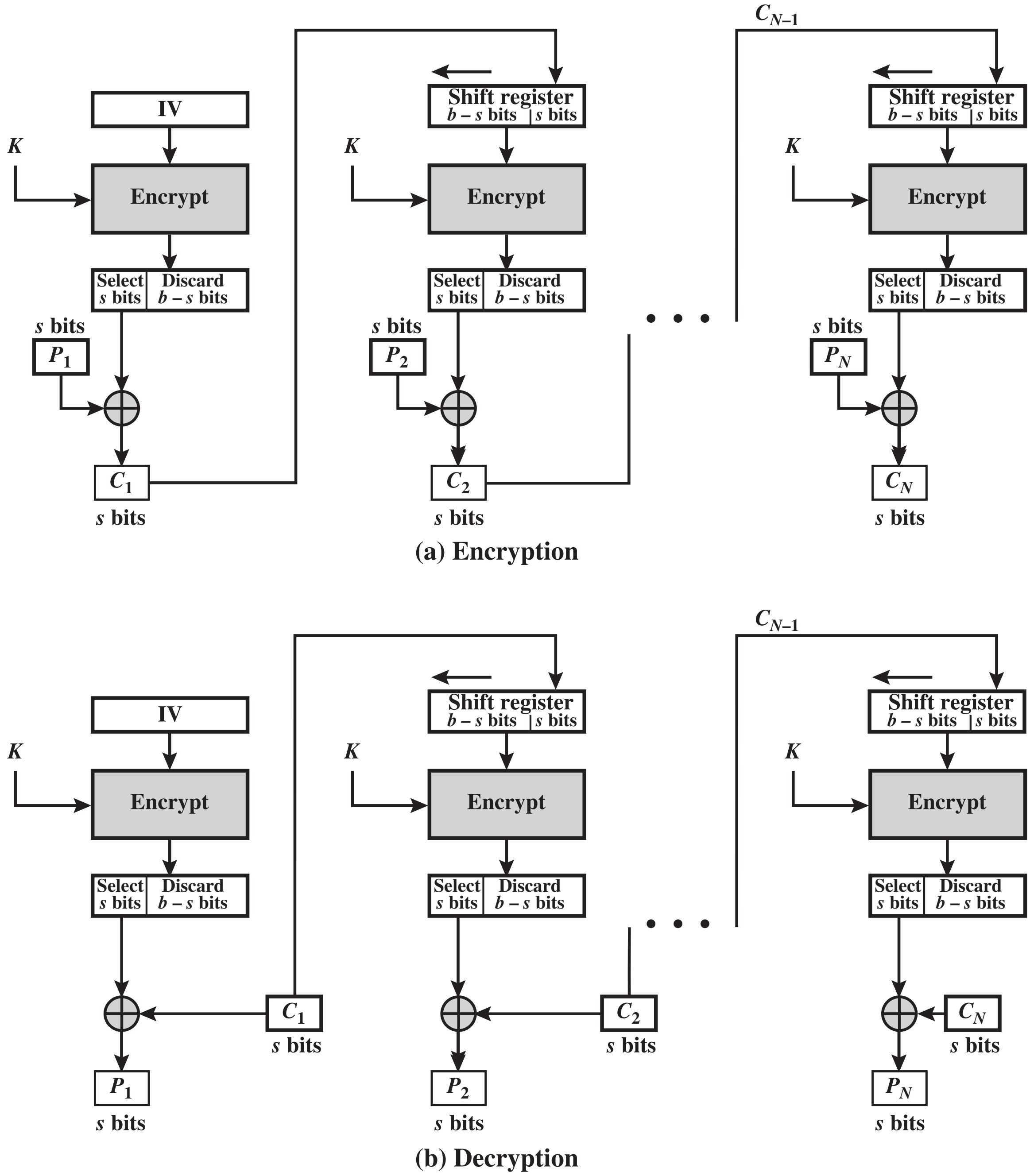


Figure 2.11 *s*-bit Cipher Feedback (CFB) Mode

Counter Mode

Although interest in the **counter mode (CTR)** has increased recently, with applications to ATM (asynchronous transfer mode) network security and IPSec (IP security), this mode was proposed early on (e.g., [DIFF79]).

Figure 2.12 depicts the CTR mode. A counter equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo

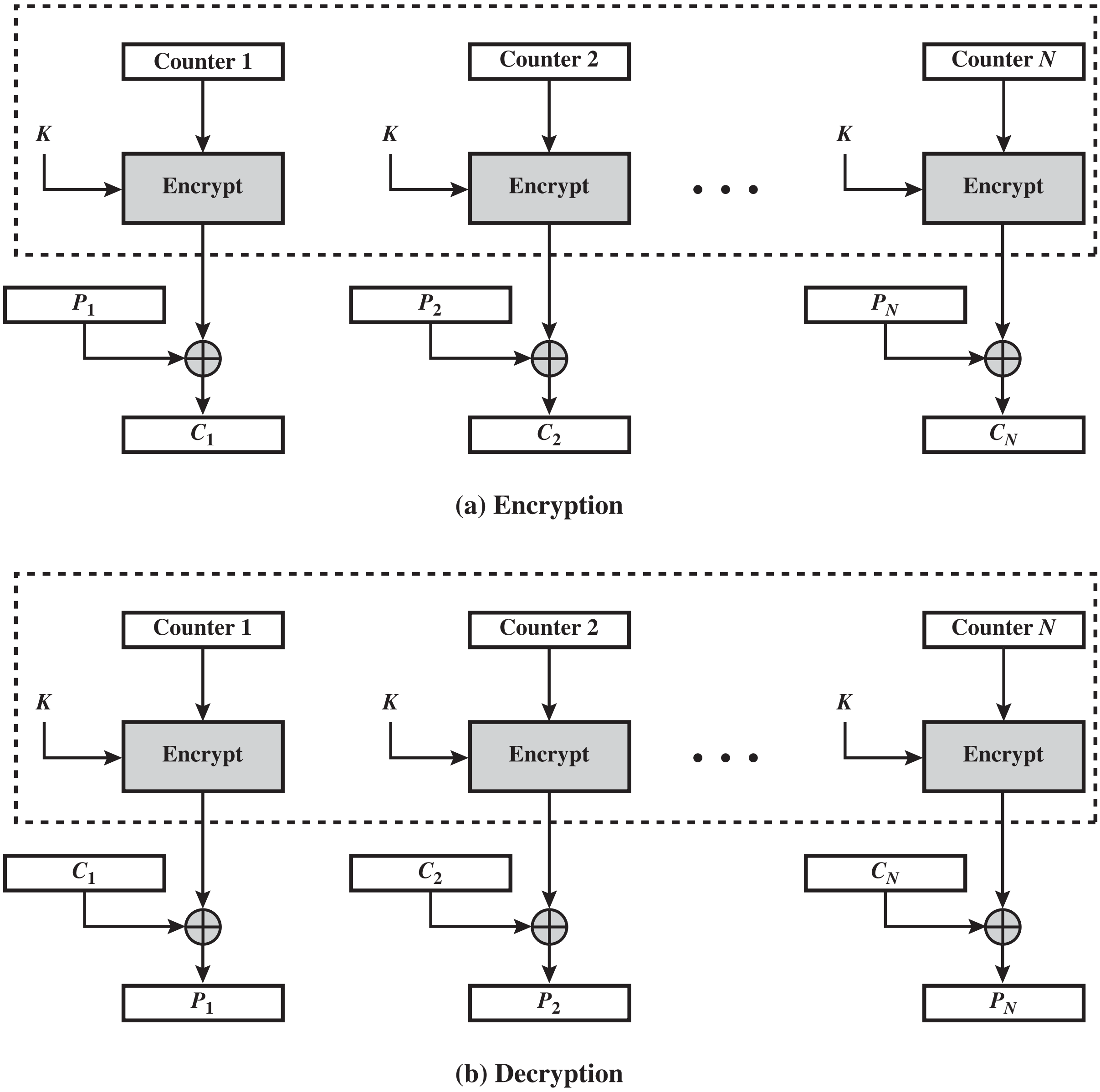


Figure 2.12 Counter (CTR) Mode

2^b, where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

[LIPM00] lists the following advantages of CTR mode.

- **Hardware efficiency:** Unlike the chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.

- **Software efficiency:** Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features (such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions) can be effectively utilized.
- **Preprocessing:** The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in Figure 2.12. When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.
- **Random access:** The *i*th block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block C_i cannot be computed until the $i - 1$ prior block are computed. There may be applications in which a ciphertext is stored, and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- **Provable security:** It can be shown that CTR is at least as secure as the other modes discussed in this section.
- **Simplicity:** Unlike ECB and CBC modes, CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.

2.6 RECOMMENDED READING AND WEB SITES

The topics in this chapter are covered in greater detail in [STAL11]. For coverage of cryptographic algorithms, [SCHN96] is an essential reference work; it contains descriptions of virtually every cryptographic algorithm and protocol published up to the time of the writing of the book. Another worthwhile and detailed survey is [MENE97]. A more in-depth treatment, with rigorous mathematical discussion, is [STIN06].

MENE97 Menezes, A.; van Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography.* Boca Raton, FL: CRC Press, 1997.

SCHN96 Schneier, B. *Applied Cryptography.* New York: Wiley, 1996.

STAL11 Stallings, W. *Cryptography and Network Security: Principles and Practice, Fifth Edition.* Upper Saddle River, NJ: Prentice Hall, 2011.

STIN06 Stinson, D. *Cryptography: Theory and Practice.* Boca Raton, FL: Chapman&Hall/CRC Press, 2006.

Recommended Web Sites:

- **AES home page:** NIST's page on AES. Contains the standard plus a number of other relevant documents.
- **AES Lounge:** Contains a comprehensive bibliography of documents and papers on AES with access to electronic copies.
- **Block Cipher Modes of Operation:** NIST page with full information on NIST-approved modes of operation.

2.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

Advanced Encryption Standard (AES)	Cryptography	keystream
block cipher	Data Encryption Standard (DES)	link encryption
brute-force attack	decryption	plaintext
cipher block chaining (CBC) mode	electronic codebook (ECB) mode	session key
cipher feedback (CFB) mode	encryption	stream cipher
ciphertext	end-to-end encryption	subkey
counter mode (CTR)	Feistel cipher	symmetric encryption
cryptanalysis	key distribution	triple DES (3DES)

Review Questions

2.1 What are the essential ingredients of a symmetric cipher?

2.2 What are the two basic functions used in encryption algorithms?

2.3 How many keys are required for two people to communicate via a symmetric cipher?

2.4 What is the difference between a block cipher and a stream cipher?

2.5 What are the two general approaches to attacking a cipher?

2.6 Why do some block cipher modes of operation only use encryption while others use both encryption and decryption?

2.7 What is triple encryption?

2.8 Why is the middle portion of 3DES a decryption rather than an encryption?

Problems

2.1 This problem uses a real-world example of a symmetric cipher, from an old U.S. Special Forces manual (public domain). The document, filename *SpecialForces.pdf*, is available at this book's Web site.

a. Using the two keys (memory words) *cryptographic* and *network security*, encrypt the following message:

> Be at the third pillar from the left outside the lyceum theatre tonight at seven. If you are distrustful bring two friends.

Make reasonable assumptions about how to treat redundant letters and excess letters in the memory words and how to treat spaces and punctuation. Indicate what your assumptions are. *Note:* The message is from the Sherlock Holmes novel, *The Sign of Four*.

b. Decrypt the ciphertext. Show your work.

c. Comment on when it would be appropriate to use this technique and what its advantages are.

2.2 Consider a very simple symmetric block encryption algorithm in which 32-bits blocks of plaintext are encrypted using a 64-bit key. Encryption is defined as

$$C = (P \oplus K_0) \boxplus K_1$$

where C = ciphertext, K = secret key, K_0 = leftmost 64 bits of K, K_1 = rightmost 64 bits of K, $\oplus$ = bitwise exclusive OR, and $\boxplus$ is addition mod 2^{64}.

a. Show the decryption equation. That is, show the equation for P as a function of C, K_0, and K_1.

b. Suppose and adversary has access to two sets of plaintexts and their corresponding ciphertexts and wishes to determine K. We have the two equations:

$$C = (P \oplus K_0) \boxplus K_1; C' = (P' \oplus K_0) \boxplus K_1$$

First, derive an equation in one unknown (e.g., K_0). Is it possible to proceed further to solve for K_0?

2.3 Perhaps the simplest "serious" symmetric block encryption algorithm is the Tiny Encryption Algorithm (TEA). TEA operates on 64-bit blocks of plaintext using a 128-bit key. The plaintext is divided into two 32-bit blocks (L_0, R_0), and the key is divided into four 32-bit blocks (K_0, K_1, K_2, K_3). Encryption involves repeated application of a pair of rounds, defined as follows for rounds i and i+1:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \boxplus \mathrm{F}(R_{i-1}, K_0, K_1, \delta_i)$$

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \boxplus \mathrm{F}(R_i, K_2, K_3, \delta_{i+1})$$

where F is defined as

$$\mathrm{F}(M, K_j, K_k, \delta_i) = ((\mathrm{M} << 4) \boxplus K_j) \oplus ((\mathrm{M} >> 5) \boxplus K_k) \oplus (\mathrm{M} \boxplus \delta_i)$$

and where the logical shift of x by y bits is denoted by $x << y$, the logical right shift of x by y bits is denoted by $x >> y$, and δ_i is a sequence of predetermined constants.

a. Comment on the significance and benefit of using the sequence of constants.

b. Illustrate the operation of TEA using a block diagram or flow chart type of depiction.

c. If only one pair of rounds is used, then the ciphertext consists of the 64-bit block (L_2, R_2). For this case, express the decryption algorithm in terms of equations.

d. Repeat part (c) using an illustration similar to that used for part (b).

2.4 Show that Feistel decryption is the inverse of Feistel encryption.

2.5 Consider a Feistel cipher composed of 16 rounds with block length 128 bits and key length 128 bits. Suppose that, for a given k, the key scheduling algorithm determines values for the first eight round keys, $k_1, k_2, \ldots, k_8$, and then sets

$$k_9 = k_8, k_{10} = k_7, k_{11} = k_6, \ldots, k_{16} = k_1$$

Suppose you have a ciphertext c. Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you, and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.)

2.6 For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 128-bit blocks of plaintext into 128-bit blocks of ciphertext. Let $\mathrm{EL}(k, m)$ denote the encryption of a 128-bit message m under a key k (the actual bit length of k is irrelevant). Thus,

$$\mathrm{EL}(k, [m_1 \oplus m_2]) = \mathrm{EL}(k, m_1) \oplus \mathrm{EL}(k, m_2) \text{ for all 128-bit patterns } m_1, m_2$$

Describe how, with 128 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k. (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 128 plaintext–ciphertext pairs to work with, and you have the ability to chose the value of the ciphertexts.)

2.7 Suppose you have a true random bit generator where each bit in the generated stream has the same probability of being a 0 or 1 as any other bit in the stream and that the bits are not correlated; that is, the bits are generated from identical independent distribution. However, the bit stream is biased. The probability of a 1 is $0.5 + \delta$ and the probability of a 0 is $0.5 - \delta$, where $0 < \delta < 0.5$. A simple deskewing algorithm is as follows: Examine the bit stream as a sequence of non-overlapping pairs. Discard all 00 and 11 pairs. Replace each 01 pair with 0 and each 10 pair with 1.

a. What is the probability of occurrence of each pair in the original sequence?
b. What is the probability of occurrence of 0 and 1 in the modified sequence?
c. What is the expected number of input bits to produce x output bits?
d. Suppose that the algorithm uses overlapping successive bit pairs instead of nonoverlapping successive bit pairs. That is, the first output bit is based on input bits 1 and 2, the second output bit is based on input bits 2 and 3, and so on. What can you say about the output bit stream?

2.8 Another approach to deskewing is to consider the bit stream as a sequence of non-overlapping groups of n bits each and output the parity of each group. That is, if a group contains an odd number of ones, the output is 1; otherwise the output is 0.

a. Express this operation in terms of a basic Boolean function.
b. Assume, as in the Problem 2.7, that the probability of a 1 is $0.5 + \delta$. If each group consists of 2 bits, what is the probability of an output of 1?
c. If each group consists of 4 bits, what is the probability of an output of 1?
d. Generalize the result to find the probability of an output of 1 for input groups of n bits.

2.9 What RC4 key value will leave S unchanged during initialization? That is, after the initial permutation of S, the entries of S will be equal to the values from 0 through 255 in ascending order.

2.10 RC4 has a secret internal state which is a permutation of all the possible values of the vector **S** and the two indices i and j.

a. Using a straightforward scheme to store the internal state, how many bits are used?
b. Suppose we think of it from the point of view of how much information is represented by the state. In that case, we need to determine how may different states

there are, then take the log to the base 2 to find out how many bits of information this represents. Using this approach, how many bits would be needed to represent the state?

2.11 Alice and Bob agree to communicate privately via e-mail using a scheme based on RC4, but they want to avoid using a new secret key for each transmission. Alice and Bob privately agree on a 128-bit key k. To encrypt a message m consisting of a string of bits, the following procedure is used.

1. Choose a random 80-bit value v
2. Generate the ciphertext $c = \text{RC4}(v \| k) \oplus m$
3. Send the bit string $(v \| c)$

a. Suppose Alice uses this procedure to send a message m to Bob. Describe how Bob can recover the message m from $(v \| c)$ using k.

b. If an adversary observes several values $(v_1 \| c_1), (v_2 \| c_2), \ldots$ transmitted between Alice and Bob, how can he/she determine when the same key stream has been used to encrypt two messages?

2.12 With the ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C_1 (Figure 2.10) obviously corrupts P_1 and P_2.

a. Are any blocks beyond P_2 affected?

b. Suppose that there is a bit error in the source version of P_1. Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?

2.13 Is it possible to perform encryption operations in parallel on multiple blocks of plaintext in CBC mode? How about decryption?

2.14 Suppose an error occurs in a block of ciphertext on transmission using CBC. What effect is produced on the recovered plaintext blocks?

2.15 CBC-Pad is a block cipher mode of operation used in the RC5 block cipher, but it could be used in any block cipher. CBC-Pad handles plaintext of any length. The ciphertext is longer than the plaintext by at most the size of a single block. Padding is used to assure that the plaintext input is a multiple of the block length. It is assumed that the original plaintext is an integer number of bytes. This plaintext is padded at the end by from 1 to *bb* bytes, where *bb* equals the block size in bytes. The pad bytes are all the same and set to a byte that represents the number of bytes of padding. For example, if there are 8 bytes of padding, each byte has the bit pattern **00001000**. Why not allow zero bytes of padding? That is, if the original plaintext is an integer multiple of the block size, why not refrain from padding?

2.16 Padding may not always be appropriate. For example, one might wish to store the encrypted data in the same memory buffer that originally contained the plaintext. In that case, the ciphertext must be the same length as the original plaintext. A mode for that purpose is the ciphertext stealing (CTS) mode. Figure 2.13a shows an implementation of this mode.

a. Explain how it works.

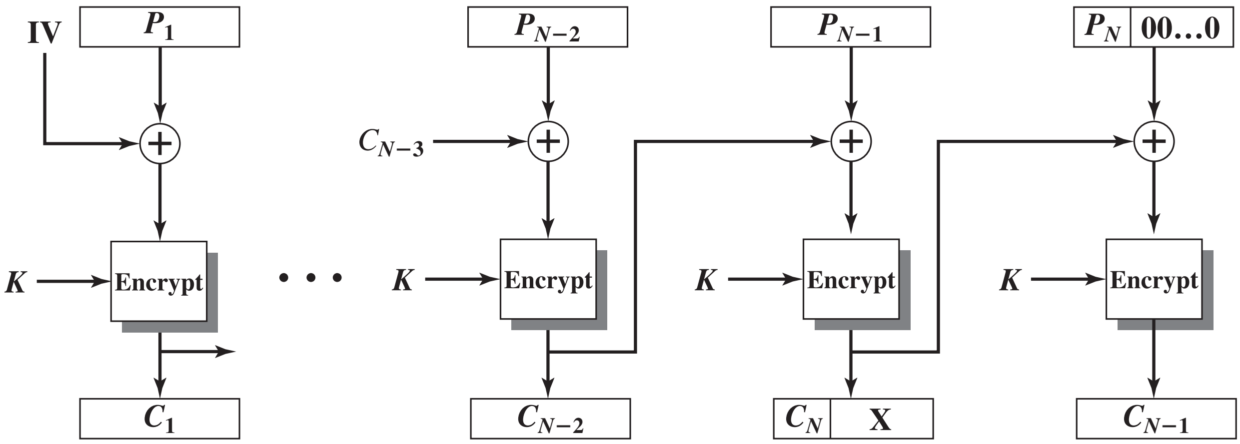
b. Describe how to decrypt C_{n-1} and C_n.

2.17 Figure 2.13b shows an alternative to CTS for producing ciphertext of equal length to the plaintext when the plaintext is not an integer multiple of the block size.

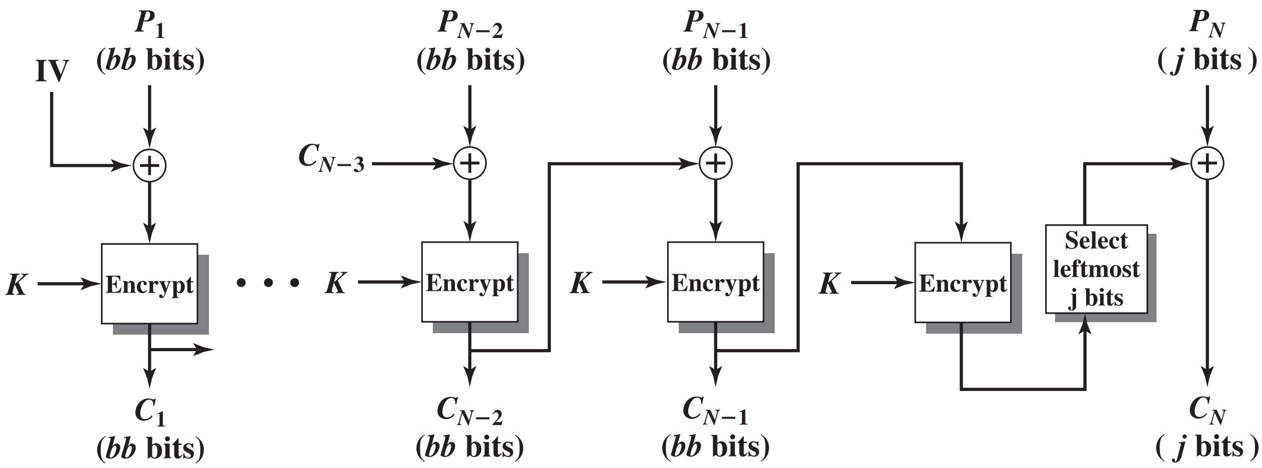
a. Explain the algorithm.

b. Explain why CTS is preferable to this approach illustrated in Figure 2.13b.

2.18 If a bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode, how far does the error propagate?



(a) Cipheretext stealing mode



(b) Alternative method

Figure 2.13 Block Cipher Modes for Plaintext not a Multiple of Block Size