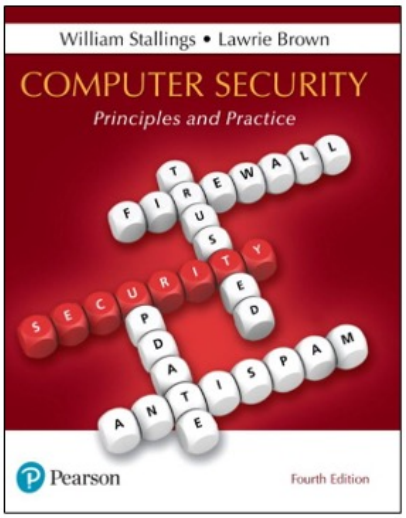
Computer Security: Principles and Practice

Fourth Edition



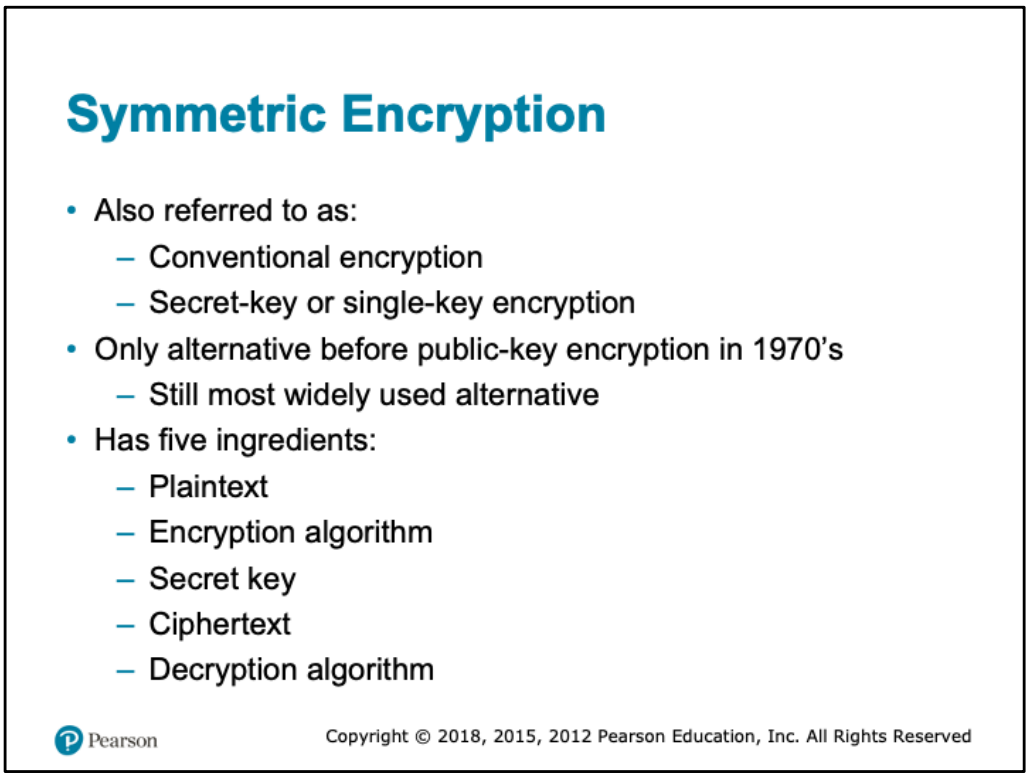
Chapter 20

Symmetric Encryption and Message Confidentiality

If this PowerPoint presentation contains mathematical equations, you may need to check that your computer has the following installed:
1) MathType Plugin
2) Math Player (free versions available)
3) NVDA Reader (free versions available)

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. 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. Next, the chapter introduces symmetric stream encryption and describes the widely used stream cipher RC4. We then examine the application of these algorithms to achieve confidentiality.



At this point the reader should review Section 2.1 . Recall that a symmetric encryption scheme has five ingredients (Figure 2.1):

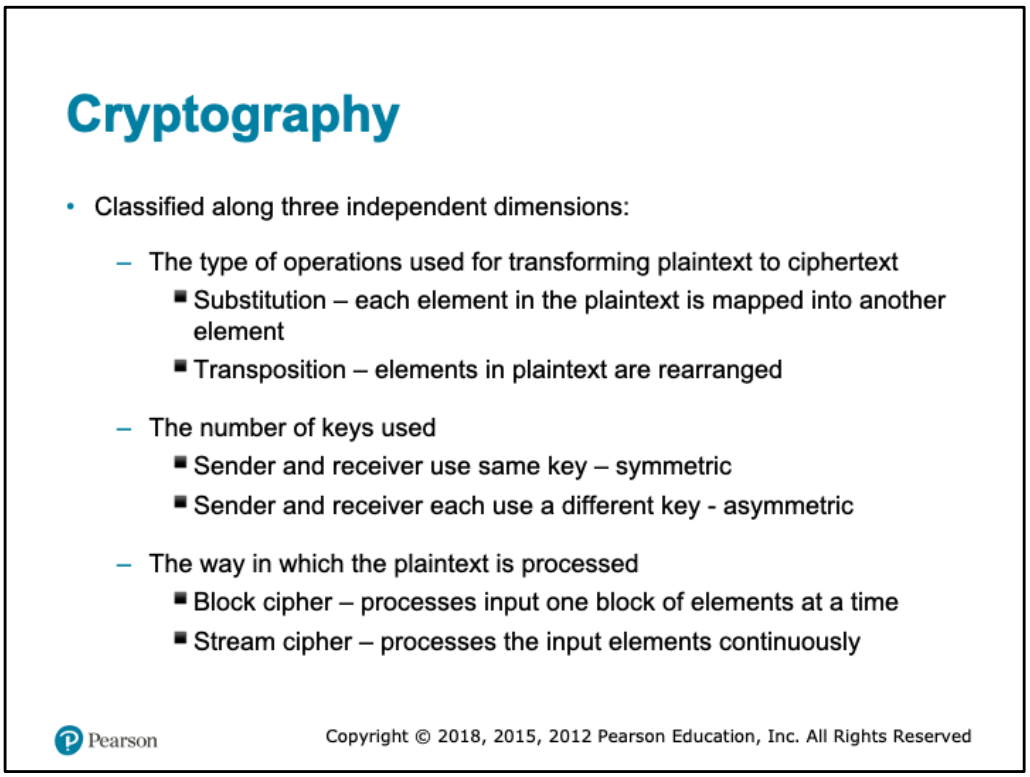
• **Plaintext:** This is the original message or data that is fed into the algorithm as input.

• **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.

• **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.



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 (i.e., 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.

Table 20.1 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

Pearson

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 20.1 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.

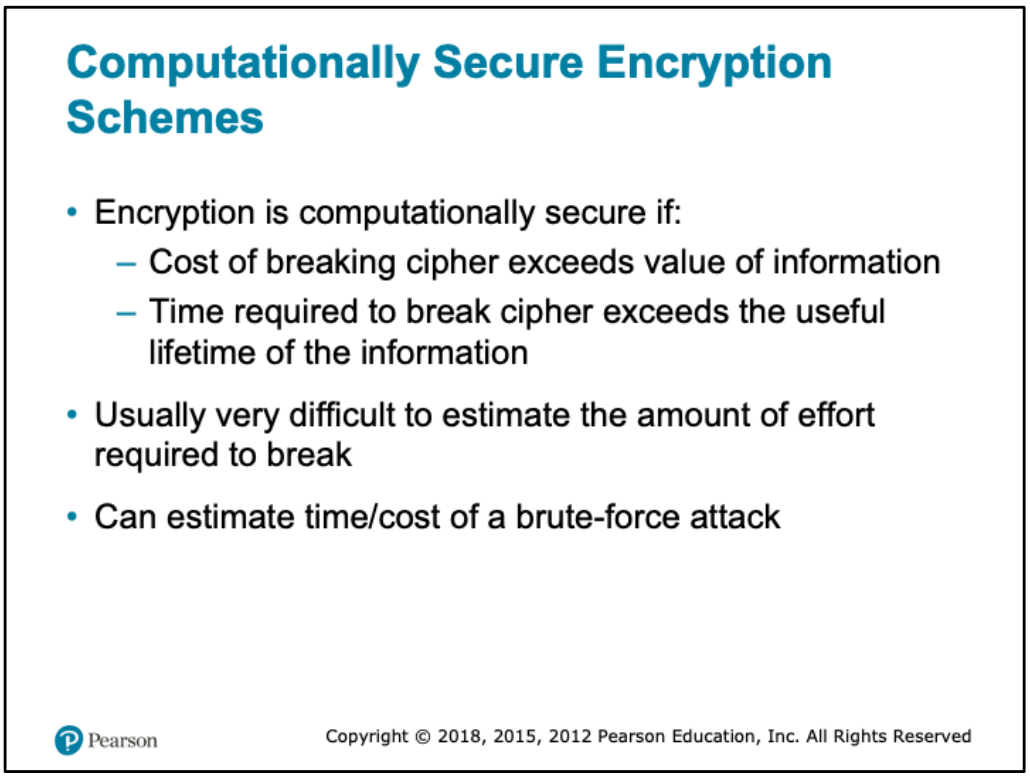
The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may 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 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 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 20.1 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:

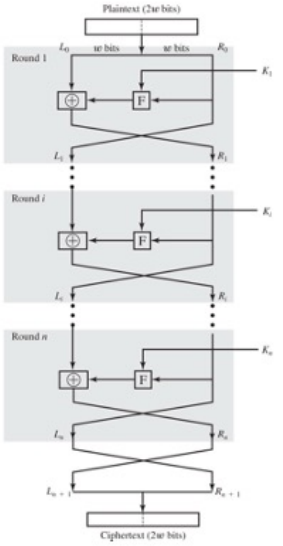
• The cost of breaking the cipher exceeds the value of the encrypted information.

• 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. This type of attack is discussed in Section 2.1 .

Figure 20.1 Classical Feistel Network



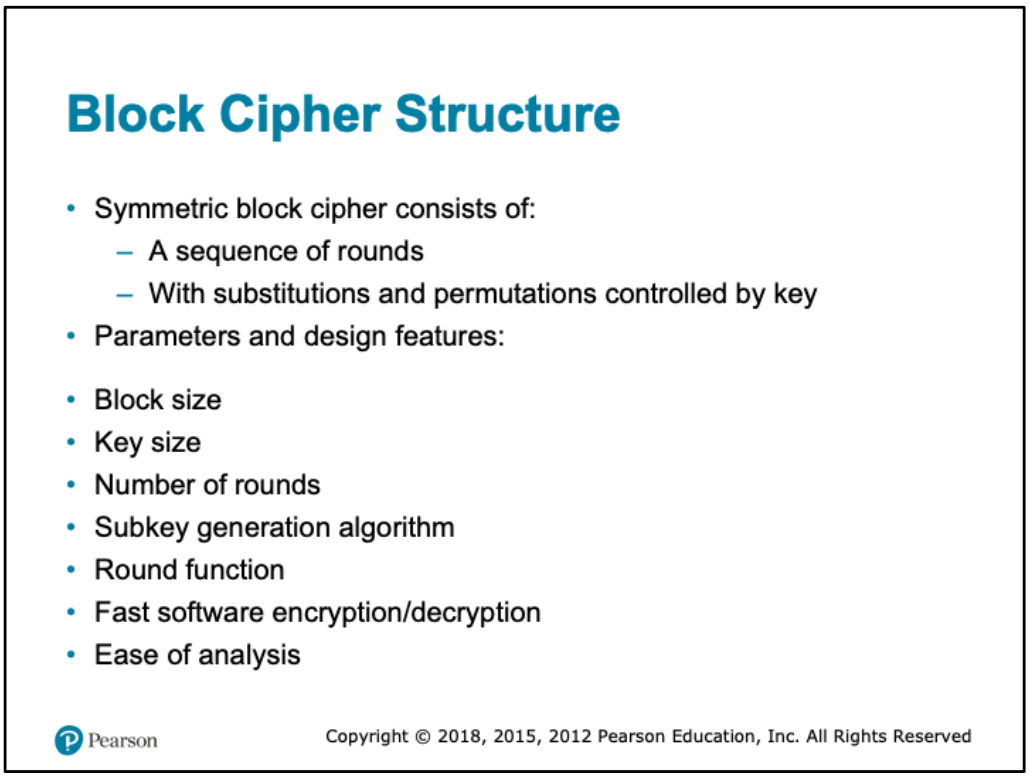
Pearson

Many symmetric block encryption algorithms, including DES, have a structure first described by Horst Feistel of IBM in 1973 [FEIS73] and shown in Figure 20.1 . The inputs to the encryption algorithm are a plaintext block of length 2 w bits and a key K . The plaintext block is divided into two halves, L 0 and R 0 . The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs L i-1 and R i-1 , derived from the previous round, as well as a subkey Ki , 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.

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 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.

Round 1. The plaintext, 2 w bits, is divided into 2 parts, L sub 0 and R sub 0, each of w bits. The transformed R sub 0 into F, with input K sub 1, is added to L sub 0. The output is L sub 1 and R sub 1. Ellipsis. Round i. the input is transformed and added the partial output F with input K sub i, to get output L sub i and R sub i. Round n. The

input is transformed and added the partial output F with input K sub n, to get output L sub n and R sub n. The output is crossed again to get L sub start expression n + 1 end expression and R sub start expression n + 1 end expression. The output is combined to get ciphertext of 2 bits.



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.

Data Encryption Standard (DES)

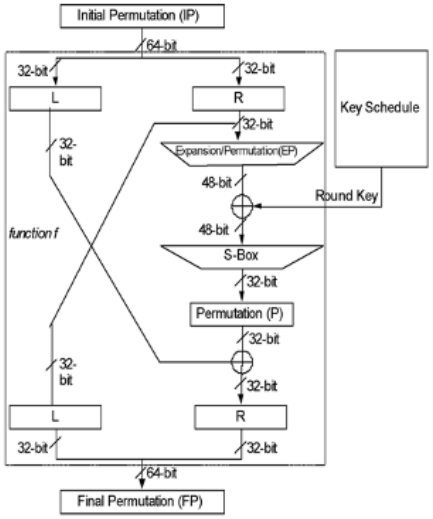
- Most widely used encryption scheme
- Adopted in 1977 by National Bureau of Standards (Now NIST)
- FIPS PUB 46
- Algorithm is referred to as the Data Encryption Algorithm (DEA)
- Minor variation of the Feistel network

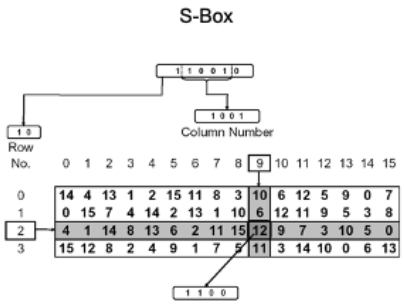
The most commonly used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as FIPS 46 *(Data Encryption Standard,* January 1977). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

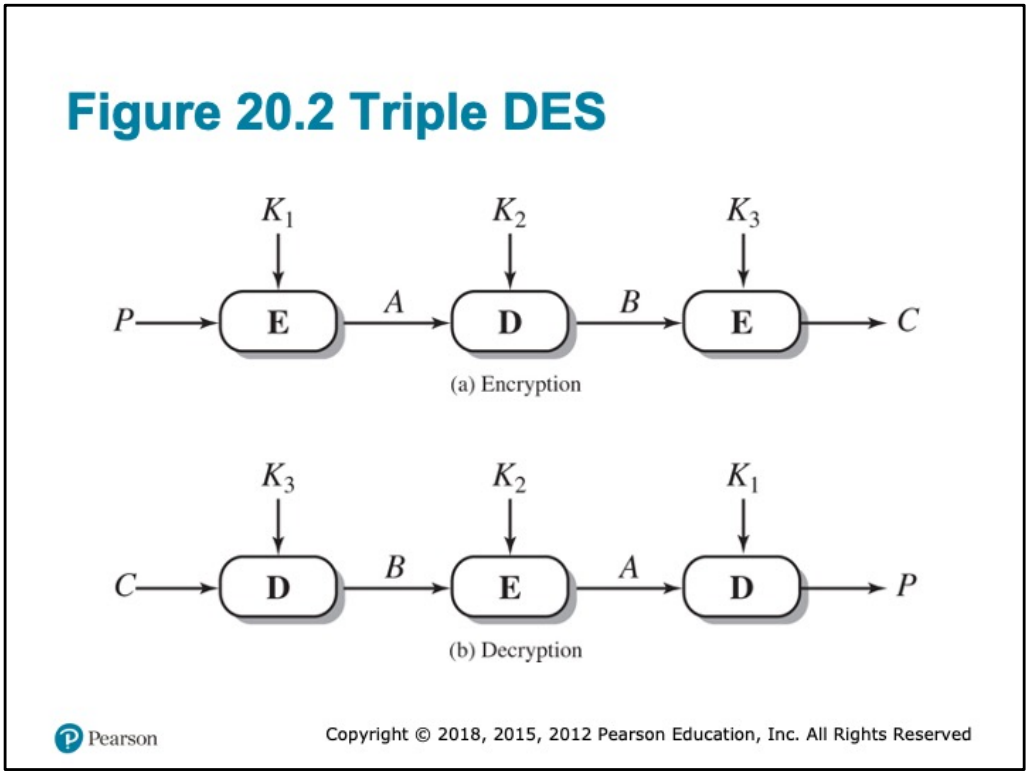
The DES algorithm can be described as follows. The plaintext is 64 bits in length and the key is 56 bits in length; longer plaintext amounts are processed in 64-bit blocks. The DES structure is a minor variation of the Feistel network shown in Figure 20.1 . There are 16 rounds of processing. From the 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 sixteenth and last iteration.

Data Encryption Standard (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 PUB 46-3. 3DES uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence (see Figure 20.2a):

C = E(K3, D(K2, E(K1, P)))

where: C = ciphertext; P = plaintext; E[K, X] = encryption of X using key K, and D[K, Y] = decryption of Y using key K. Decryption is simply the same operation with the keys reversed (Figure 20.2b):

P = D(K1, E(K2, D(K3, 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 = E(K1, D(K1, E(K1, P))) = 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 K1 = K3; 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.

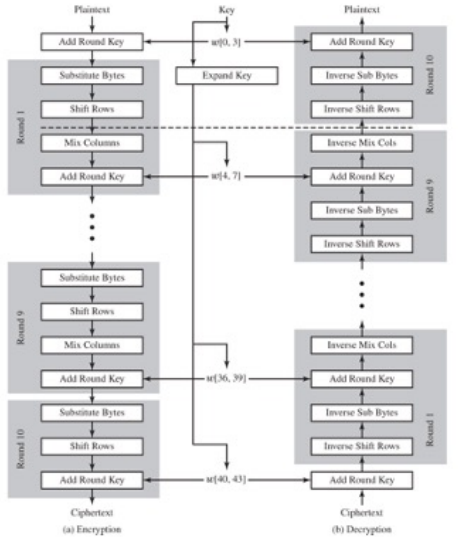
• It is anticipated that 3DES and the Advanced Encryption Standard (AES) will coexist as FIPS-approved algorithms, allowing for a gradual transition to AES. It is easy to see that 3DES is a formidable algorithm. Because the underlying cryptographic algorithm is DEA, 3DES can claim the same resistance to cryptanalysis based on the algorithm as is claimed for DEA. Further, 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.

The encryption and decryption process involves running encryption and decryption process three times. During encryption, the keys k 1 to k 3 is used to encrypt a message P to cipher text, C. The encryption, decryption, and encryption processes are run in series with key K 1 to k 3 as inputs to get outputs, A, B, and C, respectively. During decryption, the keys k 3 to k 1 is used to encrypt a cipher C to message text, P. The decryption, encryption, and decryption processes are run in series with key K 3 to k 1 as inputs to get outputs, B, A, and P, respectively.

Figure 20.3 AES Encryption and Decryption

Not in Syllabus Additional Content



The Advanced Encryption Standard (AES) was issued as a federal information processing standard FIPS 197 (Advanced Encryption Standard, November 2001). It is intended to replace DES and triple DES with an algorithm that is more secure and efficient.

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 20.3 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 4 bytes and the total key schedule is 44 words for the 128-bit key. The ordering of bytes within a matrix is by column. So, for example, the first 4 bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second 4 bytes occupy the second column, and so on. Similarly, the first 4 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:

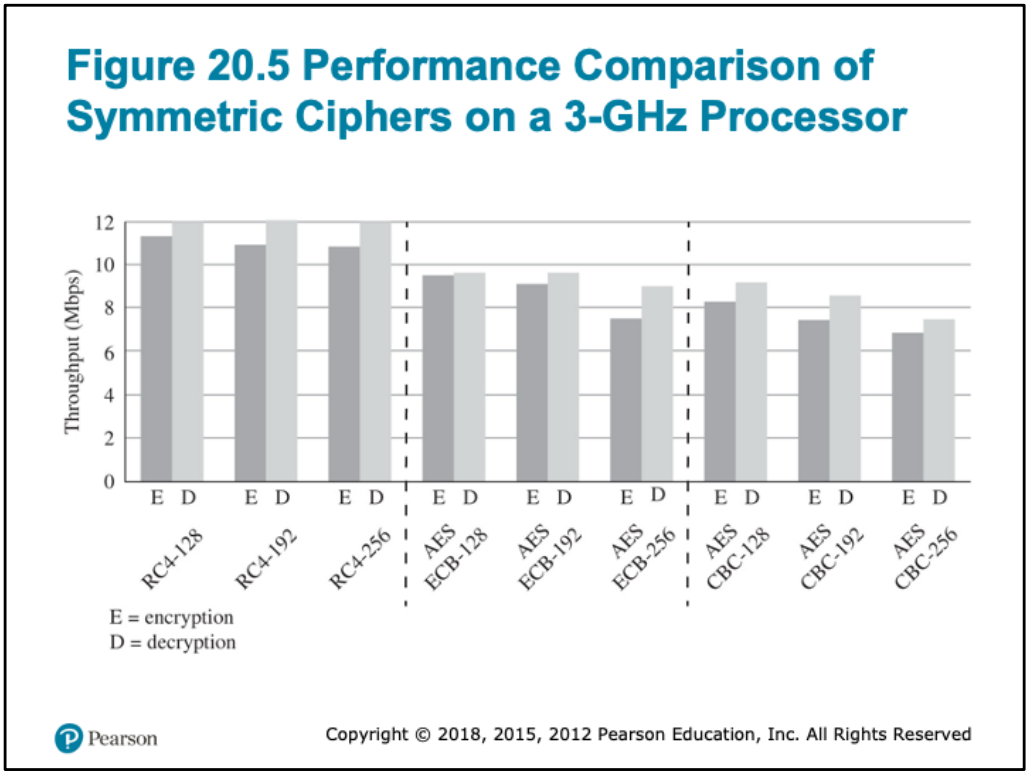
• Substitute Bytes: Uses a table, referred to as an S-box, to perform a byte-by- byte substitution of the block

• 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

The encryption process is as follows. Paintext, add round key with key left bracket 0, 3 right bracket. Round 1. Substitute bytes, shift rows, mix columns, add round key w left bracket 4, 7 right bracket. ellipsis. Round 9. Substitute bytes, shift rows, mix columns, add round key w left bracket 36, 39 right bracket. Round 10. Substitute bytes, shift rows, add round key w left bracket 40, 43 right bracket. output, cipher text. The decryption process is as follows. Cipher text. Add round key w left bracket 40, 43 right bracket. Round 1. Inverse shift, inverse sub bytesm add round key, and inverse mix C o ls. Ellipsis. Round 9. Inverse shift rows, inverse sub bytes, add round key w left bracket 4, 7 right bracket, inverse mix C o ls. Round 10. Inverse shift rows, inverse sub bytes, and add round key w left bracket 0, 3 right bracket. The output is plain text. The key expanded to w left parenthesis 4, 7 right bracket, w left bracket 36, 39 right bracket, and w left bracket 40, 43 right bracket.



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 10100 [ROBS95]. 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 SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards that have been defined for communication between Web browsers and servers. It is also used in the WEP (Wired Equivalent Privacy) 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 simply 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 2.3b) 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.

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 follows:

```
 /* 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. 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 20.5 illustrates the RC4 logic.

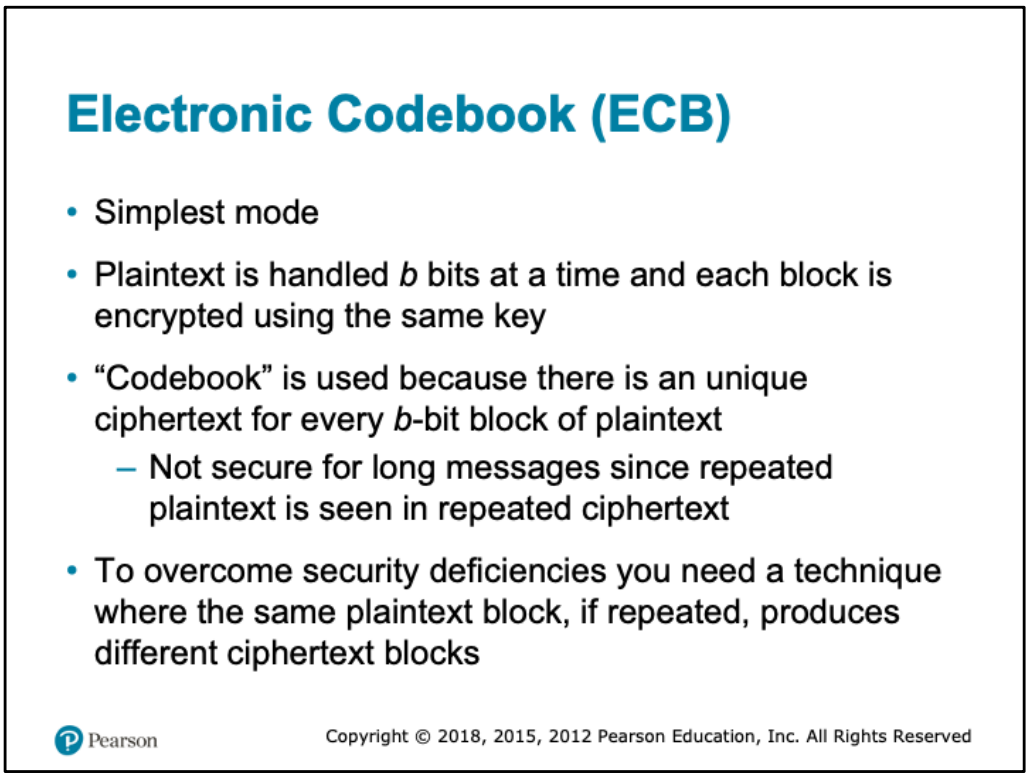
The vertical axis plots throughput, in M b p s, and ranges from 0 to 12 in increments of 2. The horizontal axis is marked at R C 4, A E S E C B, A E S C B C, from 128 to 256. The data listed are as follows. R C 4 128, E = 11, D = 12. R C 4 192, E = 10.5, D = 12. R C 4 256, E = 10.5, D = 12. A E S E C B 128, E = 9.5, D = 9.6. A E S E C B 192, E = 8.9, D = 9.3. A E S E C B 256, E = 7.6, D = 8.5. A E S C B C 128, E = 8.1, D = 8.9. A E S C B C 192, E = 7.6, D = 8.2. A E S C B C 256, E = 6.5, D = 7.2. E = encryption. D = decryption.

Table 20.3 Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Code book (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed Requirements

Pearson

A symmetric block cipher processes one block of data at a time. In the case of DES and 3DES, the block length is 64 bits. For longer amounts of plaintext, it is necessary to break the plaintext into 64-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 SP 800-38A (*Recommendation for Block Cipher Modes of Operation: Methods and Techniques,* December 2001). The five modes are intended to cover virtually all 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 modes are summarized in Table 20.3, and the most important are described briefly in the remainder of this section.

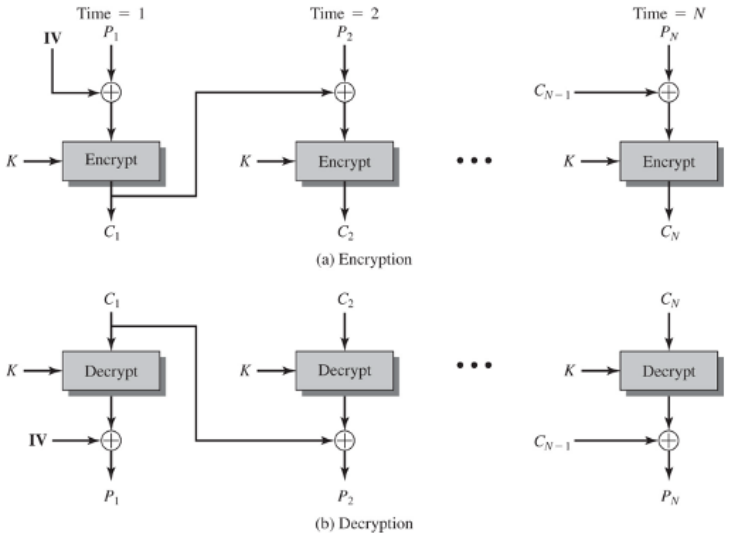


The simplest way to proceed is what is known as electronic codebook (ECB) mode, in which plaintext is handled *b* bits at a time and each block of plaintext is encrypted using the same key (Figure 2.3a). The term *codebook* is used because, for a given key, there is a unique ciphertext for every *b* -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.

Figure 20.7 Cipher Block Chaining (CBC) Mode



In the cipher block chaining (CBC) mode (Figure 20.7), 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 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.

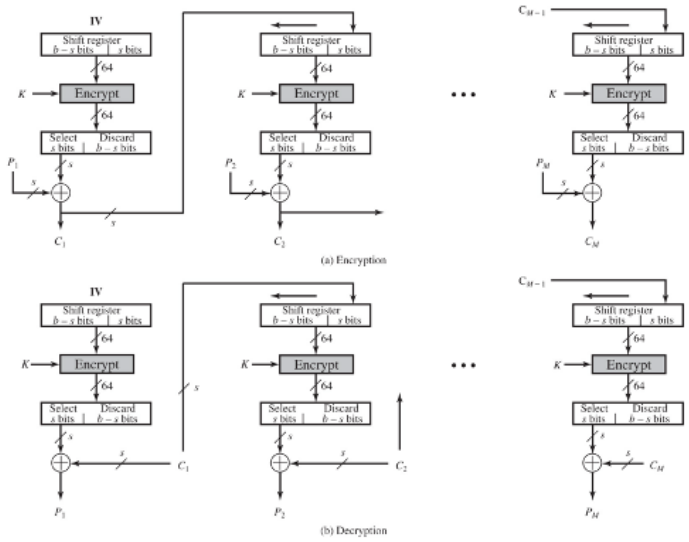
To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext.

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.

The input to the encryption algorithm is the X O R of the current plain text block and the preceding cipher text block, the same key is used for each block. An initialization vector I V is X O Red with the block of plaintext. The output is C sub 1, which is added to the input P sub 2 in the second encryption round. The output is C sub 2, which is added to the next encryption round palintext input. The n th round encryption used C sub start expression N minus 1 end expression and P sub N to get output C sub N. On decryption, the I V is X O Red with the output of the decryption algorithm to recover the first block of plaintext. The cipher text C sub 1 is decoded to plaintext P sub 1. The cipher text C sub 1 is then added to the cipher text C sub 2 to decrypt with K to get output P sub 2. For the n round decryption, C sub N is used as input, C sub start expression N minus 1 end expression is added with K to get output P sub N.

Figure 20.8 s-bit Cipher Feedback (CFB) 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 20.8 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 $S_s(X)$ be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus S_s[E(K, IV)]$$

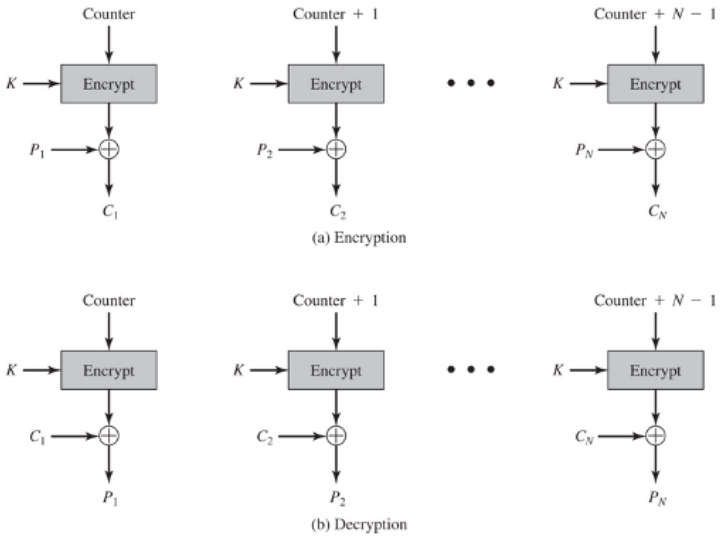
Therefore

$$P_1 = C_1 \oplus S_s[E(K, IV)]$$

The same reasoning holds for subsequent steps in the process.

The unit of transmission is s bits, a common value is s = 8. As with C B C, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. The plaintext is divided into segments of s bits. The input to the encryption function is a b bit shift register that is initially set to some initialization vector. The leftmost, most significant, s bits of the output of the encryption function are X O Red with the first segment of plaintext P 1 to produce the first unit of ciphertext C 1, which is then transmitted. The contents of the shift register are shifted left by s bits, and C 1 is placed in the least significant, s bits of the shift register. For decryption, the same scheme is used, except that the received ciphertext unit is X O Red with the output of the encryption function to produce the plaintext unit

Figure 20.9 Counter (CTR) 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 20.9 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 2b, 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 three 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.

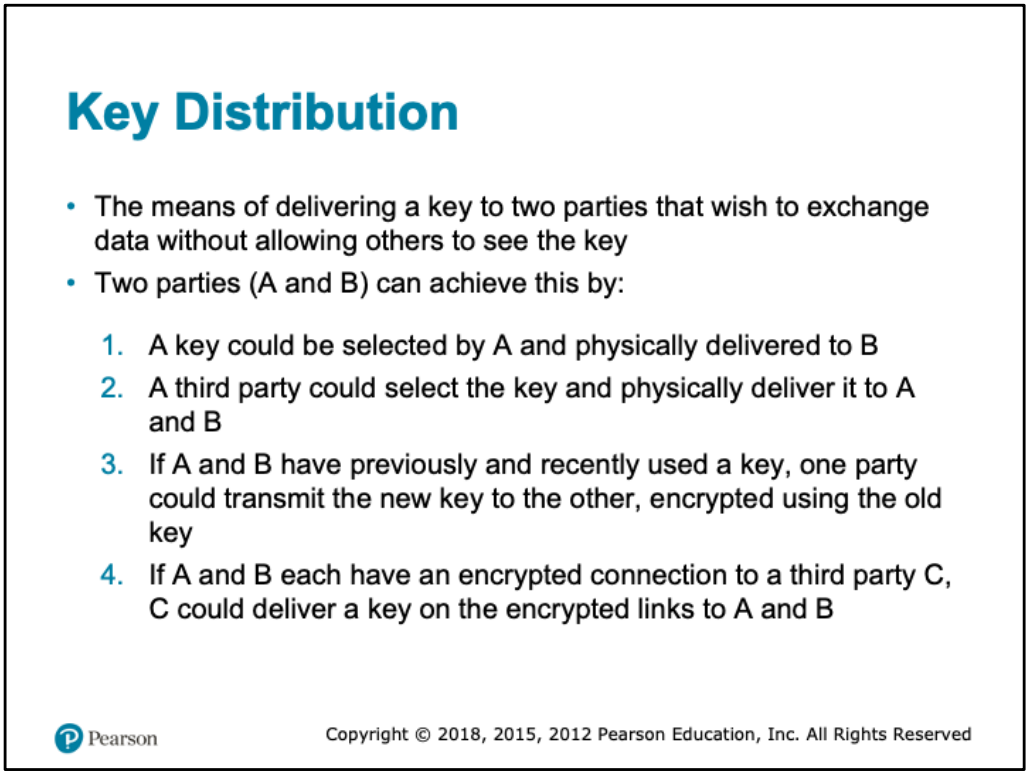
• Preprocessing: The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in Figure 20.8. 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 Ci 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.

Counter 1 is encrypted and given as input to X O R with plaintext P sub 1 to get the final value C sub 1. Counter + 1 is encrypted and given as input to X O R with plaintext P sub 2 to get the final value C sub 2. For N round enryption, counter + N minus 1 is encrypted and given as input to X O R with plain text P sub N to get the output, C sub N. Decryption. Counter is encrypted and given as input to X O R with cipher text C sub 1 to get plain text P sub 1. Counter + 1 is encrypted and given as input to X O R with cipher text C sub 2 to get the final value P sub 2. For N round decryption, counter + N minus 1 is encrypted and given as input to X O R with cipher text C sub N to get the output, P sub N.



For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the key distribution technique, a term that refers to the means of delivering a key to two parties that wish to exchange data, without allowing others to see the key. Key distribution can be achieved in a number of ways. For two parties A and B,

1. A key could be selected by A and physically delivered to B.

2. A third party could select the key and physically deliver it to A and B.

3. If A and B have previously and recently used a key, one party could transmit the new key to the other, encrypted using the old key.

4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For **link encryption** between two directly-connected devices, this is a reasonable requirement, because each link encryption device is only going to be exchanging data with its partner on the other end

of the link. However, for **end-to-end encryption** over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys, supplied dynamically. The problem is especially difficult in a wide area distributed system.

Option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys are revealed. Even if frequent changes are made to the link encryption keys, these should be done manually. To provide keys for end-to-end encryption, option 4 is preferable.

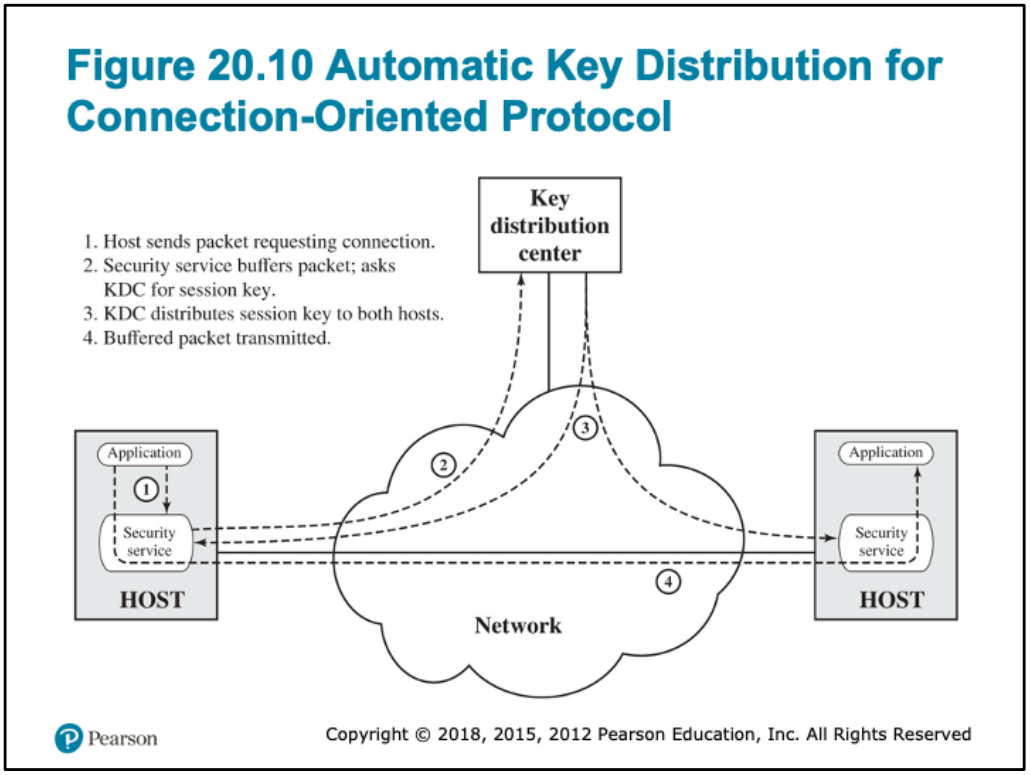


Figure 20.10 illustrates an implementation that satisfies option 4 for end-to-end encryption. In the figure, link encryption is ignored. This can be added, or not, as required. For this scheme, two kinds of keys are identified:

• Session key: When two end systems (hosts, terminals, etc.) wish to communicate, they establish a logical connection (e.g., virtual circuit). For the duration of that logical connection, all user data are encrypted with a one-time session key. At the conclusion of the session, or connection, the session key is destroyed.

• Permanent key: A permanent key is a key used between entities for the purpose of distributing session keys. The configuration consists of the following elements:

• Key distribution center: The key distribution center (KDC) determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the KDC provides a one-time session key for that connection.

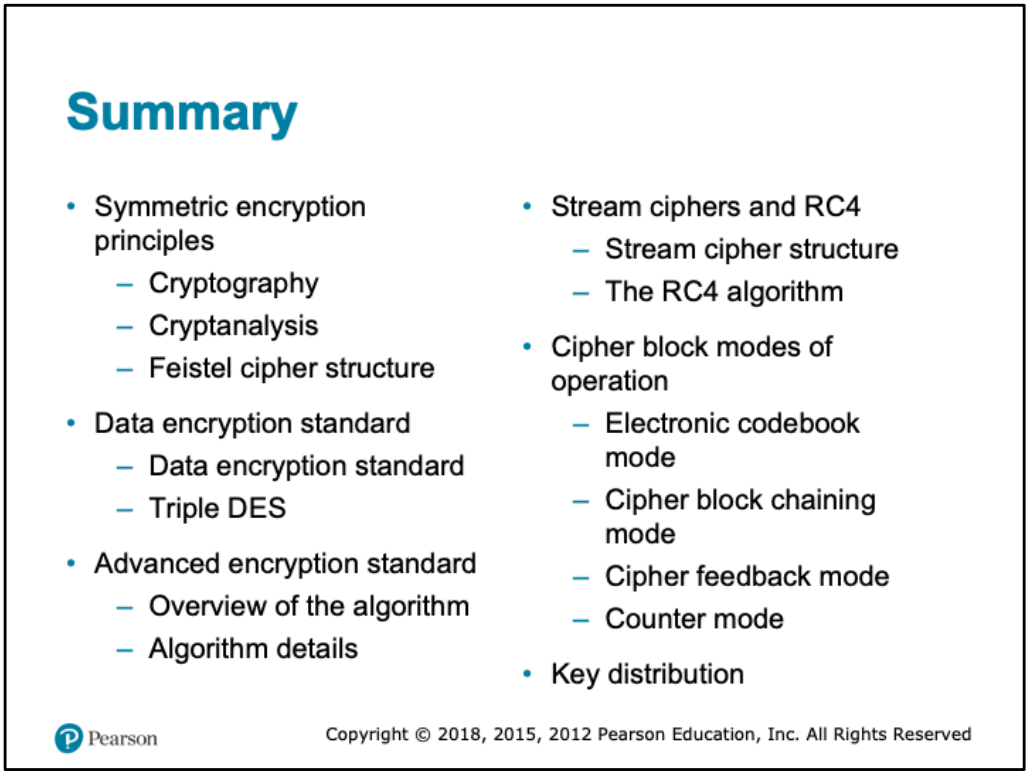
• Security service module (SSM): This module, which may consist of functionality at one protocol layer, performs end-to-end encryption and obtains session keys on behalf of users.

The steps involved in establishing a connection are shown in Figure 20.10 . When one host wishes to set up a connection to another host, it transmits a connection request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

Another approach to key distribution uses public-key encryption, which is discussed in Chapter 21 .

The steps are as follows. 1. Application through security service in the host sends packet requesting connection. 2. Security service buffers packet, asks K D C for session key. 3. K D C distributes session key to both hosts. 4. Buffered packet transmitted.



Chapter 20 summary.

Copyright