UNIT-2

Symmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the same key. It is also known as conventional encryption, Symmetric encryption, secret key or single-key encryption.

TRADITIONAL BLOCK CIPHER STRUCTURE

Stream Ciphers and Block Ciphers

A stream cipher is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher.

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

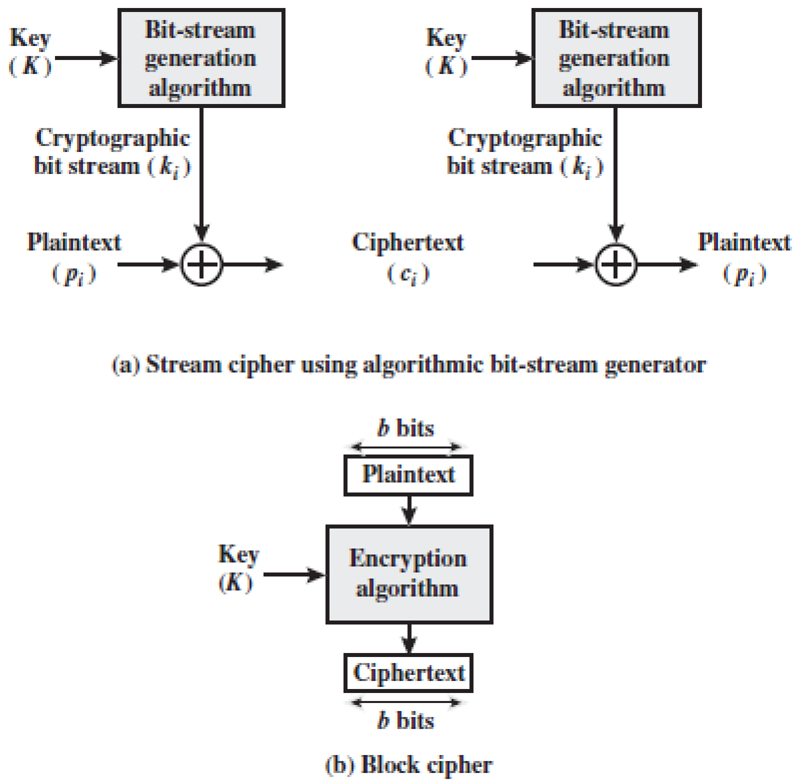
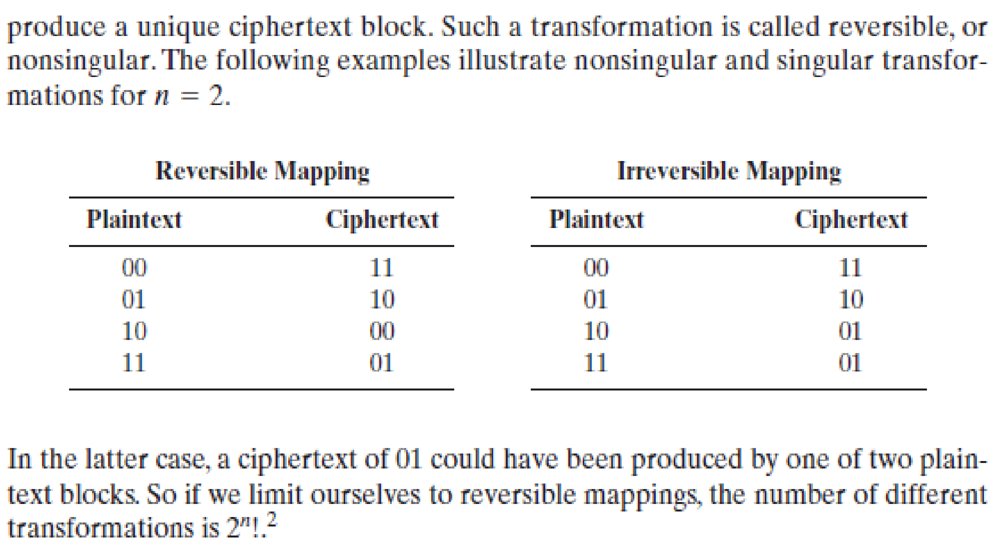

Key (K)
Bit-stream generation algorithm
Cryptographic bit stream (k_i)
Plaintext (p_i)
Key (K)
Bit-stream generation algorithm
Cryptographic bit stream (k_i)
Ciphertext (c_i)
Plaintext (p_i)
(a) Stream cipher using algorithmic bit-stream generator
b bits
Plaintext
Key (K)
Encryption algorithm
Ciphertext
b bits
(b) Block cipher

Figure 3.1 Stream Cipher and Block Cipher

A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits. There are 2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must



produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformations for $n = 2$.

Reversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	01
11	01

In the latter case, a ciphertext of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is $2^n!$.[2]

<u>THE FEISTEL CIPHER:</u>

- Feistel cipher is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers.
- In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:
- Substitution: Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.
- Permutation: A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

FEISTEL CIPHER STRUCTURE:

The left-hand side of Figure 3.3 depicts the structure proposed by Feistel.

- The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key . 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 *Li*-1 and *Ri*-1 derived from the previous round, as well as a subkey *Ki* derived from the overall K. In general, the subkeys Ki are different from K and from each other.

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

Following this substitution a Permutation is performed that consists of the interchange of the two halves of the data.

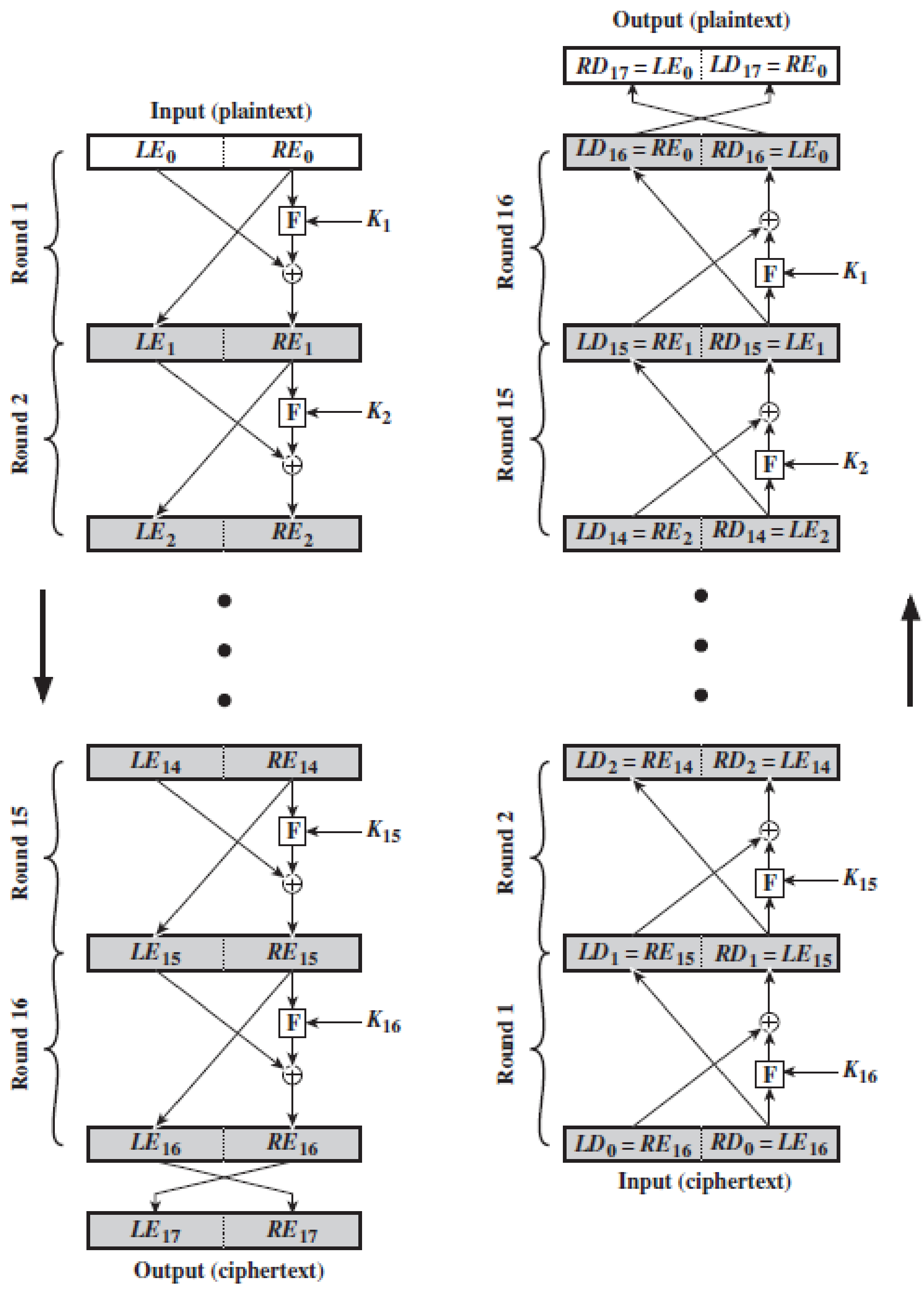


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

encryption process	On the decryption side,
$LE_{16} = RE_{15}$ $RE_{16} = LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})$ **in general** $LE_i = RE_{i-1}$ $RE_i = LE_{i-1} \oplus \mathrm{F}(RE_{i-1}, K_i)$	$LD_1 = RD_0 = LE_{16} = RE_{15}$ $RD_1 = LD_0 \oplus \mathrm{F}(RD_0, K_{16})$ $= RE_{16} \oplus \mathrm{F}(RE_{15}, K_{16})$ $= [LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})] \oplus \mathrm{F}(RE_{15}, K_{16})$

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

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

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

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

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

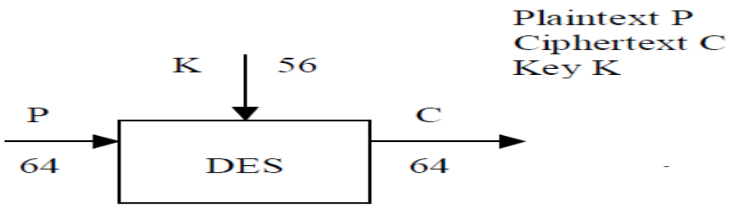
Round function F: Again, greater complexity generally means greater resistance to cryptanalysis.

DATA ENCRYPTION STANDARD (DES):

DES is a Symmetric-key algorithm for the encryption of electronic data.

Data Encryption Standard (DES) is a widely-used method of data encryption using a private (secret) key

DES applies a 56-bit key to each 64-bit block of data. The process can run in several modes and involves 16 rounds or operations.



Overall Structure

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher.

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases.

1. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
2. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput.
3. Finally, the preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit cipher text. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher,

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

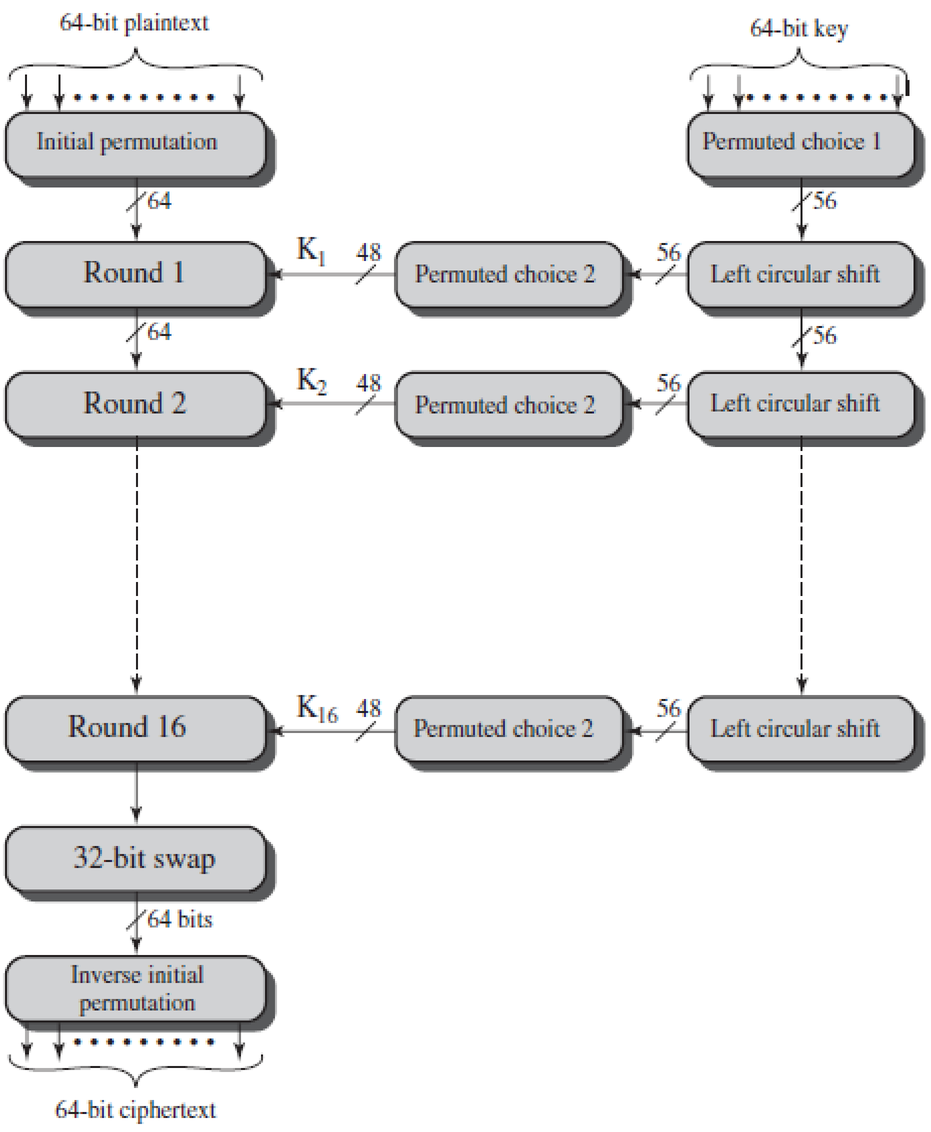


Figure 3.5 General Depiction of DES Encryption Algorithm

Details of Single Round

Below figure shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram.

- The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right).
- As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

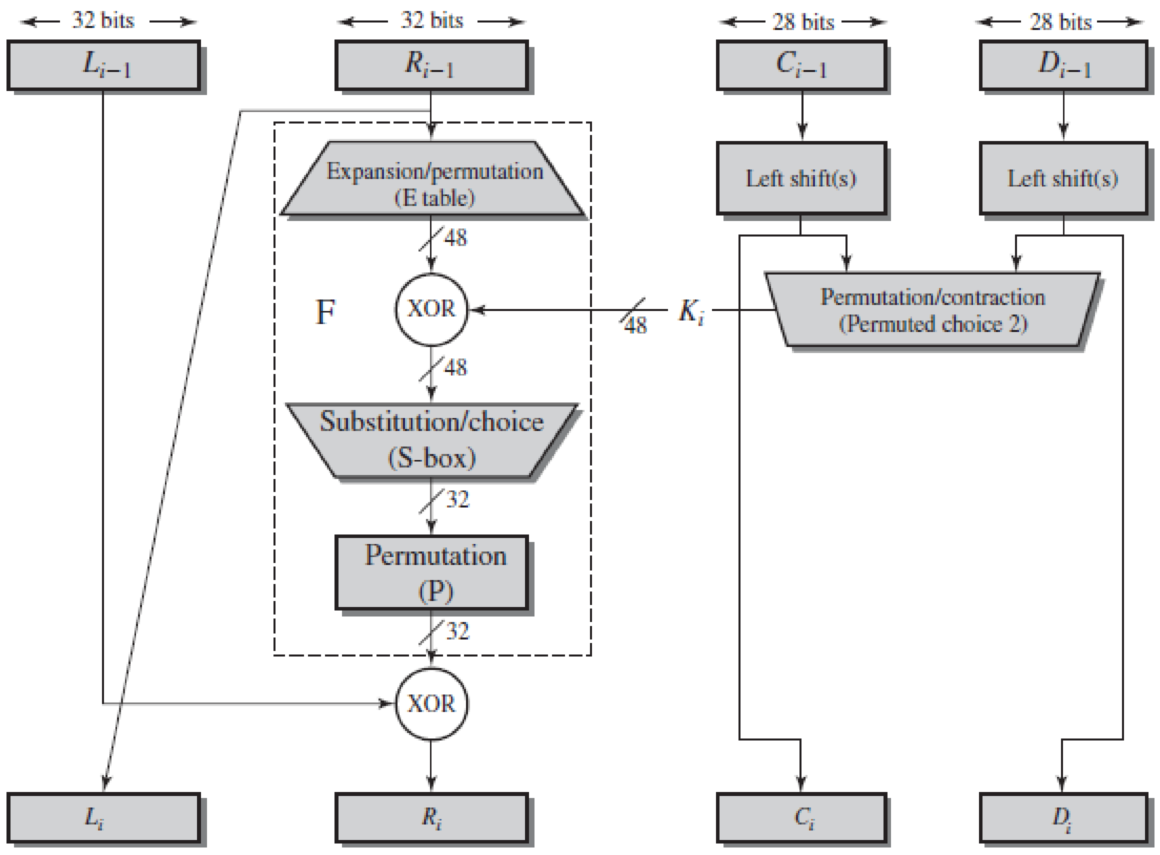


Figure :Single Round of DES Algorithm

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

.

S-Box Design in DES :

The role of the S-boxes in the function F is illustrated in Figure 3.7.The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output

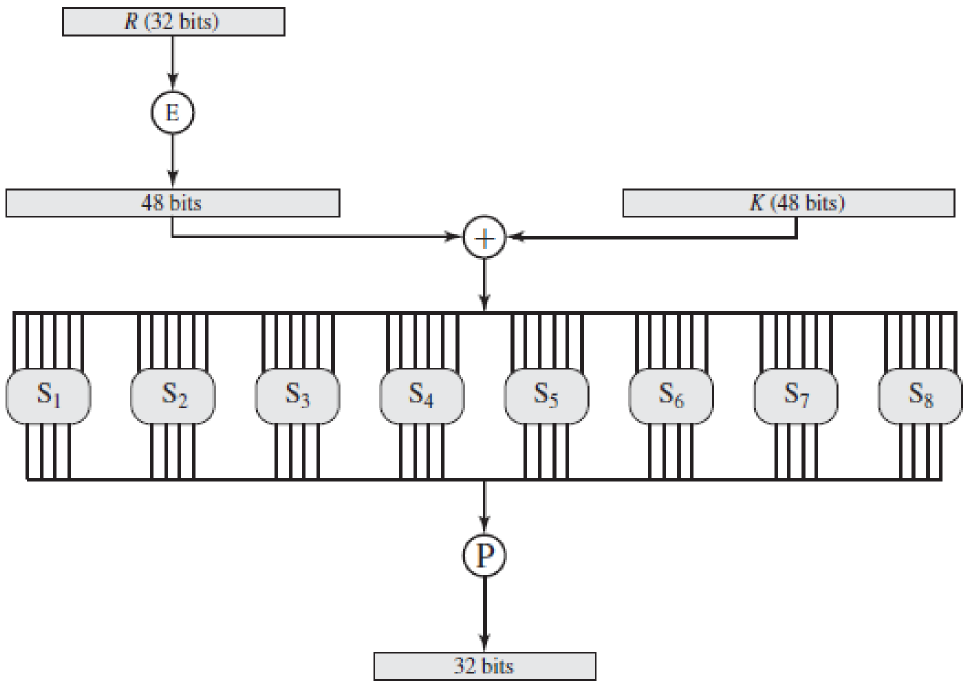


Figure 3.7 Calculation of F(R, K)

Key Generation

- Returning to above figure 3.4, we see that a 64-bit key is used as input to the algorithm.
- The bits of the key are numbered from 1 through 64; every eighth bit is ignored and The key is first subjected to a permutation .
- The resulting 56-bit key is then treated as two 28-bit quantities, labelled C0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift.
- These shifted values serve as input to the next round. They also serve as input to the part labeled Permuted Choice .which produces a 48-bit output that serves as input to the Function $F(R_{i-1}, K_i)$.

<u>Des Decryption:</u>

Whatever process we following in the encryption that process is used for decryption also but the order of key is changed on input message (cipher text).

Reverse order of keys are K_{16}, K_{15} ,……, K_1.

Strengths of DES:

- The DES is a symmetric key block cipher which takes 64bits cipher text and 56 bit key as an input and produce 64 bits cipher text as output.
- The DES function is made up of P & S boxes
- P-boxes transpose bits
- S-boxes Substitution bits to generating the cipher text.

The use of 56bits keys: 56 bit key is used in encryption, there are 2^{56} possible keys,which is approximately $2^{56}=7.2\times10^{16}$ keys, by this a brute force attack on such number of keys is impractical. A machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

The nature of algorithm: Cryptanalyst can perform cryptanalysis by exploiting the characteristic of DES algorithm but no one has succeeded in finding out the weakness. This is possible because, in DES, they using 8-substitution tables or S-boxes in each iteration & one P-box transition for the every individual iteration.

Avalanche Effect:

key desirable property of an encryption algorithm

a small change in either the plain text or the key should produce a significant change in the cipher text(this property is called Avalanche Effect)

DES exhibits strong avalanche Effect

Timing Attacks:

Timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts.

The authors conclude that DES appears to be fairly resistant to a successful timing attack

BLOCK CIPHER DESIGN PRINCIPLES

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed

There are three critical aspects of block cipher design:

- The number of rounds,
- Design of the function F,
- Key scheduling

Number of Rounds

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F.

In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES.

Design of Function F

The heart of a Feistel block cipher is the function F. in DES, this function relies on the use of S-boxes.

Design Criteria For F: *The* function F provides the element of confusion in a Feistel cipher. Thus, it must be difficult to “unscramble” the substitution performed by F. One obvious criterion is that F be nonlinear.

Several other criteria should be considered in designing F. We would like the algorithm to have good avalanche properties. Recall that, in general, this means that a change in one bit of the input should produce a change in many bits of the output.

Key Schedule Algorithm

With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key.

BLOCK CIPHER MODES OF OPERATION

- Mode of operation is a Technique for enhancing the effect of a cryptographic algorithm
- A four modes are intended to cover virtually all possible applications of encryption for which a block cipher could be used

Mode	**Description**	**Typical Application**
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	Secure transmission of single and small 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 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 block oriented Transmission --Authentication
Output Feedback (OFB)	Input is processed 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 block oriented Transmission --Authentication
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

Electronic Codebook (ECB)

- The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 6.3).
- The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext.

We can define ECB mode as follows

ECB	$C_j = E(K, P_j)$	$j = 1, \ldots, N$	$P_j = D(K, C_j)$	$j = 1, \ldots, N$

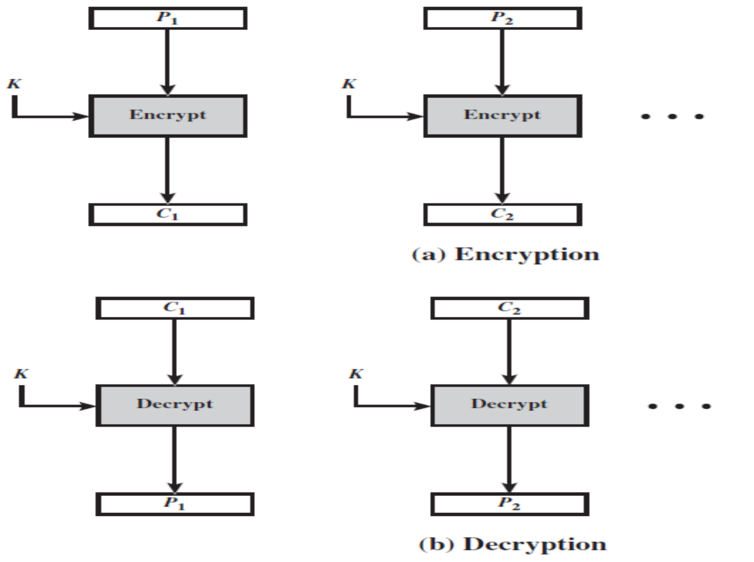


Figure 6.3 Electronic Codebook (ECB) Mode

Advantages:
- The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use.
- The most significant characteristic of ECB is that the same b-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

Disadvantages:
- 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.

Cipher Block Chaining Mode(CBC)

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

A simple way to satisfy this requirement is the cipher block chaining (CBC) mode (Figure 6.4). In this scheme, 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

We can define CBC mode as follows

CBC	$C_1 = \mathrm{E}(K, [P_1 \oplus \mathrm{IV}])$ $C_j = \mathrm{E}(K, [P_j \oplus C_{j-1}])\ \ j = 2, \ldots, N$	$P_1 = \mathrm{D}(K, C_1) \oplus \mathrm{IV}$ $P_j = \mathrm{D}(K, C_j) \oplus C_{j-1}\ \ j = 2, \ldots, N$

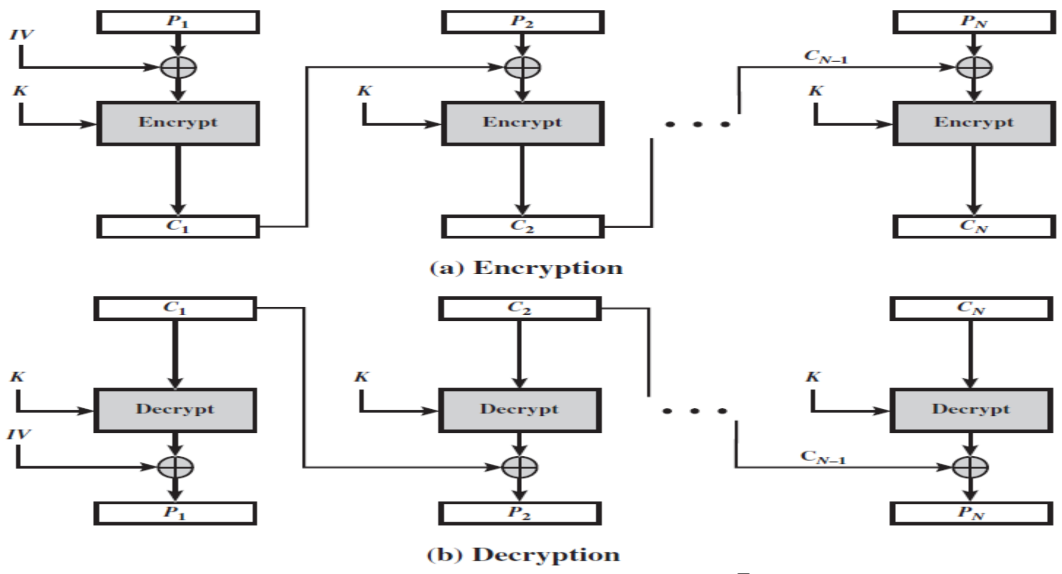


Figure 6.4 Cipher Block Chaining (CBC) Mode

Initilization Vector:

- 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 but be unpredictable by a third party.
- For maximum security, the IV should be protected against unauthorized changes.

<u>Cipher Feedback Mode(CFB)</u>

For AES, DES, or any block cipher, encryption is performed on a block of b bits. In the case of DES,b=64 and in the case of AES,b=128 . However, it is possible to convert a block cipher into a stream cipher, using one of the three modes to be discussed in this and the next two sections: cipher feedback (CFB) mode, output feedback (OFB) mode, and counter (CTR) mode.

- Message is treated as a stream of bits.
- Result is feed back for next stage (hence name)
- CFB-64 is used most often (most efficient)
- Applications: stream data encryption, authentication

The encryption function is

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

Therefore, by rearranging terms:

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

We can define CBC mode as follows

CFB	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $C_j = P_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $P_j = C_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$

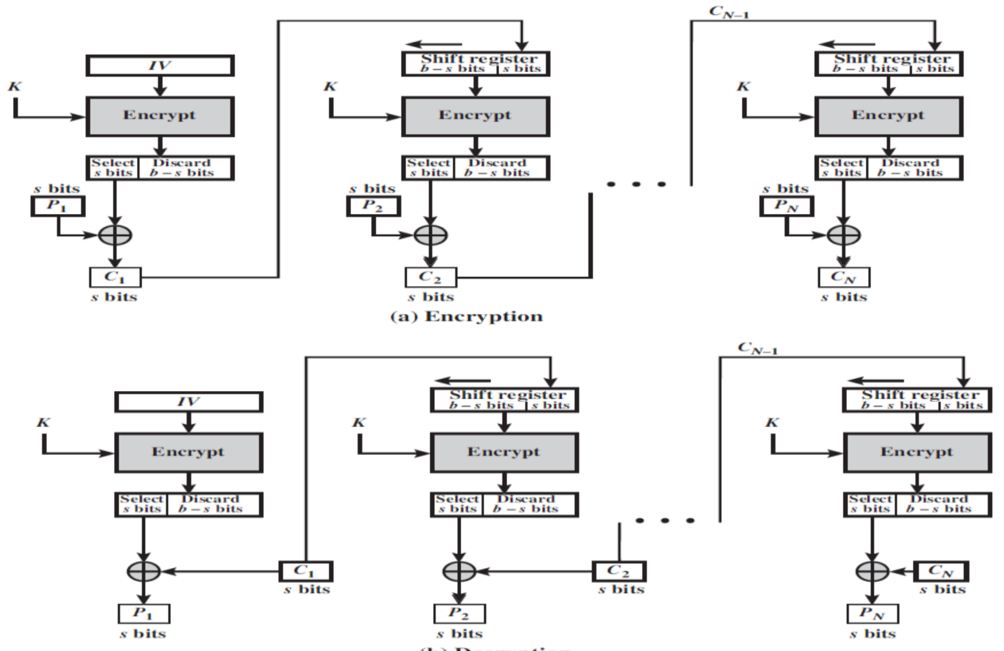


Figure 6.5 s-bit Cipher Feedback (CFB) Mode

Output feedback mode:

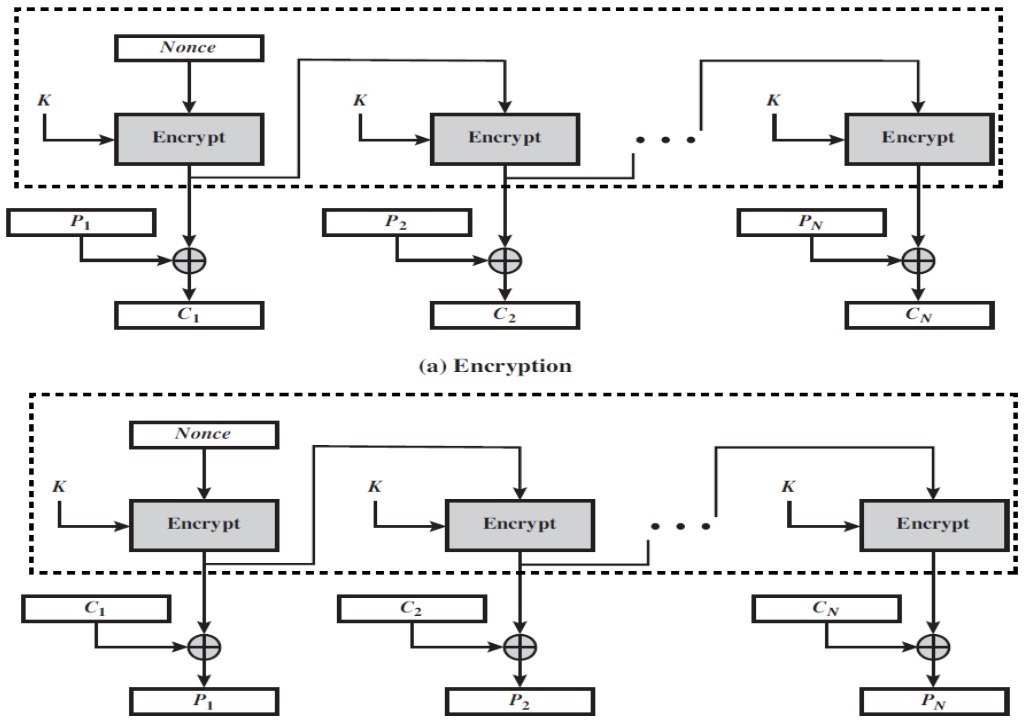
The output feedback (OFB) mode is similar in structure to that of CFB. As can be seen in Figure 6.6, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB, the ciphertext unit is fed back to the shift register. The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, not on an n-bit subset. Encryption can be expressed as

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

By rearranging terms, we can demonstrate that decryption works.

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

.



(a) Encryption

(b) Decryption

Figure 6.6 Output Feedback (OFB) Mode

We can define OFB mode as follows

OFB	$I_1 = Nonce$	$I_1 = Nonce$
	$I_j = O_{j-1} \quad j = 2, \ldots, N$	$I_j = \mathrm{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$
	$O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$	$O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$
	$C_j = P_j \oplus O_j \quad j = 1, \ldots, N - 1$	$P_j = C_j \oplus O_j \quad j = 1, \ldots, N - 1$
	$C_N^* = P_N^* \oplus \mathrm{MSB}_u(O_N)$	$P_N^* = C_N^* \oplus \mathrm{MSB}_u(O_N)$

Advantage of OFB:

- One **advantage** of the OFB method is that bit errors in transmission do not propagate.
- For example, if a bit error occurs in C1 only the recovered value of is P1 affected; subsequent plaintext units are not corrupted.

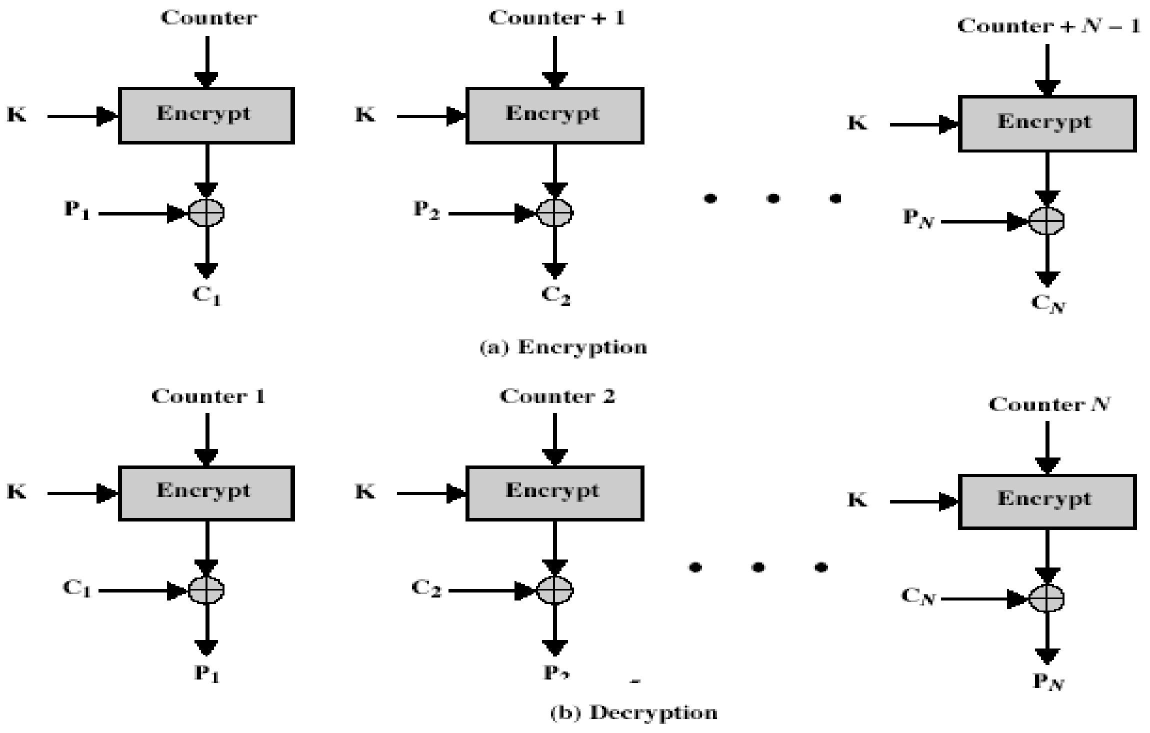
Disadvantage of OFB:

- The **disadvantage** of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

Counter Mode

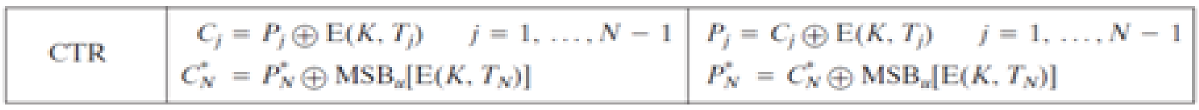
Although interest in the counter (CTR) mode has increased recently with applications to ATM (asynchronous transfer mode) network security and IP sec (IP security)

- A counter, equal to the plaintext block size is used. The only requirement 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



We can define CTR mode as follows

Figure 6.7 Counter (CTR) Mode



CTR	$C_j = P_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $C^*_N = P^*_N \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$	$P_j = C_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $P^*_N = C^*_N \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$

Advantages of CTR mode:

- **Hardware and software efficiency:** Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext.
- **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.
- **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.

- **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.

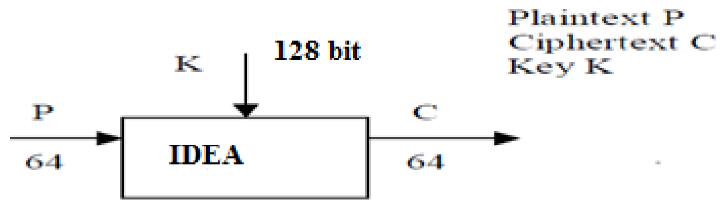
IDEA (International Data Encryption Algorithm):

IDEA originally called “IPES” (Improved proposed Encryption Standard).

IDEA is one of a number of conventional encryption algorithms that have been proposed in recent years to replace DES

IDEA is one of the most successful of these proposals. For example, IDEA is included in PGP.

Details of IDEA algorithm:



IDEA operates with 64 bit plain text and cipher text blocks and is controlled b a 128 bit key.

It avoids substitution boxes & lookup tables used in the block cipher.

The algorithm structure has been chosen such that different key sub-blocks are used; the encryption process is identical to the decryption process.

Encryption process in IDEA:

- The design principle behind IDEA is mixing of arithmetical operations form different algebraic groups.
- The underling operations are

  1. Exclusive-OR.
  2. Addition of integers modulo 2^{16}
  3. Multiplication modulo $2^{10}+1$

- The algorithm structure has been chosen such that when different key sub-blocks are used, the encryption process is identical to the decryption process
- The IDEA algorithm consists of eight rounds followed by a final transformation function. The algorithm divides the input into four 16-bit subblocks. Each of the rounds takes four

16-bit subblocks as input and produces four 16-bit output blocks. The final transformation also produces four %-bit blocks, which are concatenated to form the 64-bit ciphertext.

- Each of the rounds also makes use of six 16-bit subkeys, whereas the final transformation uses four subkeys, for a total of 52 subkeys

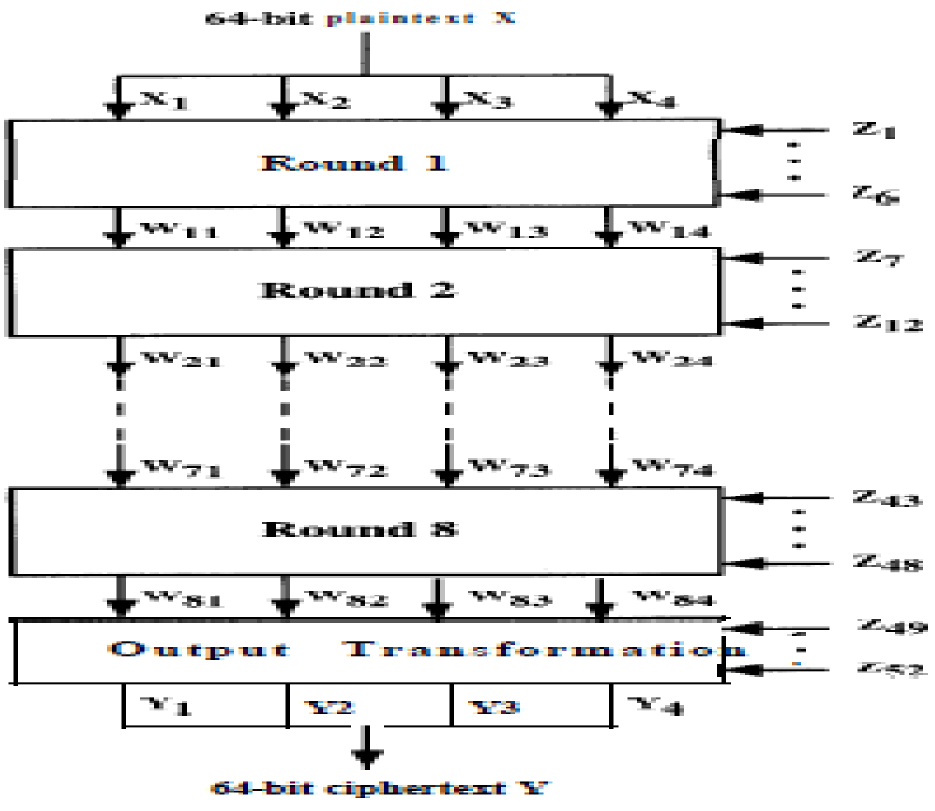


Figure Overall IDEA Structure.

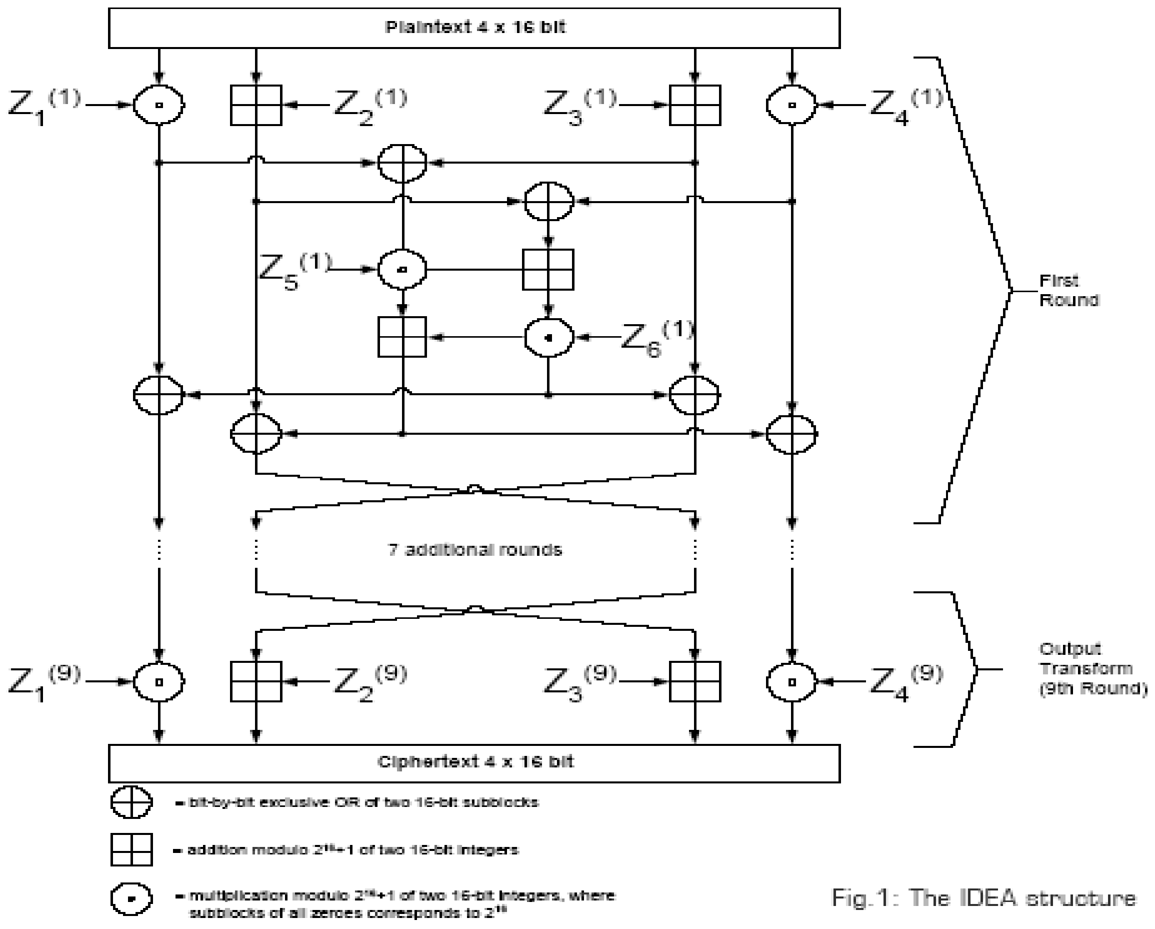
Key Expansion (Encryption):

- The 128-bit key is expanded into 52 16-bit keys: K1, K2, ...K52. (in diagram we represented these keys with Z_1 to z_{52})
  Step 1: Keys K1….K8 are generated by taking 8 chunks of 16-bits each
- Step 2: Keys K9…K16 are generated by starting from the 25th bit, wrapping around the first 25 bits at the end, and taking 16-bit chunks.
- Step 3: Wrap around 25 more bits to the end, and generate keys K17…K24. This process is repeated until all keys K1…K52 are generated

<u>Details of a Single Round:</u>

64 bit data is divided into 4 16bit data blocks. These 4 blocks are processed through 8 rounds and transformed by the above arithmetical operations among each other and with 6 16 bit subkeys.

1. Multiply X_1 and the first sub key Z_1.
2. Add X_2 and the second sub key Z_2.
3. Add X_3 and the third sub key Z_3.
4. Multiply X_4 and the fourth sub key Z_4.
5. Bitwise XOR the results of steps 1 and 3.
6. Bitwise XOR the results of steps 2 and 4.
7. Multiply the result of step 5 and the fifth sub key Z_5.
8. Add the results of steps 6 and 7.
9. Multiply the result of step 8 and the sixth sub key Z_6.
10. Add the results of steps 7 and 9.
11. Bitwise XOR the results of steps 1 and 9.
12. Bitwise XOR the results of steps 3 and 9.
13. Bitwise XOR the results of steps 2 and 10.
14. Bitwise XOR the results of steps 4 and 10.



Plaintext 4 x 16 bit
$Z_1^{(1)}$
$Z_2^{(1)}$
$Z_3^{(1)}$
$Z_4^{(1)}$
$Z_5^{(1)}$
$Z_6^{(1)}$
First Round
7 additional rounds
$Z_1^{(9)}$
$Z_2^{(9)}$
$Z_3^{(9)}$
$Z_4^{(9)}$
Output Transform (9th Round)
Ciphertext 4 x 16 bit
= bit-by-bit exclusive OR of two 16-bit subblocks
= addition modulo $2^{16}+1$ of two 16-bit integers
= multiplication modulo $2^{16}+1$ of two 16-bit integers, where subblocks of all zeroes corresponds to 2^{16}

Fig.1: The IDEA structure

BLOW FISH ALGORITHM:

Blow fish is a symmetric block cipher developed by bruce schner in year 1993.

Blow fish is designed to have following characteristics

Speed: Blowfish encrypts data on 32 bit microprocessor at a rate of 18 clock cycles per byte.

Compact: it can run in less than 5k memory.

Simple: very easy to implements.

Variably secure: the key length is variable and can be as long as 448 bits. This allows a trade off between higher speed and higher security.

Blowfish is a feistal type model.

ALGORITHM:

- Blowfish is feistel type model, iterating a simple encryption function 16 times.
- Blowfish block size is 64 & key can be upto 448 bits.
- Blow fish encryption 64bits blocks of plaintext into 64 bit block of cipher.
- Blow fish make use of a key that ranges from 32bits to 448 bits (one to fourteen 32 bit keys).
- The keys are stored in a k-array (one to 14 32 bits)

  K_1,K_2----K_j where $1\leq j \leq 14$.

- That key is used to generate 18 "32 bit" subkeys & four "8*32"bits S-boxes.
- The subkeys are stored in the p-array

  P_1,P_2,-------P_{18}

- There are four s-boxes(each s-box size is 8*32 bits) each with 256 32bit entries.

$S_{1,0}$, $S_{1,1}$,-------------------$S_{1,255}$

$S_{2,0}$, $S_{2,1}$,-------------------$S_{2,255}$

$S_{3,0}$, $S_{3,1}$,-------------------$S_{3,255}$

$S_{4,0}$, $S_{4,1}$,-------------------$S_{4,255}$

Encryption and Decryption

Blowfish uses two primitive operations:

Addition: Addition of words, denoted by +, is performed modulo 2^{32}.

Bitwise exclusive-OR: This operation is denoted by $\oplus$

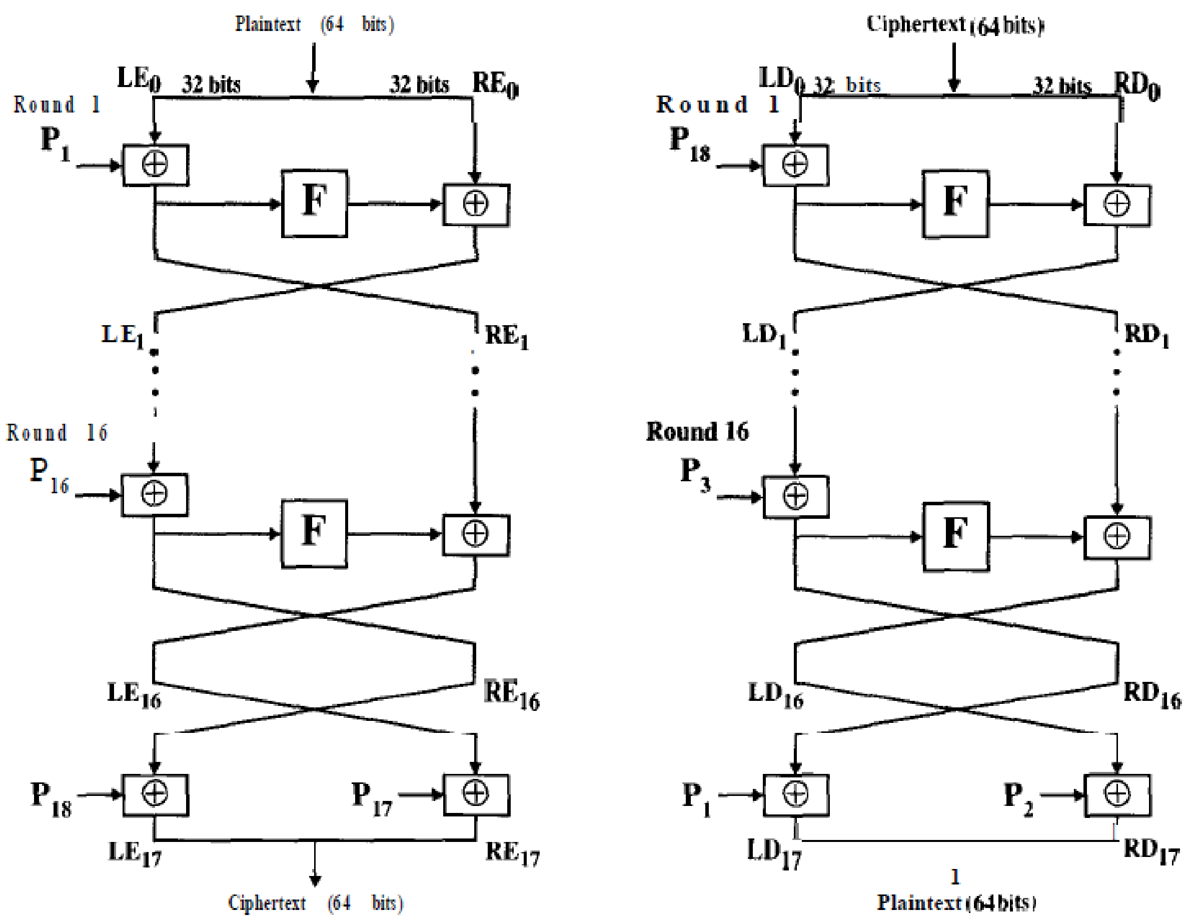


Figure Blowfish Encryption and Decryption.

In the above figure

- The plaintext is divided into two 32-bit halves LE, and RE,. We use the variables LE, and RE, to refer to the left and half of the data after round i has completed. The algorithm can be defined by the following pseudocode:

```
for $i$ = 1 to 16 do
        $RE_i = LE_{i\ 1} \oplus P_i;$
        $LE_, = F[RE_i] \oplus RE_{i-1};$
$LE_{,,} = RE_{,,} \oplus P_{18};$
$RE_{,,} = LE_{,,} \oplus P_{17};$
```

Single round of Blowfish

The function F is shown in below Figure. The 32-bit input to F is divided into 4 bytes. If we label those bytes a, b, c, and d, then the function can be defined as follows:

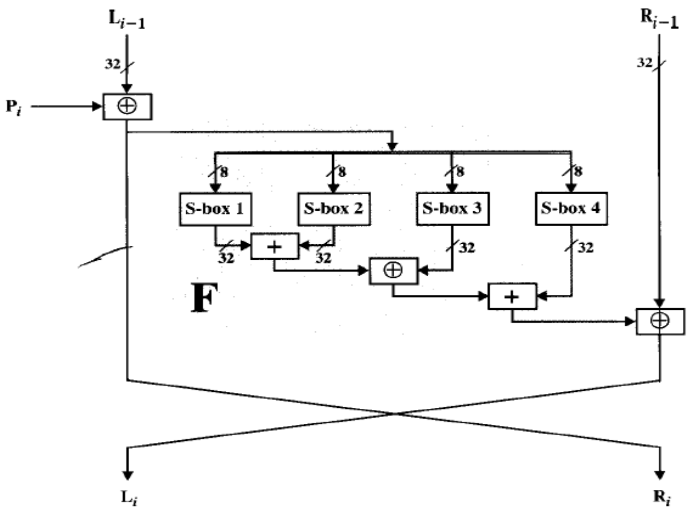


Figure Detail of Single Blowfish Round.

$$F[a, b, c, d,] = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$$

<u>CAST-128</u>

- In cryptography, CAST-128 (alternatively CAST5) is a symmetric-key block cipher.
- CAST-128, also known as CAST5
- CAST-128 algorithm was created in 1996 by Carlisle Adams and Stafford Tavares. The CAST name is based on the initials of its inventors
- CAST-128 is a 12- or 16-round Feistel network with a 64-bit block size and a key size of between 40 to 128 bits (but only in 8-bit increments). The full 16 rounds are used when the key size is longer than 80 bits.
- CAST-128 uses four primitive operations:

  1. Addition and subtraction: Addition of words, denoted by +, is performed modulo 2^{32}. The inverse operation, denoted by -, is subtraction modulo 2^{32}.
  2. Bitwise exclusive-OR: This operation is denoted by $\oplus$.
  3. Left circular rotation: The cyclic rotation of word x left by y bits is denoted by x <<< y.

The CAST-128 encryption algorithm can be defined by the following pseudocode. The plaintext is divided into two 32-bit halves L_0, and R_0. We use the variables L_i and R_i, to refer to the left and right half of the data after round *i* has completed. The ciphertext is formed by swapping the output of the sixteenth round; that is, the ciphertext is the concatenation of R16 and L16.

```
$L_0 \| R_0$ = Plaintext
for $i$ = 1 to 16 do
        $L_i = R_{i-1}$;
        $R_i = L_{i-1} \oplus F_i[R_{i-1}, Km_i, Kr_i]$
Ciphertext = $R_{16} \| L_{16}$
```

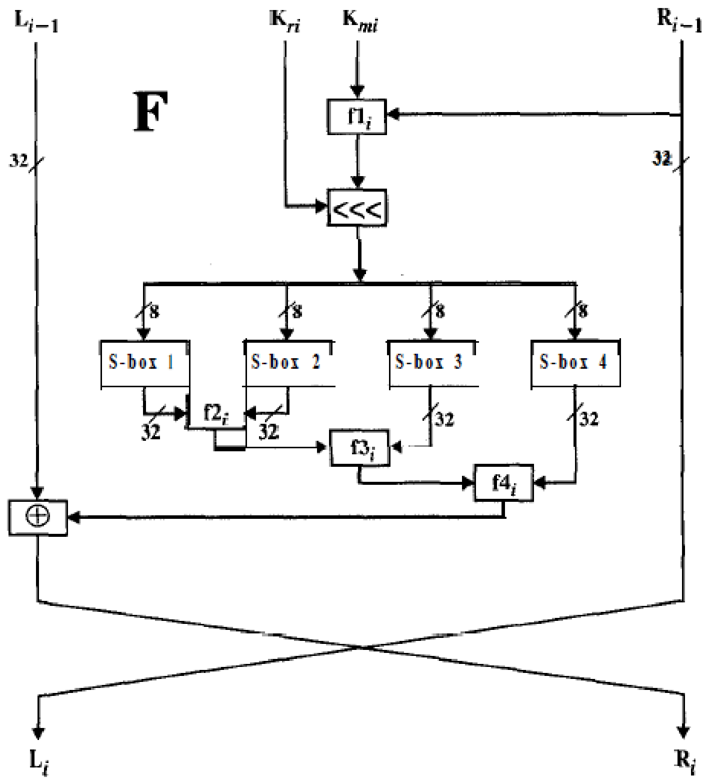


Figure 4.14 Detail of Single CAST-128 Round.

Features of CAST-128:

There are several notable features of CAST worthy of comment.,

CAST makes use of fixed S-boxes. The designers felt that fixed S-boxes with good nonlinearity characteristics are preferable to random S-boxes as might be obtained if the S-boxes were key dependent. The subkey-generation process used in CAST-128 is different from that employed in other symmetric encryption algorithms described in the literature.

The CAST designers were concerned to make subkeys as resistant to known cryptanalytic attacks as possible and felt that the use of highly nonlinear S-boxes provided this strength. We have seen other approaches with the same goal.

For example. Blowfish uses the encryption algorithm itself to generate the subkeys.

The function F is designed to have good confusion, diffusion. and avalanche properties. It uses S-box substitutions, mod 2 addition and subtraction, exclusive- OR operations, and key-dependent rotation.

The strength of the F function is based primarily on the strength of the S-boxes, but the further use of these arithmetic. Boolean, and rotate operators adds to its strength. Finally, F is not uniform from round to round, as was described. This dependence of F on round number may provide.

ADVANCED ENCRYPTION STANDARD

- The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001.
- AES is a block cipher intended to replace DES for commercial applications.
- It uses a 128-bit block size and a key size of 128, 192, or 256 bits.the algorithm is referred as AES-128,AES-192 OR AES-256
- AES does not use a Feistel structure. Instead, each full round consists of four separate functions: byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key.
- AES parameters:

Key size(words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round Key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

AES STRUCTURE

General structure

- The input to the encryption and decryption algorithms is a single 128-bit block. , this block is depicted as a 4 * 4 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. These operations are depicted in Figure 5.2a. Similarly, the key is depicted as a square matrix of bytes.
- This key is then expanded into an array of key schedule words. Figure 5.2b shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key

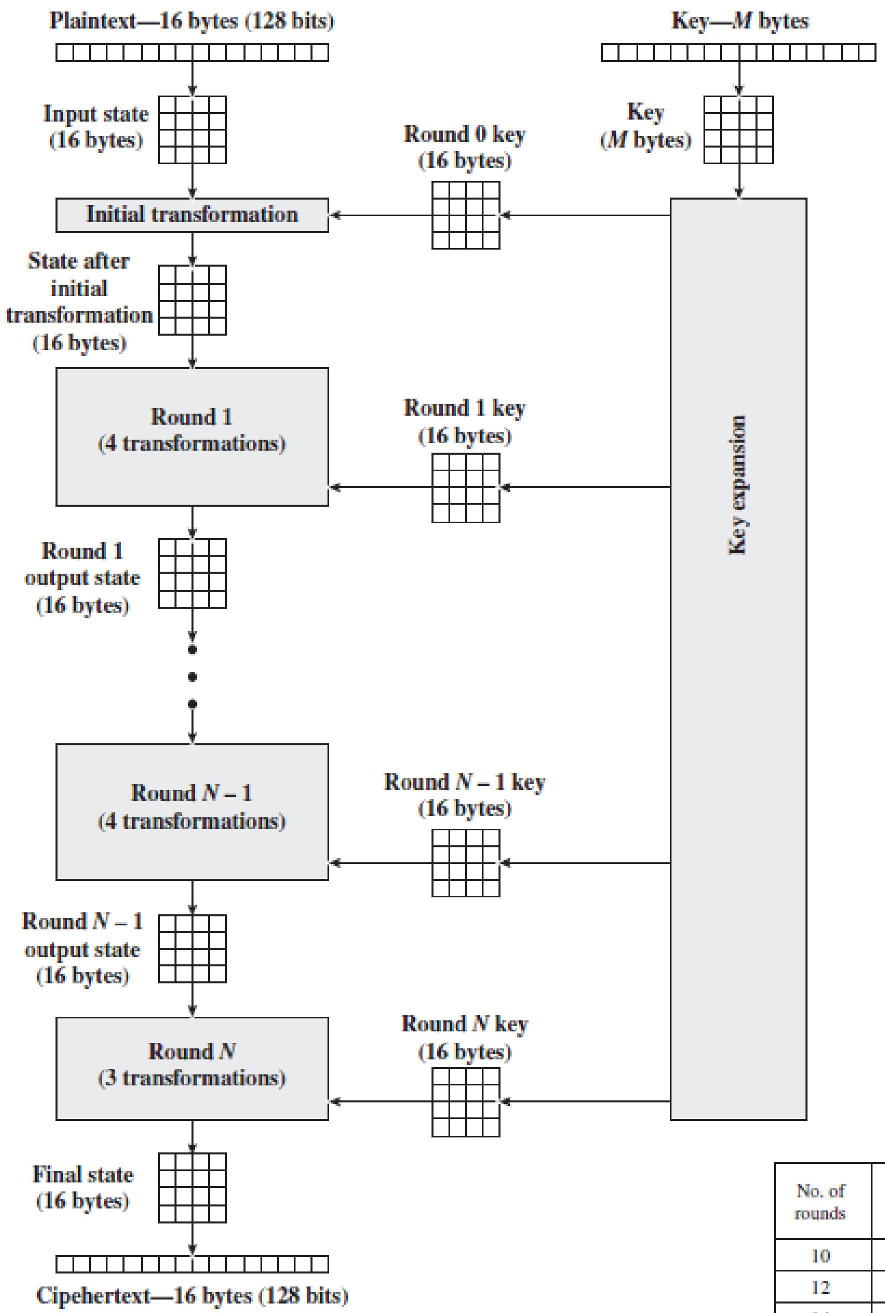

Plaintext—16 bytes (128 bits)
Key—M bytes
Input state (16 bytes)
Key (M bytes)
Round 0 key (16 bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext—16 bytes (128 bits)

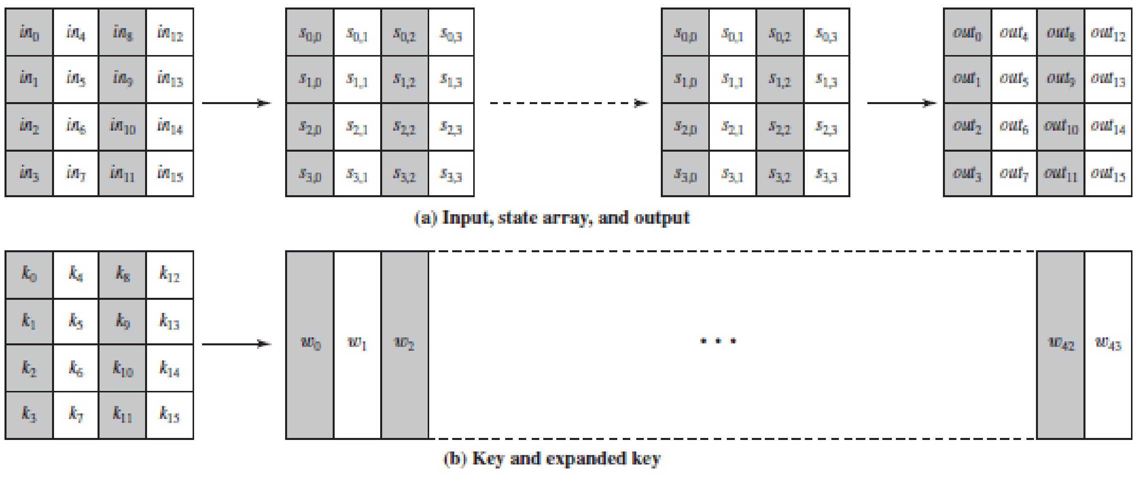
Figure 5.1 AES Encryption Process

No. of rounds	Key Length (bytes)
10	16
12	24
14	32

The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key (Table 5.1).

The first N-1 rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only Three Transformations, and there is a initial single transformation (AddRoundKey) before the first round,

which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the ciphertext. Also, the key expansion function generates $N + 1$ round keys, each of which is a distinct 4×4 matrix. Each round key serve as one of the inputs to the AddRoundKey transformation in each round.



in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}
(a) Input, state array, and output
k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 . . . w_{42} w_{43}
(b) Key and expanded key

Figure 5.2 AES Data Structures

Detailed Structure

- Figure 5.3 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function

Four different stages are used, one of permutation and three of substitution:

• Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block

• ShiftRows: A simple permutation

• MixColumns: A substitution that makes use of arithmetic

• AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

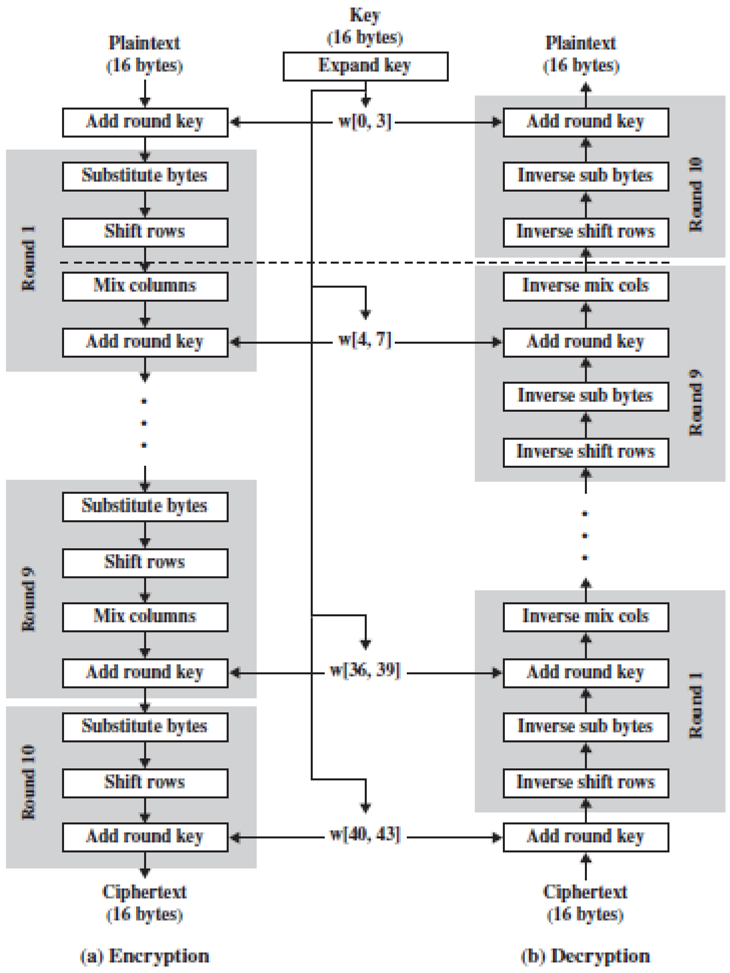


Figure 5.3 AES Encryption and Decryption

<u>AES TRANSORMATION FUNCTIONS</u>

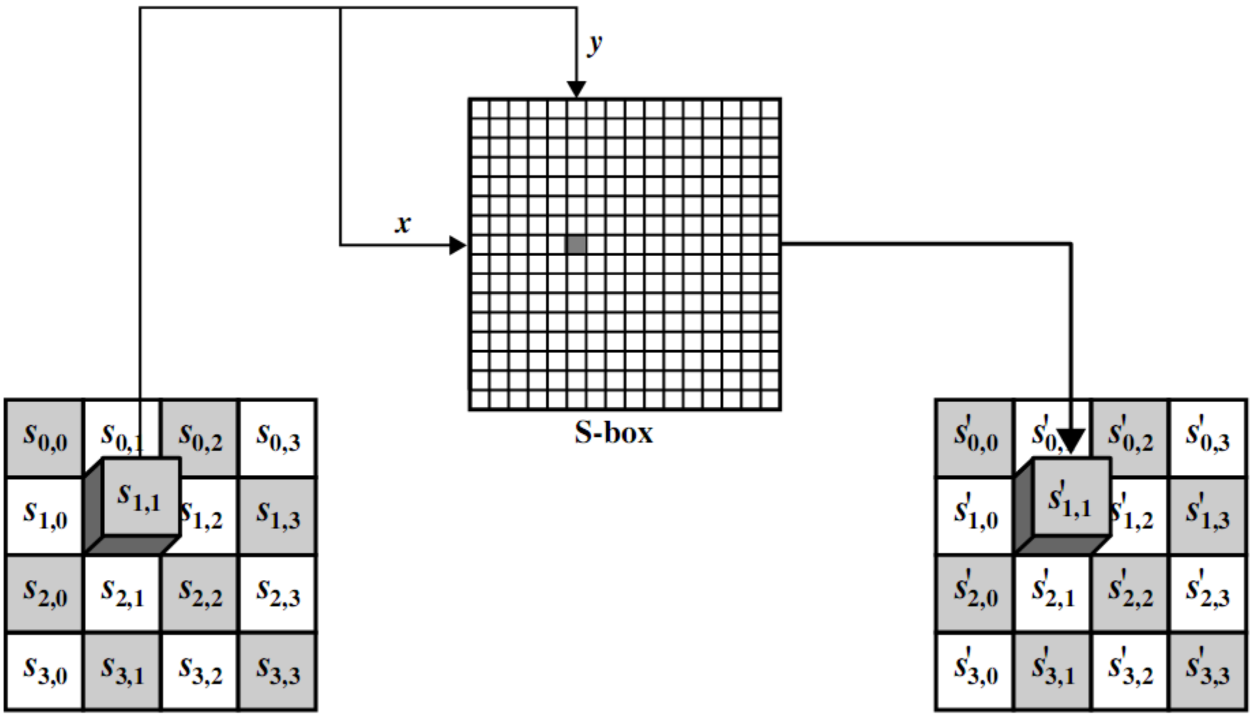
The four transformation functions are

- Substitute bytes
- ShiftRows
- MixColumns
- AddRoundKey

Substitute Bytes Transformation

- The forward substitute byte transformation, called SubBytes, is a simple table lookup (Figure 5.5a).
- AES defines a16 *16 matrix of byte values, called an S-box (Table 5.2a), that contains a permutation of all possible 256 8-bit values.
- Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a

column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value3 {95} references row 9, column 5 of the S-box,which contain the value {2A}


y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

(a) Substitute bvte transformation

Table 5.2 AES S-Boxes

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Shift Rows Transformation

- The first row of State is not altered.
- For the second row, a 1-byte circular left shift is performed.
- For the third row, a 2-byte circular left shift is performed.
- For the fourth row, a 3-byte circular left shift is performed

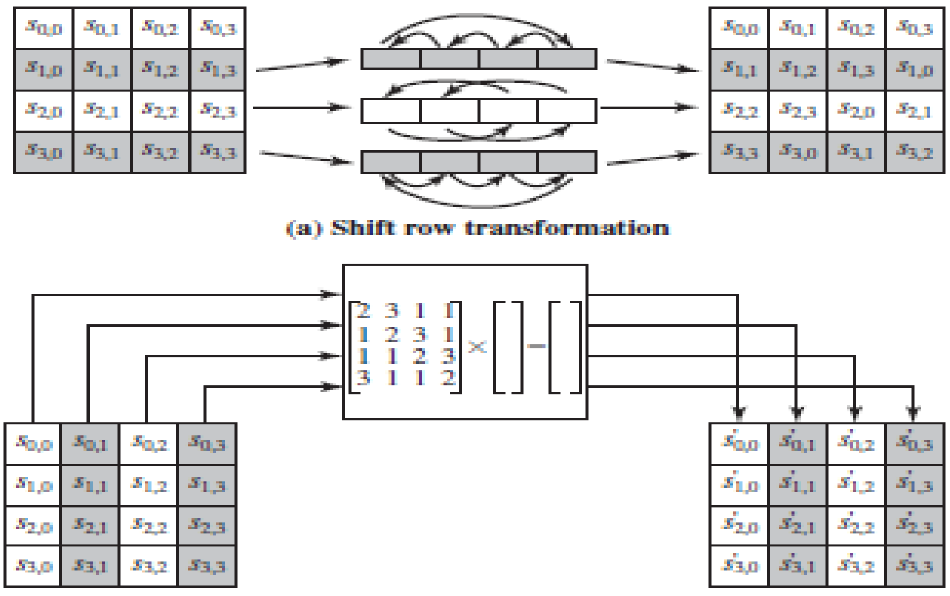
The following is an example of ShiftRows:

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

The **inverse shift row transformation**, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$
$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times [\ ] = [\ ]$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

(a) Shift row transformation

(b) Mix column transformation

Figure 5.7 AES Row and Column Operations

Mix columns Transformation

- The forward mix column transformation, called MixColumns, operates on each column individually.
- Each byte of a column is mapped into a new value that is a function of all four bytes in that column.

transformation can be defined by the following matrix multiplication on **State** (Figure 5.7b):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(5.3)}$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$$\begin{aligned}
&(\{02\}\cdot\{87\}) \oplus (\{03\}\cdot\{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\} \\
&\{87\} \oplus (\{02\}\cdot\{6E\}) \oplus (\{03\}\cdot\{46\}) \oplus \{A6\} = \{37\} \\
&\{87\} \oplus \{6E\} \oplus (\{02\}\cdot\{46\}) \oplus (\{03\}\cdot\{A6\}) = \{94\} \\
&(\{03\}\cdot\{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\}\cdot\{A6\}) = \{ED\}
\end{aligned}$$

For the first equation, we have $\{02\}\cdot\{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$ and $\{03\}\cdot\{6E\} = \{6E\} \oplus (\{02\}\cdot\{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010)$. Then,

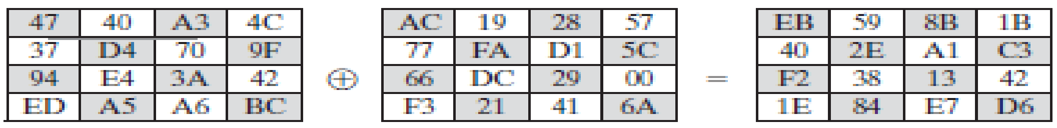
$$\begin{aligned}
\{02\}\cdot\{87\} &= 0001\ 0101 \\
\{03\}\cdot\{6E\} &= 1011\ 0010 \\
\{46\} &= 0100\ 0110 \\
\{A6\} &= \underline{1010\ 0110} \\
&\ 0100\ 0111 = \{47\}
\end{aligned}$$

The other equations can be similarly verified.

AddRoundKey Transformation

- In the AddRoundKey transformation, the 128 bits of State are bitwise XORed with the 128 bits of the round key.
- The operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation.

The following is an example of AddRoundKey:



$$\begin{bmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{bmatrix} \oplus \begin{bmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ F3 & 21 & 41 & 6A \end{bmatrix} = \begin{bmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D6 \end{bmatrix}$$

The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

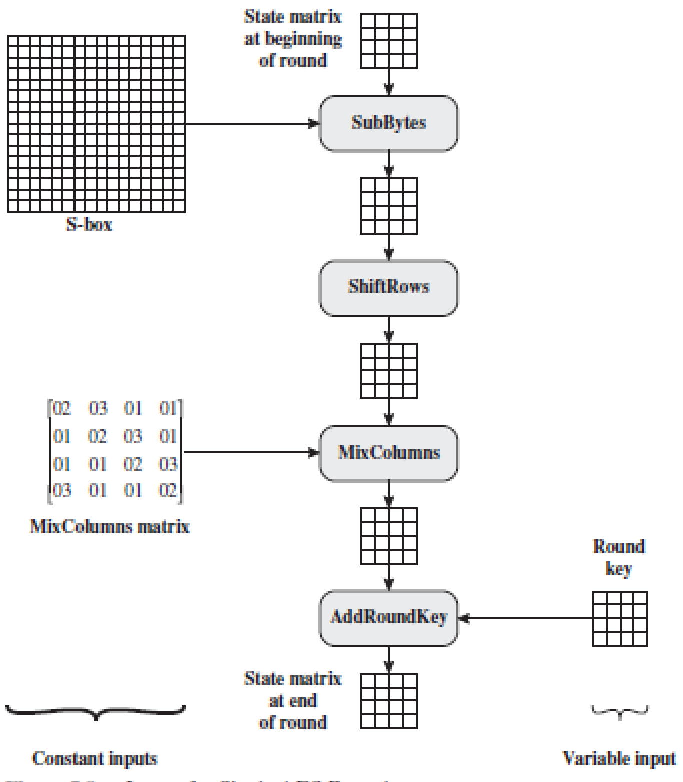
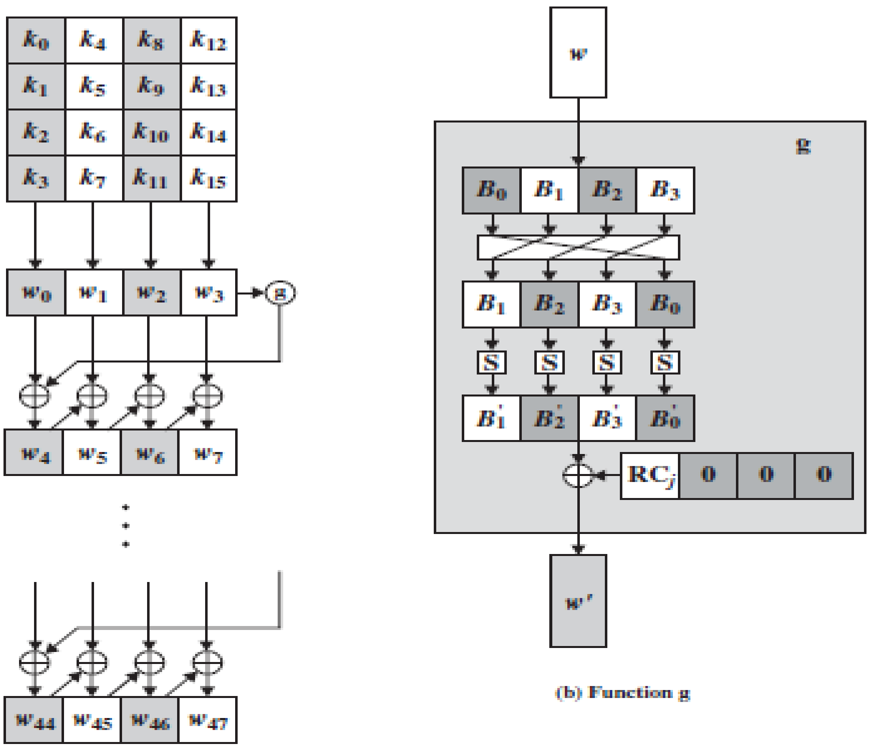


Figure 5.8 Inputs for Single AES Round

AES KEY EXPANSION

- The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes).
- This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.
- The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time.
- Each added word w[i] depends on the immediately preceding word, w[i 1], and the word four positions back,w[i 4].


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w_{44} w_{45} w_{46} w_{47}
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B_1' B_2' B_3' B_0'
RC_j 0 0 0
w'

(a) Overall algorithm

(b) Function g

Figure 5.9 AES Key Expansion

In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used. The function ‘g’ consists of the following subfunctions:

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].