UNIT II SYMMETRIC KEY CRYPTOGRAPHY

MATHEMATICS OF SYMMETRIC KEY CRYPTOGRAPHY: Algebraic Structures - Modular arithmetic - Euclid's Algorithm - Congruence and Matrices - Groups, Rings, Fields - Finite fields - SYMMETRIC KEY CIPHERS: SDES - Block cipher principles of DES - Strength of DES - Differential and Linear Cryptanalysis - Block cipher design principles - Block cipher mode of operation - Evaluation criteria of AES - Advanced Encryption Standard - RC4 - Key Distribution

2.1 ALGEBRAIC STRUCTURES

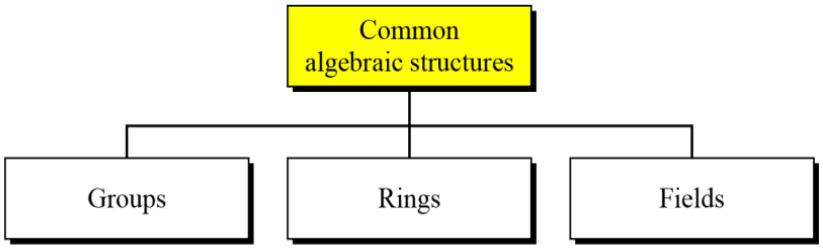


Figure 2.1 Common Algebraic Structures

2.1.1 Groups, Rings, Fields

Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra.

Groups

A group G , sometimes denoted by {G,*} ,is a set of elements with a binary operation denoted by * that associates to each ordered pair (a,b) of elements G in an element(a*b) in , such that the following axioms are obeyed:

(A1) Closure: If a and b belong to G, then a*b is also in G. (A2) Associative: a*(b*c)=(a*b)*c for all a, b, , in G .
(A3) Identity element: There is an element e in G such that a*e=e*a=a for all in G .
(A4) Inverse element: For each a in G, there is an element a' in G such that a*a'=a'*a=e .

If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**.

A group is said to be **abelian** if it satisfies the following additional condition:
(A5) Commutative: a*b = b*a for all a b, in G.

CYCLIC GROUP: A group is cyclic if every element of G is a power a^k (k is an integer) of a fixed element a£ G .The element is a said to generate the group G or to be a generator of
G.A cyclic group is always abelian and may be finite or infinite**.**

Rings

A ring R, sometimes denoted by {R, +, X}, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c ,in R the following axioms are obeyed

(A1–A5)	R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as $-a$.
(M1) Closure under multiplication:	If a and b belong to R, then ab is also in R.
(M2) Associativity of multiplication:	$a(bc) = (ab)c$ for all a, b, c in R.
(M3) Distributive laws:	$a(b + c) = ab + ac$ for all a, b, c in R. $(a + b)c = ac + bc$ for all a, b, c in R.

A ring is said to be **commutative** if it satisfies the following additional condition:

(M4) Commutativity of multiplication: $ab = ba$ for all a, b in R.

Next, we define an integral domain, which is a commutative ring that obeys the following axioms

(M5) Multiplicative identity:	There is an element 1 in R such that $a1 = 1a = a$ for all a in R.
(M6) No zero divisors:	If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Fields

A field F, sometimes denoted by {F, +, X}, is a set of elements with two binary operations, called addition and subtraction, such that for all a, b, c , in F the following axioms are obeyed

(A1–M6)	F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.
(M7) Multiplicative inverse:	For each a in F, except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.

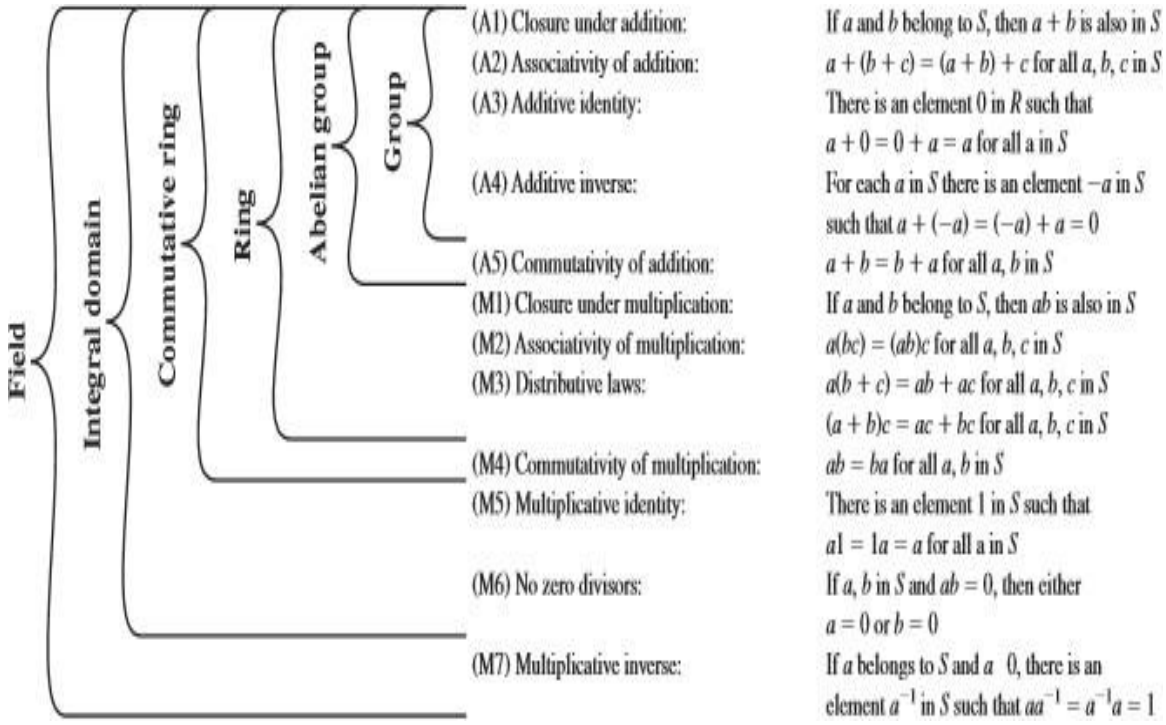


Figure 2.2 Groups, Ring and Field

2.2 MODULAR ARITHMETIC

If is an integer and n is a positive integer, we define a mod n to be the remainder when a is divided by n. The integer n is called the modulus. Thus, for any integer a, we can rewrite Equation as follows

$$a = qn + r \qquad 0 \le r < n;\ q = \lfloor a/n \rfloor$$

$$a = \lfloor a/n \rfloor \times n + (a \bmod n)$$

11 mod 7 = 4; −11 mod 7 = 3

Two integers a and b are said to be **congruent modulo n**, if $(a \bmod n) = (b \bmod n)$. This is written as $a \equiv b \pmod n$.[2]

$73 \equiv 4 \pmod{23}$; $21 \equiv -9 \pmod{10}$

Note that if $a \equiv 0 \pmod n$, then $n|a$.

Modular Arithmetic Operations

A kind of integer arithmetic that reduces all numbers to one of a fixed set [0,….,n-1] for some number n. Any integer outside this range is reduced to one in this range by taking the remainder after division by n.

Modular arithmetic exhibits the following properties

1. $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
2. $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
3. $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

We demonstrate the first property. Define $(a \bmod n) = r_a$ and $(b \bmod n) = r_b$. Then we can write $a = r_a + jn$ for some integer j and $b = r_b + kn$ for some integer k. Then

$$\begin{aligned}(a + b) \bmod n &= (r_a + jn + r_b + kn) \bmod n \\ &= (r_a + r_b + (k + j)n) \bmod n \\ &= (r_a + r_b) \bmod n \\ &= [(a \bmod n) + (b \bmod n)] \bmod n\end{aligned}$$

The remaining properties are proven as easily. Here are examples of the three properties:

Table 2.1 Arithmetic Modulo 8

$11 \bmod 8 = 3; 15 \bmod 8 = 7$
$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$
$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$
$[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = -4 \bmod 8 = 4$
$(11 - 15) \bmod 8 = -4 \bmod 8 = 4$
$[(11 \bmod 8) \times (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$
$(11 \times 15) \bmod 8 = 165 \bmod 8 = 5$

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Additive and multiplicative inverses modulo 8

2.3 EUCLID' S ALGORITHM

One of the basic techniques of number theory is the Euclidean algorithm, which is a simple procedure for determining the greatest common divisor of two positive integers. First, we need a simple definition: Two integers are relatively prime if their only common positive integer factor is 1.

Greatest Common Divisor

Recall that nonzero b is defined to be a divisor ofa if a =mb for some m, where a,b, and m are integers. We will use the notation gcd(a , b) to mean the greatest common divisor of a and b .The greatest common divisor of a and b is the largest integer that divides both a and b

.We also define gcd(0,0) = 0.

Algorithm

The Euclid's algorithm (or Euclidean Algorithm) is a method for efficiently finding the greatest common divisor (GCD) of two numbers. The GCD of two integers X and Y is the largest number that divides both of X and Y (without leaving a remainder).

For every non-negative integer, a and any positive integer b

gcd (a, b) = gcd (b, a mod b)

```
Algorithm Euclids (a, b)
        α = a
        β = b
        while (β > 0)
                Rem = α mod β
                α = β
                β = Rem
        returnα
```

Steps for Another Method

a = q1b + r1; 0 < r1 < b

b = q2r1 + r2; 0 < r2 < r1

r1 = q3r2 + r3; 0 < r3 < r2

rn-2 = qnrn-1 + rn; 0 < rn < rn-1

rn-1 = q1rn + 0

d = gcd (a, b) = rn

Example 1:

gcd (55, 22) = gcd (22, 55 mod 22)

= gcd (22, 11)

= gcd (11, 22 mod 11)

= gcd (11, 0)

gcd (55, 22) is 11

Example 2:

gcd (30, 50) = gcd (50, 30 mod 50)
= gcd (50, 30)
= gcd (30, 50 mod 30)
= gcd (30, 20)
= gcd (20, 30 mod 20)
= gcd (20, 10)
= gcd (10, 20 mod 10)
= gcd (10, 0)

gcd (30, 50) is 10

Another Method

To find gcd (30,50)

50	= 1 x 30 + 20	gcd (30, 20)
30	= 1 x 20 + 10	gcd (20,10)
20	= 1 x 10 + 10	gcd (10,10)
10	= 1 x 10 + 0	gcd (10,0)

Therefore, gcd (30,50) = 10

Example 3:

gcd (1970, 1066) = gcd (1066, 1970 mod 1066)
=gcd (1066, 904)
=gcd (904, 1066 mod 904)
=gcd (904, 162)
=gcd (162, 904 mod 162)
=gcd (162, 94)
=gcd (94, 162 mod 94)
=gcd (94, 68)
=gcd (68, 94 mod 68)
=gcd (68, 26)
=gcd (26, 68 mod 26)
=gcd (26, 16)
=gcd (16, 26 mod 16)
=gcd (16, 10)
=gcd (10, 16 mod 10)
=gcd (10, 6)
=gcd (6, 10 mod 6)
=gcd (6, 4)

=gcd (4, 6 mod 4)

=gcd (4, 2)

=gcd (2, 4 mod 2)

=gcd (2, 0)

gcd (1970, 1066) is 2

Another Method

To find gcd (1970, 1066)

1970	= 1 x 1066 + 904	gcd (1066, 904)
1066	= 1 x 904 + 162	gcd (904,162)
904	= 5 x 162 + 94	gcd (162, 94)
162	= 1 x 94 + 68	gcd (94, 68)
94	= 1 x 68 + 26	gcd (68, 26)
68	= 2 x 26 + 16	gcd (26, 16)
26	= 1 x 16 + 10	gcd (16, 10)
16	= 1 x 10 + 6	gcd (10, 6)
10	= 1 x 6 + 4	gcd (6, 4)
6	= 1 x 4 + 2	gcd (4, 2)
4	= 2 x 2 + 0	gcd (2, 0)

Therefore, gcd (1970, 1066) = 2

Extended Euclidean Algorithm

Extended Euclidean Algorithm is an efficient method of finding modular inverse of an integer.

Euclid's algorithm can be improved to give not just gcd (a, b), but also used to find the multiplicative inverse of a number with the modular value.

Example 1

Find the Multiplicative inverse of 17 mod 43

17-1 mod 43

17* X = mod 43

X= 17-1 mod 43

43 = 17 * 2 + 9

17 = 9 * 1 + 8

9 = 8* 1+ 1

Rewrite the above equation

9+8(-1) = 1 → (1)

17+9(-1) = 8 → (2)

43+17(-2) = 9 → (3)

Substitution

sub equ 2 in equ 1

(1)→ 9+8(-1) = 1 [Sub 17+9(-1) = 8]

9+(17+9(-1))(-1) = 1

9+17(-1)+9(1)=1

17(-1)+9(2) =1 → (4)

Now sub equ (3) in equ (4)

43+17(-2) = 9 → (3)

17(-1)+(43+17(-2))(2)=1

17(-1)+43(2)+17(-4)=1

17(-5)+43(2) = 1→(5)

Here -5 is the multiplicative inverse of 17. But inverse cannot be negative

17-1 mod 43 = -5 mod 43 = 38

So, 38 is the multiplicative inverse of 17.

Checking, 17* X ≡ 1 mod 43

17 * 38 ≡ 1 mod 43

646 ≡ 1 mod 43 (15*43 = 645)

Example 2

Find the Multiplicative inverse of 1635 mod 26

1635-1 mod 26

1635 = 26 (62) + 23

26 = 23 (1) + 3

23 = 3(7) + 2

3 = 2(1) + 1

Rewriting the above equation

3+2(-1) =1 → (1)

23+3(-7) = 2 → (2)

26+23(-1) = 3 → (3)

1635+26(-62) = 23 → (4)

Substitution

sub equ (2) in equ (1)

(2) => 23+3(-7) = 2

3+2(-1) =1

3+(23+3(-7))(-1) =1

3+23(-1)+3(7)=1

3(8)+23(-1) = 1 → (5)

sub equ (3) in equ (5)

26+23(-1) = 3 → (3)

(26+23(-1))(8) + 23 (-1) =1

26(8) + 23 (-8) + 23 (-1) =1

26 (8) + 23 (-9) =1 → (6)

Sub equ (4) in equ (6)

1635+26(-62) = 23 → (4)

26 (8) + (1635 + 26 (-62)) (-9) = 1

26 (8) + 1635 (-9) + 26 (558) =1

1635 (-9) + 26 (566) = 1 → (7)

From equ (7) -9 is inverse of 1635, but negative cannot be inverse.

1635-1 mod 26 = -9 mod 26 = 17

So, the inverse of 1635 is 17.

Checking, 1635* X ≡ 1 mod 26

1635 * 17 ≡ 1 mod 26

27795 ≡ 1 mod 26 (1069*26 = 27794)

2.4 CONGRUENCE AND MATRICES

Properties of Congruences

Congruences have the following properties:

1. $a = b \pmod{n}$ if $n|(a - b)$.
2. $a = b \pmod{n}$ implies $b = a \pmod{n}$.
3. $a = b \pmod{n}$ and $b = c \pmod{n}$ imply $a = c \pmod{n}$.

To demonstrate the first point, if $n|(a - b)$, then $(a - b) = kn$ for some k. So we can write $a = b + kn$. Therefore, $(a \bmod n)$ = (remainder when $b + kn$ is divided by n) = (remainder when b is divided by n) = $(b \bmod n)$.

$23 = 8 \pmod{5}$	because	$23 - 8 = 15 = 5 \times 3$
$-11 = 5 \pmod{8}$	because	$-11 - 5 = -16 = 8 \times (-2)$
$81 \equiv 0 \pmod{27}$	because	$81 - 0 = 81 = 27 \times 3$

The remaining points are as easily proved.

Matrices

Matrix is a rectangular array in mathematics, arranged in rows and columns of numbers, symbols or expressions.

A matrix will be represented with their dimensions as l x m where l defines the row and m defines the columns

$$\begin{array}{c} \boldsymbol{m} \text{ columns} \\ \boldsymbol{l} \text{ rows} \end{array} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}$$

Examples of Matrices
1. Row Matrix
2. Column Matrix
3. Square Matrix
4. Zero Matrixes
5. Identity Matrix

$\begin{bmatrix} 2 & 1 & 5 & 11 \end{bmatrix}$ Row matrix

$\begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$ Column matrix

$\begin{bmatrix} 23 & 14 & 56 \\ 12 & 21 & 18 \\ 10 & 8 & 31 \end{bmatrix}$ Square matrix

$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$ **0**

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ **I**

2.5 FINITE FIELDS

FINITE FIELDS OF THE FORM GF(p)

The finite field of order is generally written ; GF stands for Galois field,in honor of the mathematician who first studied finite fields

Finite Fields of Order p

For a given prime, , we define the finite field of order , , as the set of integers together with the arithmetic operations modulo **.**

The simplest finite field is GF(2). Its arithmetic operations are easily summarized:

+	0	1
0	0	1
1	1	0

Addition

×	0	1
0	0	0
1	0	1

Multiplication

w	$-w$	w^{-1}
0	0	—
1	1	1

Inverses

In this case, addition is equivalent to the exclusive-OR (XOR) operation, and multiplication is equivalent to the logical AND operation.

Finding the Multiplicative Inverse in It is easy to find the multiplicative inverse of an element in for small values of .You simply construct a multiplication table, such as shown in Table 2.2b,and the desired result can be read directly. However, for large values of , this approach is not practical. p p GF(p) GF(p)

Table 2.2 Arithmetic in GF(7)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

2.5.1 Polynomial Arithmetic

We are concerned with polynomials in a single variable and we can distinguish three classes of polynomial arithmetic. • Ordinary polynomial arithmetic, using the basic rules of algebra. • Polynomial arithmetic in which the arithmetic on the coefficients is performed modulo

;that is,the coefficients are in .

Polynomial arithmetic in which the coefficients are in ,and the polynomials are defined modulo a polynomial whose highest power is some integer **.**

Ordinary Polynomial Arithmetic

A polynomial of degree (integer) is an expression of the form

A **polynomial** of degree n (integer $n \geq 0$) is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

where the a_i are elements of some designated set of numbers S, called the **coefficient set**, and $a_n \neq 0$. We say that such polynomials are defined over the coefficient set S.

A zero-degree polynomial is called a **constant polynomial** and is simply an element of the set of coefficients. An nth-degree polynomial is said to be a **monic polynomial** if $a_n = 1$.

In the context of abstract algebra, we are usually not interested in evaluating a polynomial for a particular value of x [e.g., $f(7)$]. To emphasize this point, the variable x is sometimes referred to as the **indeterminate**.

Addition and subtraction are performed by adding or subtracting corresponding coefficients. Thus, if

$$f(x) = \sum_{i=0}^{n} a_i x^i; \qquad g(x) = \sum_{i=0}^{m} b_i x^i; \qquad n \geq m$$

then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^{m} (a_i + b_i)x^i + \sum_{i=m+1}^{n} a_i x^i$$

and multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{k-1} b_1 + a_k b_0$$

As an example, let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$, where S is the set of integers. Then

$$f(x) + g(x) = x^3 + 2x^2 - x + 3$$
$$f(x) - g(x) = x^3 + x + 1$$
$$f(x) \times g(x) = x^5 + 3x^2 - 2x + 2$$

Polynomial Arithmetic with Coefficients in

Let us now consider polynomials in which the coefficients are elements of some field F; we refer to this as a polynomial over the field F. In that case, it is easy to show that the set of such polynomials is a ring, referred to as a polynomial ring. That is, if we consider each distinct polynomial to be an element of the set, then that set is a ring 8 when polynomial arithmetic is performed on polynomials over a field, then division is possible. Note that this does not mean that exact division is possible. Let us clarify this distinction. Within a field, given two elements and, the quotient is also an element of the field. However, given a ring that is not a field, in Ra /b ba Zp

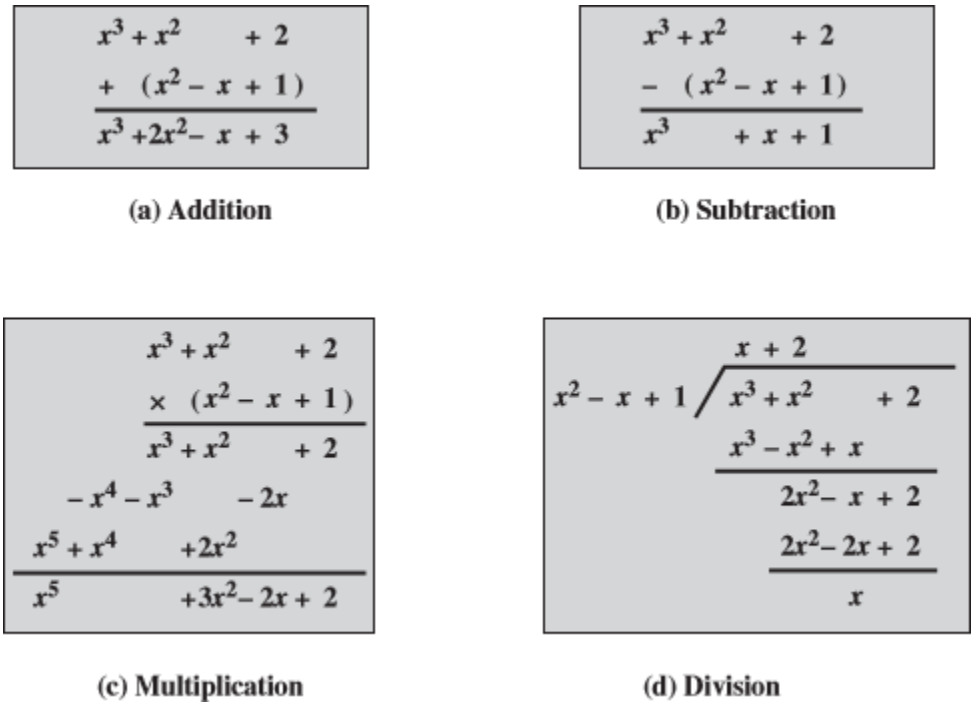


Figure 2.3 Examples of Polynomial Arithmetic

A polynomial over a field is called irreducible if and only if cannot be expressed as a product of two polynomials, both over, and both of degree lower than that of. By analogy to integers, an irreducible polynomial is also called a prime polynomial.

2.6 SYMMETRIC KEY CIPHERS

Symmetric ciphers use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. They are faster than asymmetric ciphers and allow encrypting large sets of data. However, they require sophisticated mechanisms to securely distribute the secret keys to both parties

Definition **A symmetric cipher** defined over (K, M, C), where:

- K - a set of all possible keys,
- M - a set of all possible messages,
- C - a set of all possible ciphertexts

is a pair of efficient algorithms (E, D), where:

- $E: K \times M \rightarrow C$
- $D: K \times C \rightarrow M$

such that for every m belonging to M, k belonging to K there is an equality:

- $D(k, E(k, m)) = m$ (the consistency rule)

➡ *Function E is often randomized*

➡ *Function D is always deterministic*

Types of keys are used in symmetric key cryptography

Symmetric encryption (figure 2.4) uses a single key that needs to be shared among the people who need to receive the message while asymmetrical encryption uses a pair of public key and a private key to encrypt and decrypt messages when communicating.

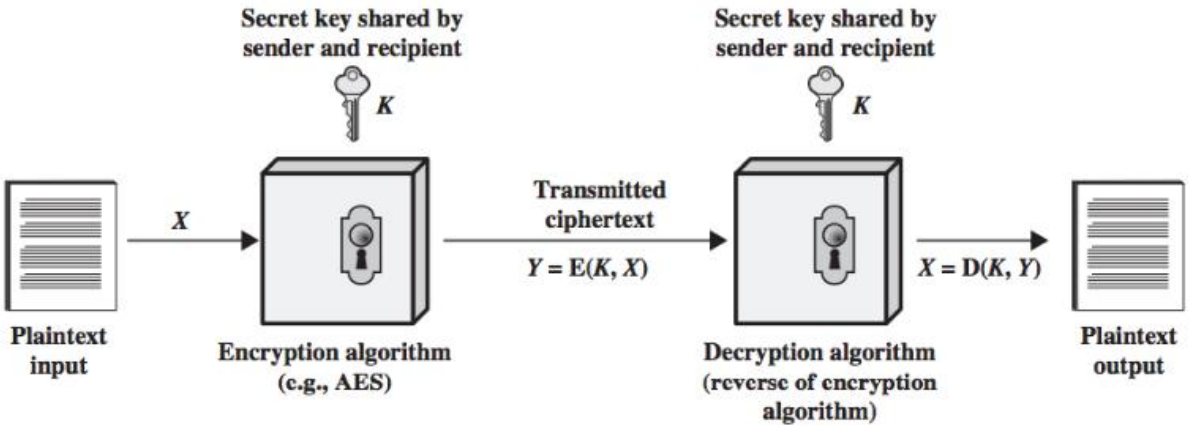


Figure 2.4 Simplified Model of Symmetric Encryption

2.7 SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES)

The overall structure of the simplified DES shown in Figure 2.5. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output.

The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

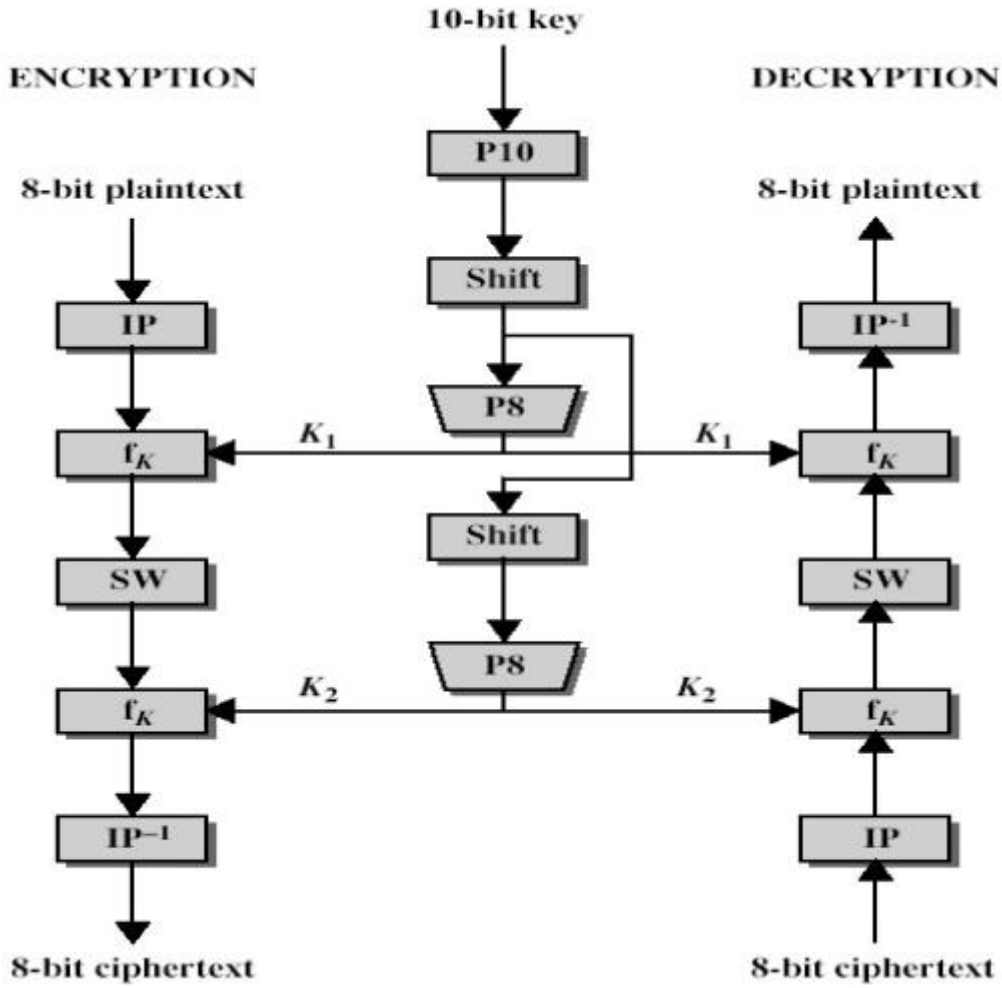


Figure 2.5 Overview of S-DES Algorithm

The encryption algorithm involves five functions:

- An initial permutation (IP)
- A complex function labeled fk, which involves both permutation and substitution operations and depends on a key input.
- A simple permutation function that switches (SW) the two halves of the data.
- The function fk again.

A permutation function that is the inverse of the initial permutation

The function fk takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated.

The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1).

The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition1 of functions:

IP-1 o fK2 o SW o fk1 o IP, which can also be written as

Ciphertext = IP-1 (fK2 (SW (fk1 (IP (plaintext)))))

Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as Plaintext = IP-1 (fK1 (SW (fk2 (IP (ciphertext)))))

2.7.1 S-DES Key Generation

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm.(Figure 2.6)

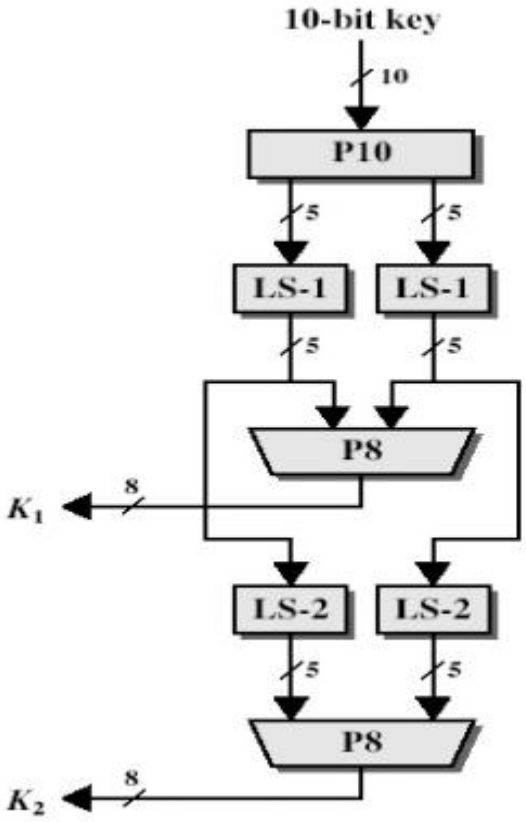


Figure 2.6 S-DES Key Generation

First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10 (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, K2, k7, k4, k10 10, k1, k9, k8, k6).

P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So, the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on.

Example

The 10 bit key is (1010000010), now find the permutation from P10 for this key so it becomes (10000 01100).

Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000).

Next, apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

So, The result is subkey 1 (K1). In our example, this yield (10100100).

Then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011).

Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

2.7.2 S-DES Encryption

Encryption involves the sequential application of five functions (Figure 2.7).

1. **Initial Permutations**

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function

IP							
2	6	3	1	4	8	5	7

The plaintext is 10111101

Permutated output is 01111110

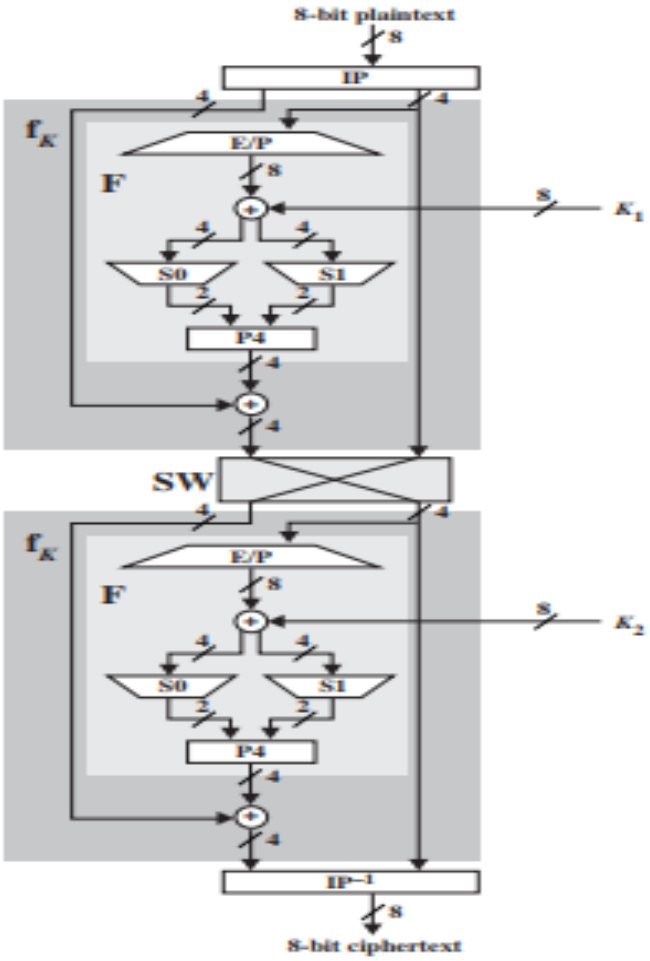


Figure 2.7 S-DES Encryption

2. The Function f_k

The most complex component of S-DES is the function fk, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f K, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$Fk (L, R) = (L \oplus F (R, SK), R)$$

Where SK is a sub key and ⊕is the bit-by- bit exclusive OR function

Now, describe the mapping F. The input is a 4-bit number (n1 n2 n3 n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

Now, find the E/P from IP

IP = 01111110, it becomes

E/P = 01111101

Now, XOR with K1

=> 01111101 ⊕10100100 = 11011001

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the S-box. Each s box gets 4-bit input and produce 2 bits as output. It follows 00- 0, 01-1, 10-2, 11-3 scheme.

Here, take first 4 bits,

S_0 => 1101

11 - > 3

10 -> 2 } => 3 =>11

So, we get 1110

- Now, find P_4

Second 4 bits

S1 => 1001

11 -> 3

00 -> 0 = > 2 => 10

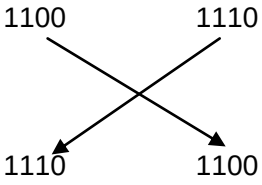
P4			
2	4	3	1

After P_4, the value is 1011

Now, XOR operation 1011⊕ 0111 => 1100

3. The Switch function

- The switch function (sw) interchanges the left and right 4 bits.



.

4. Second function f_k

- First, do E/P function and XOR with K_2, the value is 01101001⊕01000011, the answer is 00101010
- Now, find S_0 and S_1

S_0 => 00 - > 0

01 -> 1 } => 0 = 00

S_1 = > 10 -> 2

01 -> 1 = > 0 => 00

Value is 0000

- Now, find P_4 and XOR operation

After P_4 => 0000 ⊕ 1110 = 1110, then concatenate last 4 bits after interchange in sw.

- Now value is 11101100

5. Find IP^{-1}

IP –1							
4	1	3	5	7	2	8	6

So, value is 01110101

The Ciphertext is 01110101

2.8.3 S-DES Decryption

- Decryption involves the sequential application of five functions.

1. **Find IP**
   - After IP, value is 11101100
2. **Function f_k**
   - After step 2, the answer is 11101100
3. **Swift**
   - The answer is 11001110
4. **Second f_k**
   - The answer is 01111110
5. **Find IP-1**
   - **101111101 -> Plaintext**

2.8 DATA ENCRYPTION STANDARD

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977. The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output.

2.8.1 DES Encryption

The overall scheme for DES encryption is illustrated in the Figure 2.8. There are two inputs to the encryption function: the **plaintext** to be encrypted and the **key**. The plaintext must be 64 bits in length and the key is 56 bits in length.

2.8.2 General Depiction of DES Encryption Algorithm

Phase 1

Looking at the left-hand side of the figure 2.8, we can see that the processing of the plaintext proceeds in three phases.

First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.

Phase 2:

This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions.

The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput.

Phase 3:

Finally, the preoutput is passed through a permutation (IP^{-1}) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.
The right-hand portion of Figure shows the way in which the 56-bit key is used.

Operation on key:

Initially, the key is passed through a permutation function. Then, for each of the 16 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.

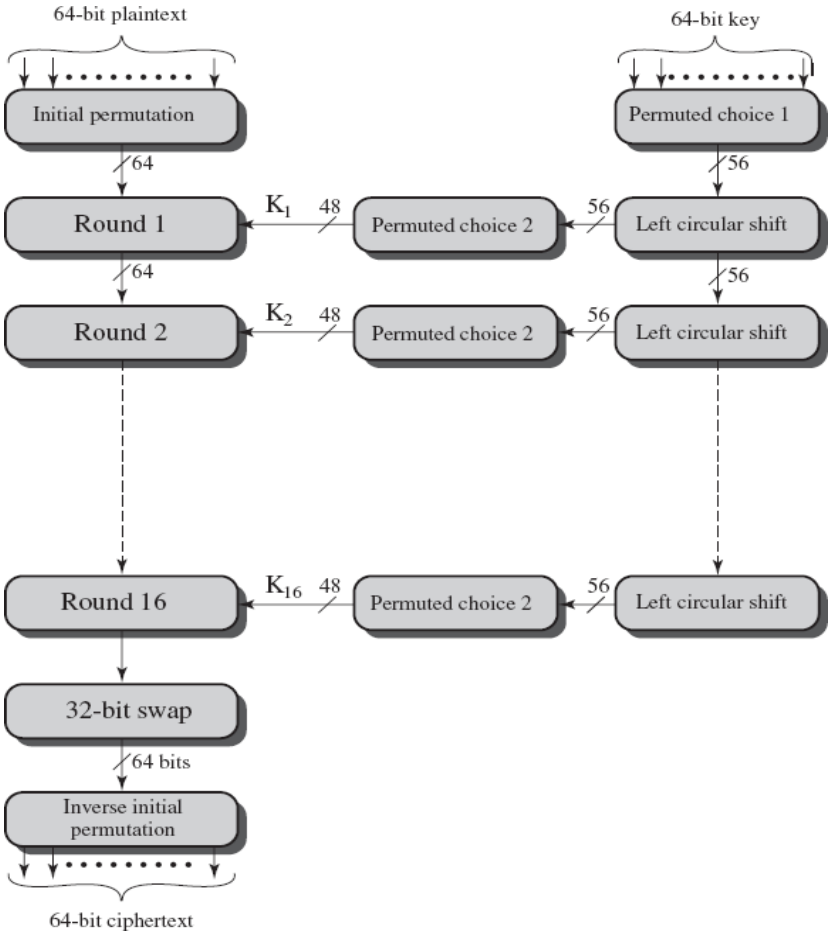


Figure 2.8 DES Encryption Algorithm

Initial Permutation

The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Permutation Tables for DES

(a) Initial Permutation (IP)

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

Inverse Initial Permutation (IP^{-1})

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

Expansion Permutation (E)

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

Permutation Function (P)

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

Consider the following 64-bit input *M*:

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

where *Mi* is a binary digit. Then the permutation $X = IP(M)$ is as follows:

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

Inverse permutation $Y = IP^{-1}(X) = IP^{-1}(IP(M))$,Therefore we can see that the original ordering of the bits is restored.

2.8.3 Details of Single Round

The below figure 2.9 shows the internal structure of a single round. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). The overall processing at each round can be summarized in the following formulas:

$L_i = R_{i-1}$

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

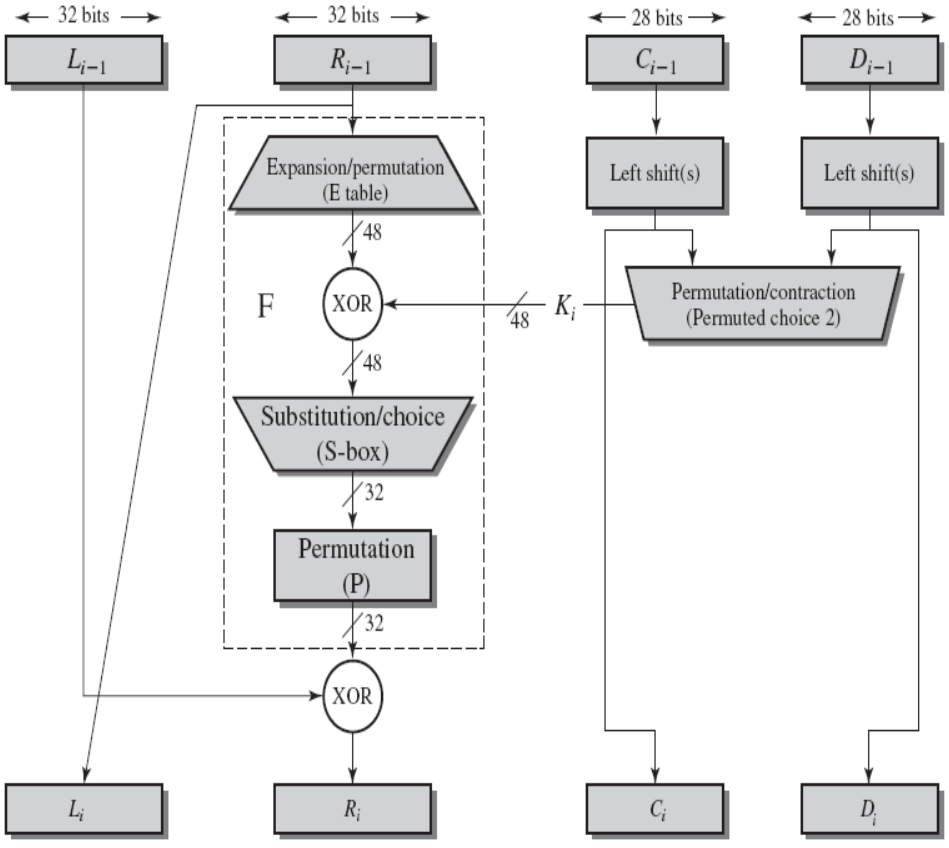


Figure 2.9 Single Round of DES Algorithm

The round key *Ki* 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 *Ki*. This 48-bit result passes through a substitution function that produces a 32-bit output, which is then permuted.

Definition of S-Boxes

The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. The first and last bits of the input to box S*i* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S*i*. The middle four bits select one of the sixteen columns as shown in figure 2.10.

The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output.

For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

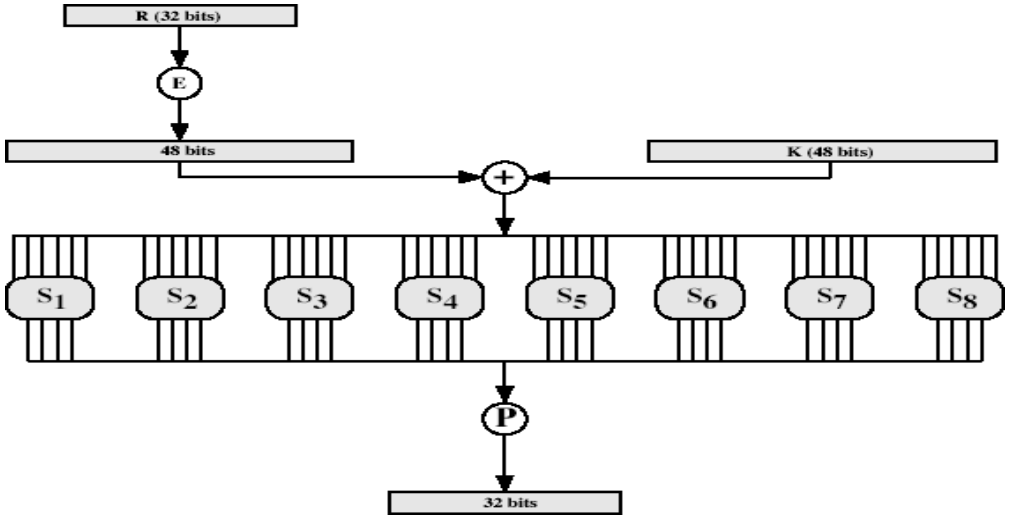


Fig 2.10 Calculation of F(R, K)

2.8.4 Key Generation

The 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. The key is first subjected to a permutation governed by a table labeled Permuted Choice One. The resulting 56-bit key is then treated as two 28-bit quantities, labeled *C*0 and *D*0.

At each round, *Ci*-1 and *Di*-1 are separately subjected to a circular left shift, or rotation, of 1 or 2 bits. These shifted values serve as input to the next round. They also serve as input to Permuted Choice 2, which produces a 48-bit output that serves as input to the function F(*Ri*-1, *Ki*).

DES Key Schedule Calculation

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

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts

Roundnumber:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

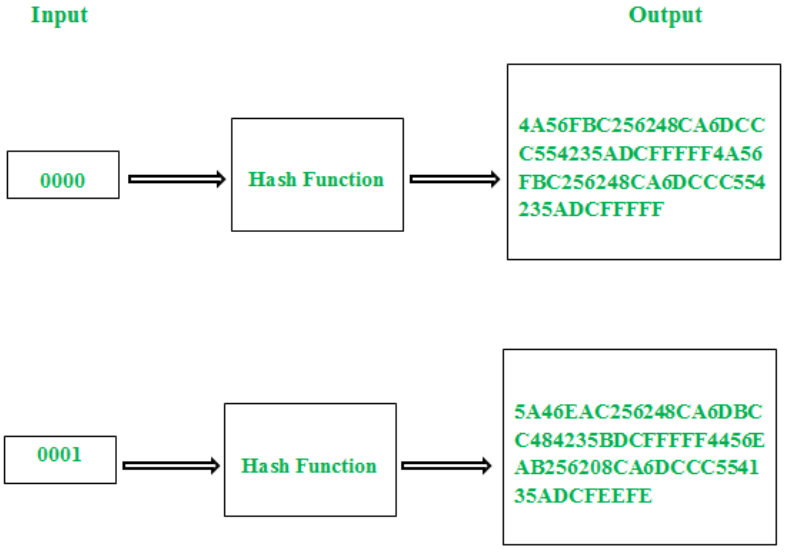
Bits rotated : 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

2.8.5 DES Decryption:

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

2.8.6 The Avalanche Effect:

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext.



2.9 THE STRENGTH OF DES

The strength of DES depends on two factors: **key size** and the **nature of the algorithm**.

1. **The Use of 56-Bit Keys**

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2 x 10^{16}. Thus, a brute-force attack appears impractical.

2. **The Nature of the DES Algorithm**

In DES algorithm, eight substitution boxes called S-boxes that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

3. **Timing Attacks**

A 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. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

2.9.1 Attacks on DES:

Two approaches are:

1. Differential crypt analysis
2. Linear crypt analysis

2.9.1.1 Differential Cryptanalysis

Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 255 complexities. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P.

- One of the most significant recent (public) advances in cryptanalysis
- Powerful method to analyze block ciphers
- Used to analyze most current block ciphers with varying degrees of success

Differential Cryptanalysis Attack:

The differential cryptanalysis attack is complex. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block.

Consider the original plaintext block *m* to consist of two halves *m*0, *m*1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round.

So, at each round, only one new 32-bit block is created. If we label each new block $m_1(2 \le i \le 17)$, then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i), \; i = 1, 2, ..., 16$$

In differential cryptanalysis, we start with two messages, *m* and *m*', with a known XOR difference $\Delta m = m \oplus m'$, and consider the difference between the intermediate message halves: $m_i = m_i \oplus m_i'$ Then we have:

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m''_{i-1} \\ &= [m_{i-1} \oplus f(m_i, k_i)] \oplus [m''_{i-1} \oplus f(m''_i, k_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, k_i) \oplus f(m''_i, ki)]\end{aligned}$$

Let us suppose that there are many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used.

Therefore, if we know Δm_{i-1} and Δm_i with high probability, then we know Δm_{i+1} with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f.

2.9.1.2 Linear Cryptanalysis

This attack is based on the fact that linear equation can be framed to describe the transformations.

The principle of linear crypt analysis is as follows

Length of CT and PT =n bits;

key=mbit

Block of cipher text is c[1]c[2]…c[n]; Block of key is k[1]k[2]….k[m]

A[I,j,..k] = A[i] ⊕A[j] ⊕. ⊕A[k]

- Can attack DES with 247 known plaintexts, still in practice infeasible
- Find linear approximations with prob p != ½
- $P[i_1,i_2,...,i_a](+)c[j_1,j_2,...,j_b] = k[k_1,k_2,...,k_c]$Where i_a, j_b, k_c are bit locations in p, c, k

2.10 BLOCK CIPHER PRINCIPLES

There are three critical aspects of block cipher design:

1. Number of rounds,
2. Design of the function F
3. Key scheduling.

Number of Rounds

- When the greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F.
- The number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack
- When round DES S= 16, a differential cryptanalysis attack is slightly less efficient than brute force, the differential cryptanalysis attack requires 2^{55} operations.
- It makes it easy to judge the strength of an algorithm and to compare different algorithms.

Design of Function F

This is the most important function

Criteria needed for F,

- It must be difficult to "unscramble" the substitution performed by F.
- The function should satisfy **strict avalanche criterion (SAC)** which states that any output bit *j* of an S-box should change with probability 1/2 when any single input bit *i* is inverted for all *i*, *j*.
- The function should satisfy **bit independence criterion(BIC)**, which states that output bits *j* and *k* should change independently when any single input bit *i* is inverted for all *i*, *j*, and *k*.

Key Schedule Algorithm

- The key is used to generate one sub key for each round.
- The sub keys to maximize the difficulty of deducing individual sub keys and the difficulty of working back to the main key.

2.10.1 Stream Cipher and Block Cipher

A stream cipher is one that encrypts a digital data stream one bit or one byte at a time.
E.g, vigenere cipher. Figure (2.11a)

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically, a block size of 64 or 128 bits is used. Figure (2.11b)

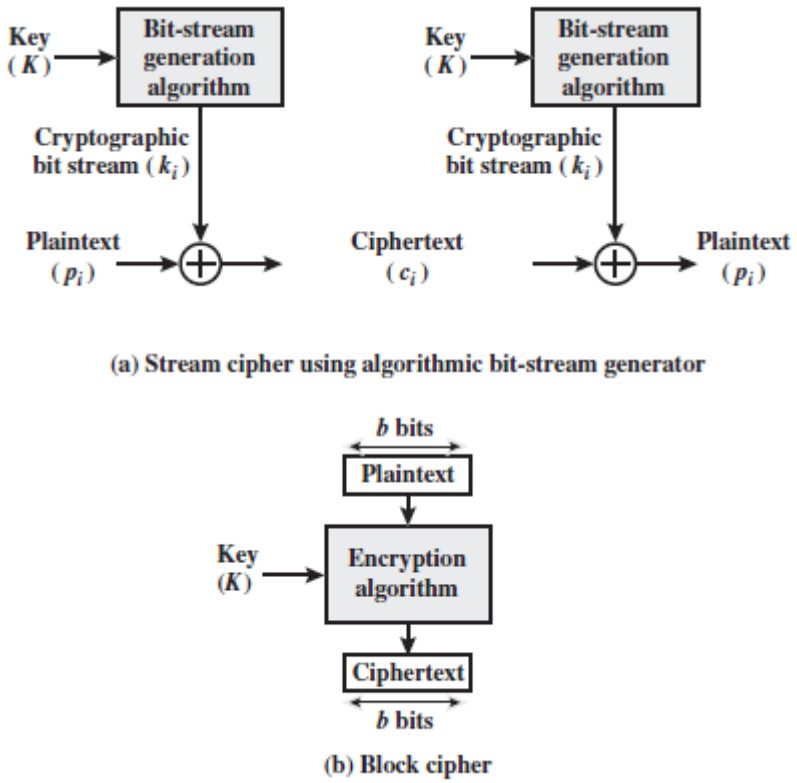
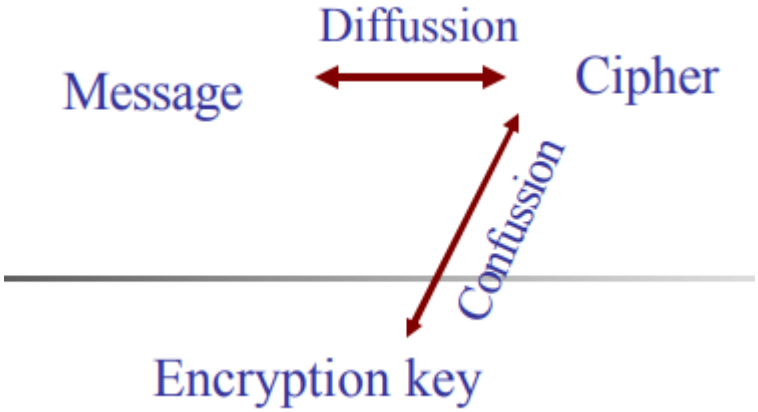


Figure 2.11 Stream Cipher and Block Cipher

- Many block ciphers have a Feistel structure. Such a structure consists of a number of identical rounds of processing.
- In each round, a substitution is performed on one half of the data being processed, followed by a permutation that interchanges the two halves.
- The original key is expanded so that a different key is used for each round.
- The Data Encryption Standard (DES) has been the most widely used encryption algorithm. It exhibits the classic Feistel structure.
- The DES uses a 64-bit block and a 56-bit key. Two important methods of cryptanalysis are differential cryptanalysis and linear cryptanalysis. DES has been shown to be highly resistant to these two types of attack.
- A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits. There are 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 non singular
- 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.

- Two methods for frustrating statistical cryptanalysis are:
  - **Diffusion –** Each plaintext digit affects many ciphertext digits, or each ciphertext digit is affected by many plaintext digits.
  - **Confusion** – Make the statistical relationship between a plaintext and the corresponding ciphertext as complex as possible in order to thread attempts to deduce the key.



2.10.2 Feistel cipher structure

- The left-hand side of figure 2.12 depicts the structure proposed by Feistel.
- The input to the encryption algorithm is a plaintext block of length 2w 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 inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as the subkey K_i, derived from the overall key K.
- In general, the subkeys K_i 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 (as similar to S-DES). This is done by applying a round function F to the right half of the data and then taking the 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.
- This structure is a particular form of the substitution-permutation network.

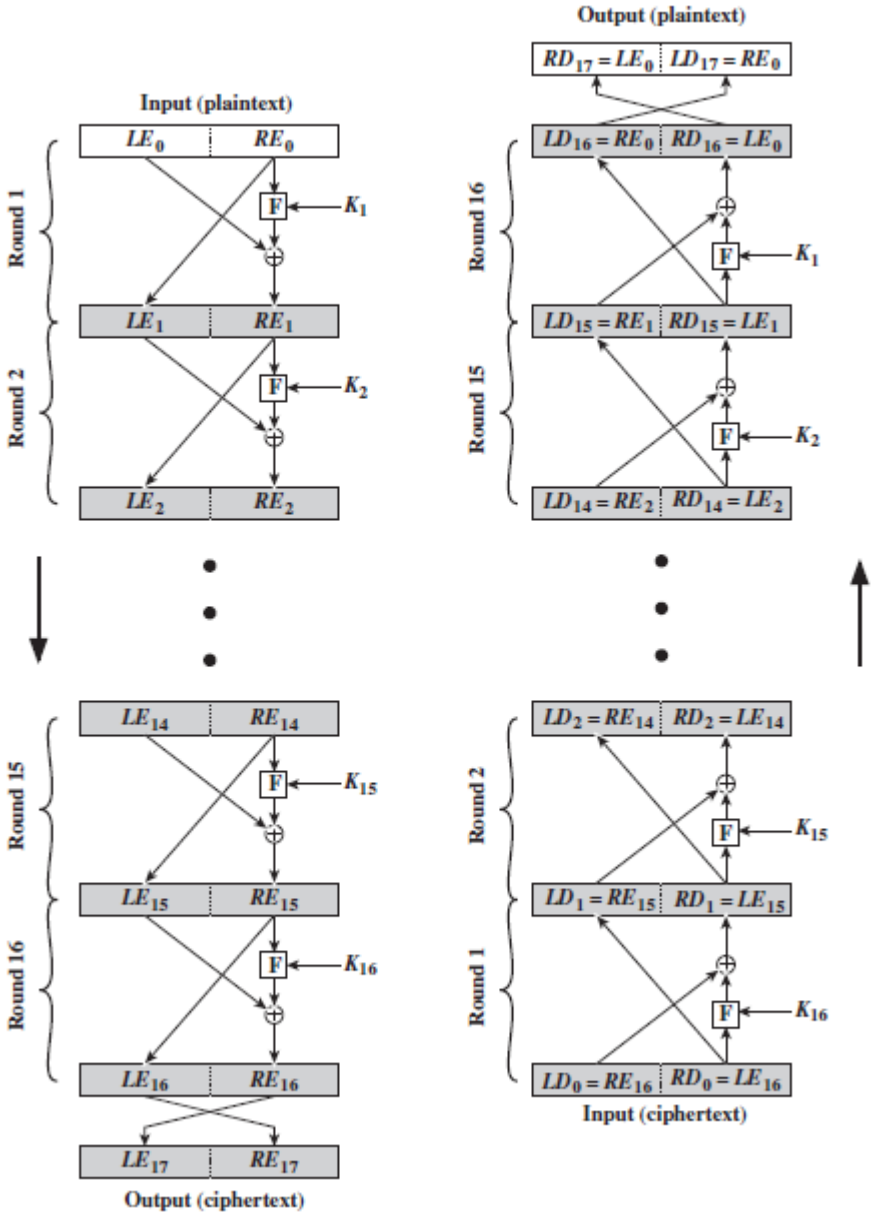


Figure 2.12 Feistel Encryption and Decryption (16 rounds)

The features of Feistel network are:

- **Block size** - Increasing size improves security, but slows cipher
- **Key size** - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **Number of rounds** - Increasing number improves security, but slows cipher
- **Subkey generation** - Greater complexity can make analysis harder, but slows cipher
- **Round function** - Greater complexity can make analysis harder, but slows cipher

- The process of decryption is essentially the same as the encryption process.
- The rule is as follows: use the cipher text as input to the algorithm, but use the subkey k_i in reverse order. i.e., k_n in the first round, k_{n-1} in second round and so on.
- For clarity, we use the notation LE_i and RE_i for data traveling through the decryption algorithm and LD_i and RD_i.
- The above diagram indicates that, at each round, the intermediate value of the decryption process is same (equal) to the corresponding value of the encryption process with two halves of the value swapped.

i.e., $RE_i \| LE_i$ (or) equivalently $RD_{16-i} \| LD_{16-i}$

- After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is $RE_{16} \| LE_{16}$.
- The output of that round is the cipher text. Now take the cipher text and use it as input to the same algorithm.
- The input to the first round is $RE_{16} \| LE_{16,}$ which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.
- Now we will see how the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process.
- First consider the encryption process,

$$\mathbf{LE_{16} = RE_{15}}$$
$$\mathbf{RE_{16} = LE_{15} \oplus F\,(RE_{15}, K_{16})}$$

On the decryption side,

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$
$$RD_1 = LD_0 \oplus F\,(RD_O, K_{16})$$
$$= RE_{16} \oplus F\,(RE_{15}, K_{16})$$
$$= [LE_{15} \oplus F\,(RE_{15}, K_{16})] \oplus F\,(RE_{15}, K_{16})$$
$$= LE_{15}$$

Therefore, $LD_1 = RE_{15}$, $RD_1 = LE_{15}$

In general, for the ith iteration of the encryption algorithm,

$$\mathbf{LE_i = RE_{i-1}}$$
$$\mathbf{RE_i = LE_{i-1} \oplus F\,(RE_{i-1}, K_i)}$$

- Finally, the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext.

2.11 BLOCK CIPHER MODES OF OPERATION

- Block Cipher is the basic building block to provide data security.
- To apply the block cipher to various applications, NIST has proposed 4 modes of operation. The block cipher is used to enhance the security of the encryption algorithm

2.11.1 Multiple Encryption and Triple DES

The vulnerability of DES to a brute-force attack has been detected by using two approaches are shown in figure 2.13

1. One approach is to design a completely new algorithm, of which AES is a prime example
2. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryptions with DES and multiple keys.

Double DES

The simplest form of multiple encryptions has two encryption stages and two keys. Given a plaintext P and two encryption keys K_1 and K_2, cipher text C is generated as

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

Decryption requires that the keys be applied in reverse order:

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

For DES, this scheme apparently involves a key length of 56 * 2 = 112 bits, resulting in a dramatic increase in cryptographic strength.

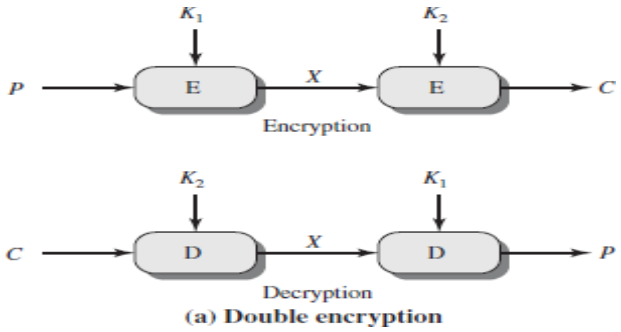


Figure 2.13 Multiple Encryption

Reduction to a Single Stage

Suppose it were true for DES, for all 56-bit key values, that given any two keys K_1 and K_2, it would be possible to find a key K_3 such that

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

Meet-in-the-Middle Attack

The use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is a way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher. This algorithm, known as a meet-in-the-middle attack.

It is based on the observation that, if we have

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

Then

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

Given a known pair, (P, C), the attack proceeds as follows. First, encrypt P for all 256 possible values of K_1. Store these results in a table and then sort the table by the Values of X.

Next, decrypt C using all 256 possible values of K2. As each decryption is produced,

check the result against the table for a match.

If a match occurs, then test the two resulting keys against a new known plaintext–cipher text pair. If the two keys produce the correct cipher text, accept them as the correct keys.

For any given plaintext P, there are 264 possible cipher text values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that there are 2112 possible keys.

Triple DES with Two Keys

To overcome the meet-in-the-middle attack is to use three stages of encryption with three different keys. This is called ad Triple DES or 3DES as shown in figure 2.14.

The known plain text attack in 2^{112}. The key length of 56 * 3 = 168 bits which is a drawback.

Tuchman proposed a triple encryption method that uses only two keys given plain text k_1,k_2 . The final cipher text is

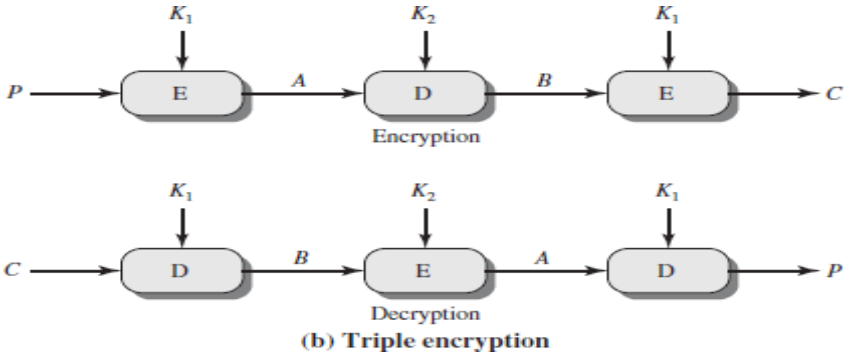
$$C = \mathrm{E}(K_1, \mathrm{D}(K_2, \mathrm{E}(K_1, P)))$$
$$P = \mathrm{D}(K_1, \mathrm{E}(K_2, \mathrm{D}(K_1, C)))$$

- The function follows an encrypt-decrypt-encrypt (EDE)sequence

Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

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

- 3DES with two keys is a relatively popular alternative to DES
- There are no practical cryptanalytic attacks on 3DES.
- The cost of a brute-force key search on 3DES is on the order of2^{112}



(b) Triple encryption

Figure 2.14 Triple DES

The first serious proposal came from Merkle and Hellman

1. Merkle and Hellman

The concept is to find plaintext values that produce a first intermediate value of A = 0 and then using the meet-in-the-middle attack to determine the two keys.

- The level of effort is 2^{56},
- The technique requires 256 chosen plaintext–cipher text pairs, which is a number unlikely to be provided.

2. known - plaintext attack:

The attack is based on the observation that if we know A and C then the problem reduces to that of an attack on double DES.

The attacker does not know A, even if P and C are known, as long as the two keys are unknown. The attacker can choose a potential value of A and then try to find a known (P, C) pair that produces A.

The attack proceeds as follows.

Step 1:

- Obtain n (P, C) pairs. This is the known plaintext. Place these in a table sorted on the values of P

Step 2:

- Pick an arbitrary value a for A, and create a second table with entries defined in the following fashion.
- For each of the 2^{56} possible keys K_1 = i, calculate the plaintext value Pi that produces a.
- For each P_i that matches an entry in Table 1, create an entry in Table 2 consisting of the K_1 value and the value of B that is produced.

Step 3:

- We now have a number of candidate values of K_1 in Table 2 and are in a position to search for a value of K_2.
- For each of the 256 possible keys K_2 = j, calculate the second intermediate value for our chosen value of a
- If there is a match, then the corresponding key i from Table 2 plus this value of j are candidate values for the unknown keys (K1, K2).

Step 4:

- Test each candidate pair of keys (i, j) on a few other plaintext–cipher text pairs.
- If a pair of keys produces the desired cipher text, the task is complete. Ifno pair succeeds, repeat from step 1 with a new value of a.

2.11.2 MODE 1: Electronic Code Book

The simplest mode is the electronic codebook (ECB) mode shown in figure 2.15. Here plaintext is handled one block at a time and each block of plaintext is encrypted using the same key.

The term codebook is used because, for a given key, there isa unique cipher text for every b-bit block of plaintext.

When the message longer than b bits, to break the message into b-bit blocks. For the last block when the no of bits is less than b, padding the last block if necessary.

Decryption is performed one block at a time, always using the same key.

Uses: The ECB method is ideal for a short amount of data, such as an encryption key.

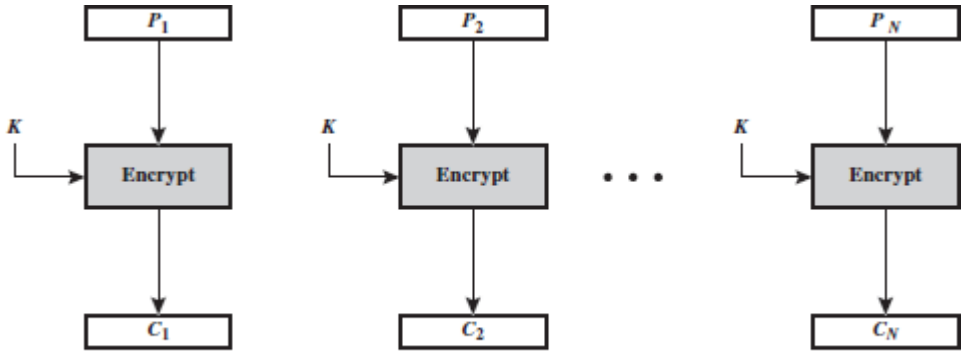
Disadvantage:

When b‟ -bit block of plaintext appears more than once in the message, it always produces the same cipher text output.

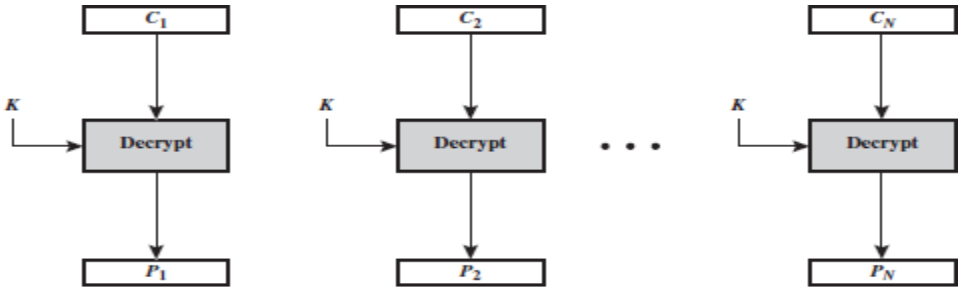
For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities.

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.



(a) Encryption



(b) Decryption

Figure 2.15 Electronic Code Book (ECB)

Mode Properties for Evaluating and Constructing ECB

Overhead: The additional operations for the encryption and decryption operation when compared to encrypting and decrypting in the ECB mode.

Error recovery: The property that an error in the *i*th cipher text block is inherited by only a few plaintext blocks

Error propagation: It is meant here is a bit error that occurs in the transmission of a cipher text block, not a computational error in the encryption of a plaintext block.

Diffusion: Low entropy plaintext blocks should not be reflected in the cipher text blocks. Roughly, low entropy equates to predictability or lack of randomness

Security: Whether or not the cipher text blocks leak information about the plaintext blocks.

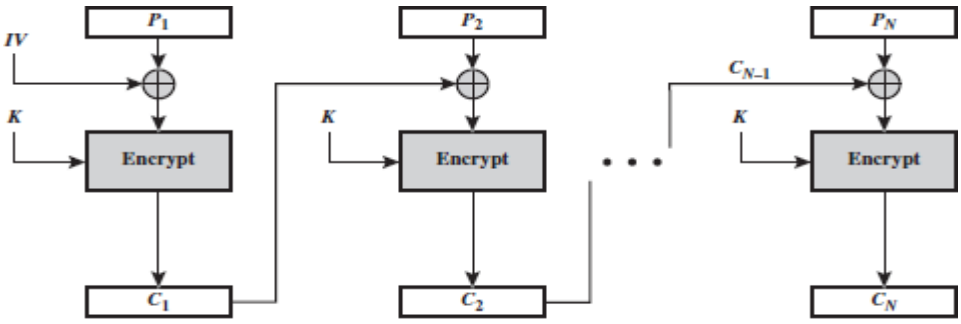
2.11.3 MODE 2: Cipher Block Chaining Mode

This method is to overcome the disadvantage of ECB (i.e) when the PT block is repeated CBC produces different cipher text blocks

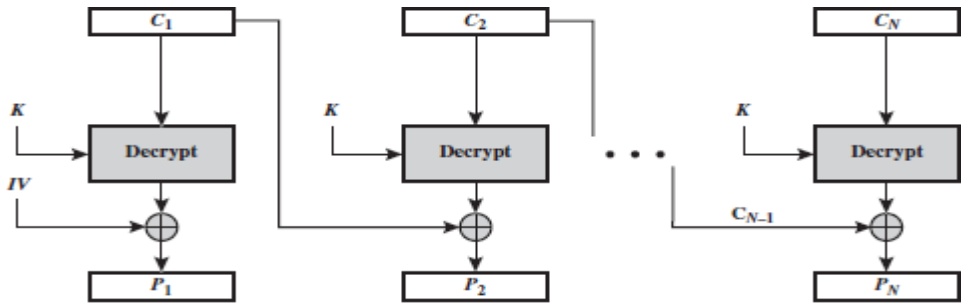
The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding cipher text block to produce the plaintext block are shown in figure 2.16.

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



(a) Encryption



(b) Decryption

Figure 2.16 Cipher Block Chaining (CBC) Mode

Then

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

To produce the first block of cipher text, 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.

Size of IV = Size of data Blocks

We can define CBC mode as

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

For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption

Reason for protecting the IV:

If an opponent is able to fool the receiver in to using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider

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

Now use the notation that X[i] denotes the ith bit of the b-bit quantity X. Then

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

Then, using the properties of XOR, we can state

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

Where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P1 can be changed.

2.11.4 MODE 3: Cipher Feedback Mode:

We know that the DES is a block cipher.it is possible to convert block cipher into stream Cipher using CFB mode

The advantages of CFB is that

- Eliminates the need to pad a message
- It also can operate in real time
- The length of the CT =Length of PT

Figure 2.17 depicts the CFB scheme. In the figure 2.17, it is assumed that the unit of transmission is s bits; a common value is s = 8.

The units of plaintext are chained together; to get the cipher text is a function of all preceding plaintext. Here the plaintext is divided into segments of s bits.

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 segment of plaintext P1 to produce the first unit of cipher text C1.

The contents of the shift register are shifted left by s bits, and C1 is placed in the rightmost (least significant) s bits of the shift register.

This process continues until all plaintext units have been encrypted.

Decryption:

The same scheme is used, except that the received cipher text unit is XORed with the output of the encryption function to produce the plaintext unit.

Let MSBs(X) be defined as the most significant s bits of X. Then

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

Therefore, by rearranging terms:

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

The same reasoning holds for subsequent steps in the process.

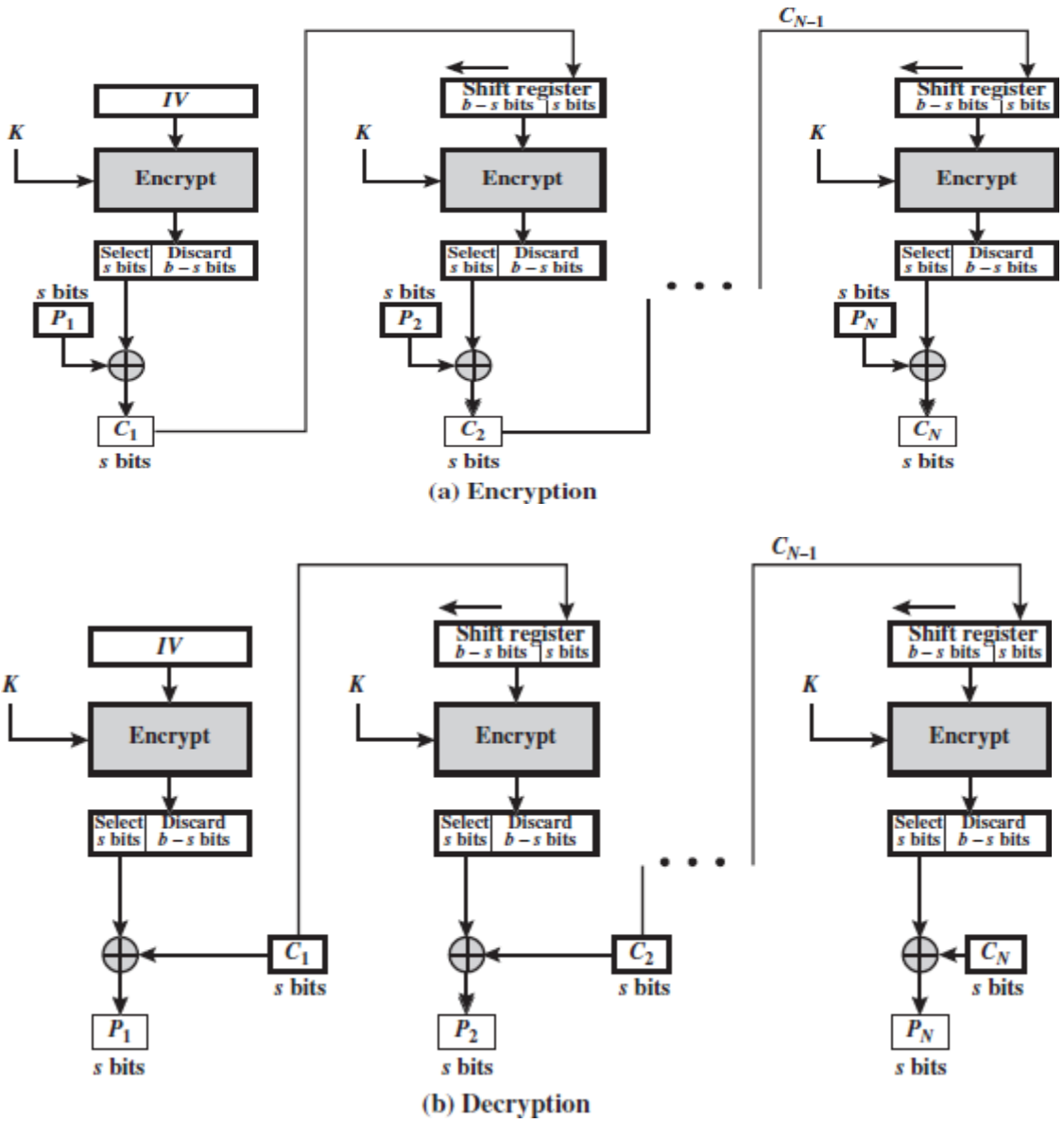


Figure 2.17 S-bit Cipher Feedback (CFB) mode

We can define CFB mode as follows

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

2.11.5 Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB.

The output of the encryption function is fed back to become the input for encrypting the next block of plaintext as shown in figure 2.18.

Comparison between OFB and CFB

In CFB, the output of the XOR unit is fed back to become input for encrypting the next block.

The other difference is that the OFB mode operates on full blocks of plaintext and cipher text, whereas CFB operates on an s-bit subset. OFB encryption can be expressed as

Where

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

$$O_{j-1} = \mathrm{E}(K, O_{j-2})$$

we can rewrite the encryption expression as:

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

By rearranging terms, we can demonstrate that decryption works.

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

We can define OFB mode as follows.

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

Let the size of a block be b. If the last block of plaintext contains u bits (indicated by *), with u<b, the most significant u bits of the last output block O_N are used for the XOR operation The remaining b - u bits of the last output block are discarded.

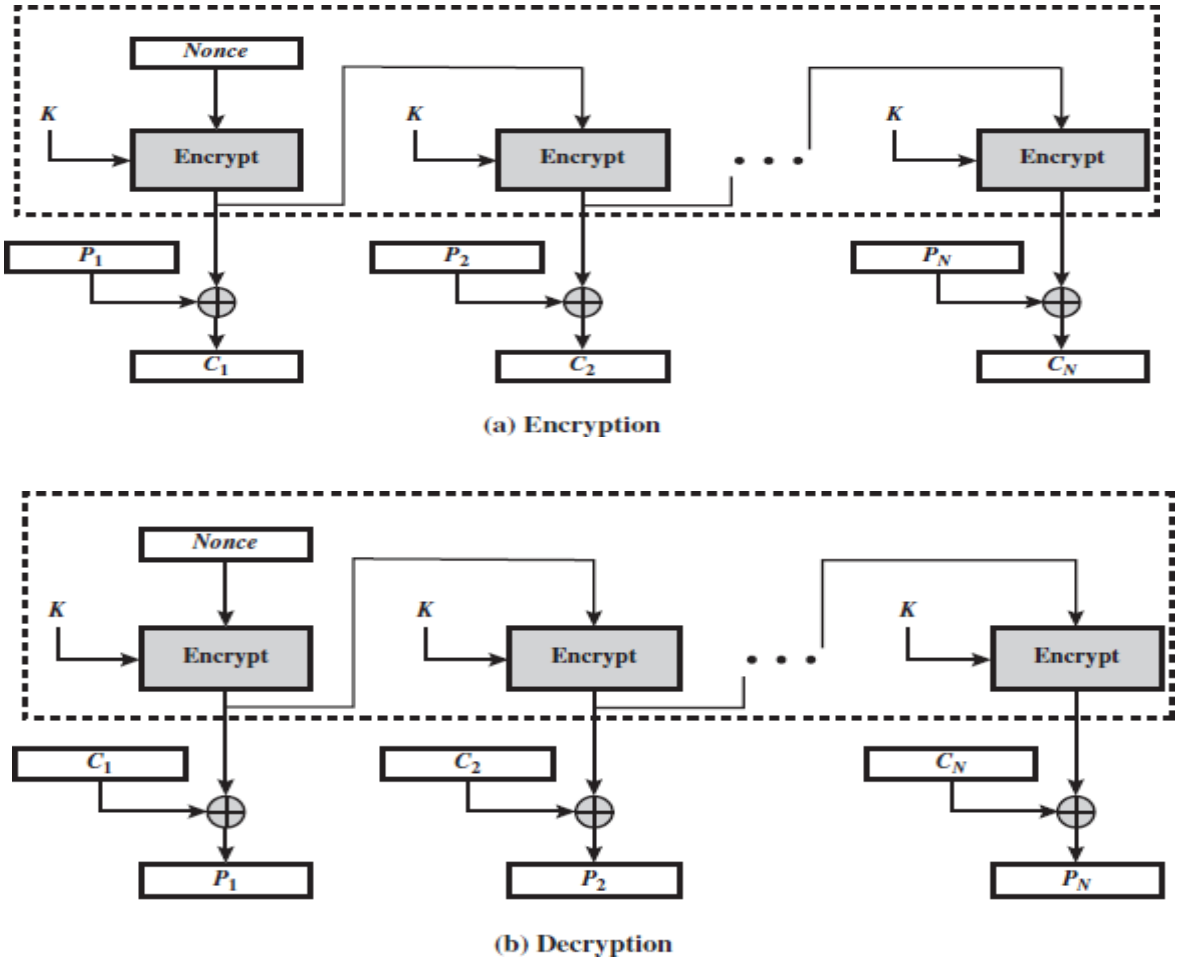


Figure 2.18 Output Feedback Mode

Advantage:

Bit errors in transmission do not propagate (i.e.) when bit errors occurs in Ci, Pi is alone affected

Disadvantage:

Vulnerable to message stream modification attack

2.11.6 Counter Mode

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 counter value must be different for each plaintext block as shown in figure 2.19.

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 cipher text block.

For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a cipher text block to recover the corresponding plaintext block.

Advantage:

Hardware efficiency

- CTR can be done in parallel

Software efficiency

- CTR supports parallel feature pipelining

Preprocessing

Simplicity

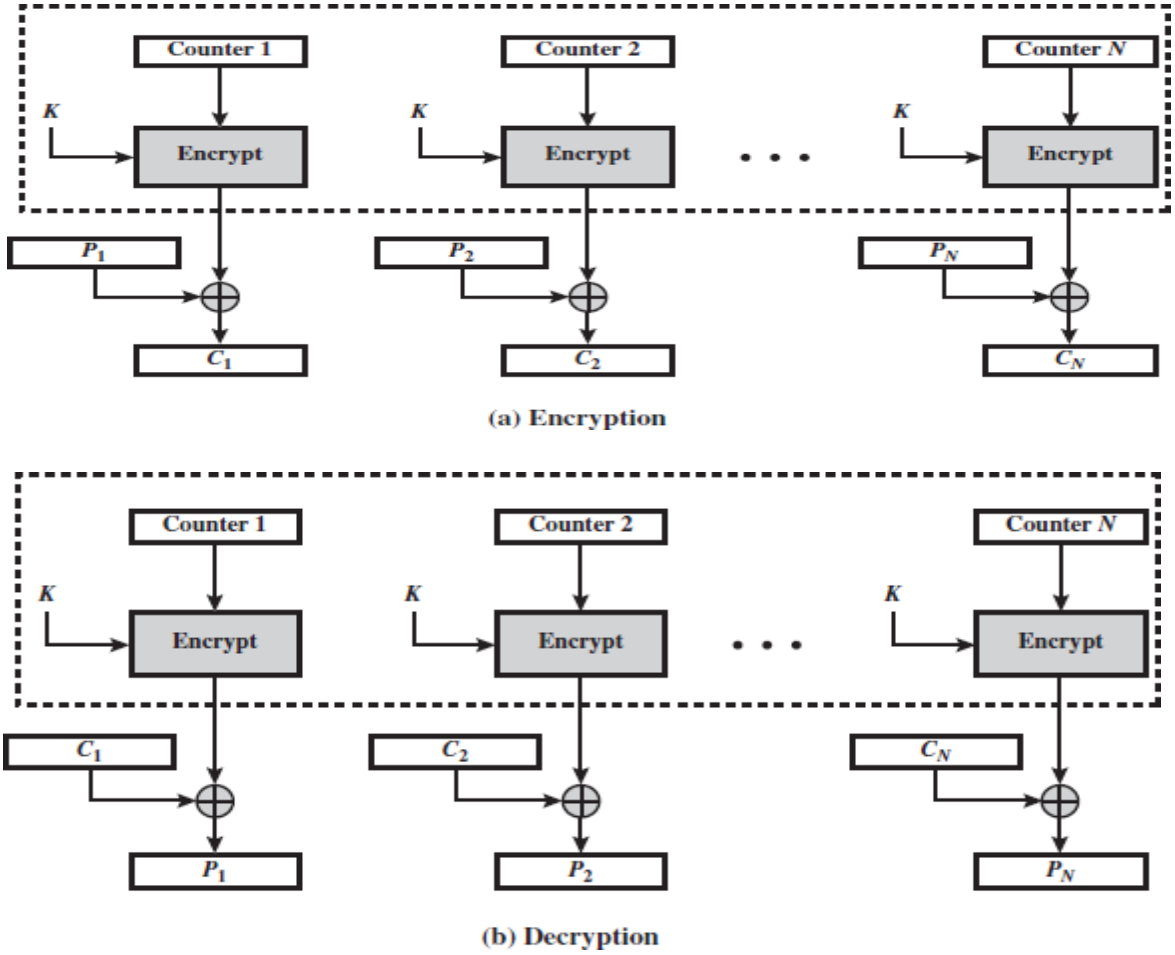


Figure 2.19 Counter Mode

2.12 ADVANCED ENCRYPTION STANDARD (AES)

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic, algorithms.

2.12.1 Finite Field Arithmetic

In AES, all operations are performed on 8-bit bytes. The arithmetic operations of addition, multiplication, and division are performed over the finite field GF.A field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: *a*/*b* = *a*(*b*-1).

An example of a finite field (one with a finite number of elements) is the set Z*p* consisting of all the integers {0, 1, c, *p* - 1}, where *p* is a prime number and in which arithmetic is carried out modulo *p*.

The way of defining a finite field containing 2^nelements; such a field is referred to as GF(2^n). Consider the set, *S*, of all polynomials of degree *n* - 1 or less with binary coefficients. Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

Where each a_i takes on the value 0 or 1. There are a total of 2^ndifferent polynomials in *S*.
For *n* = 3, the 2^3 = 8 polynomials in the set are

$$\begin{array}{llll} 0 & x & x^2 & x^2 + x \\ 1 & x + 1 & x^2 + 1 & x^2 + x + 1 \end{array}$$

Appropriate definition of arithmetic operations, each such set *S* is a finite field.

The definition consists of the following elements.

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra with the following two refinements.
2. Arithmetic on the coefficients is performed modulo 2. This is the same as the XOR operation.
3. If multiplication results in a polynomial of degree greater than *n* - 1, then the n polynomial is reduced modulo some irreducible polynomial *m*(*x*) of degree *n*. That is, we divide by *m*(*x*) and keep the remainder. For a polynomial *f*(*x*), the remainder is expressed as *r*(*x*) = *f*(*x*) mod *m*(*x*). A polynomial *m*(*x*) is called **irreducible** if and only if *m*(*x*) cannot be expressed as a product of two polynomials, both of degree lower than that of *m*(*x*).

A polynomial in GF(2*n*) can be uniquely represented by its *n* binary coefficients(*an*-1*an*-2 c*a*0). Therefore, every polynomial in GF(2*n*) can be represented by an *n*-bit number.

2.12.2 AES Structure

General Structure

- Figure 2.20 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, orAES-256, depending on the key length.

- The input to the encryption and decryption algorithms is a single 128-bit block. The 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 2.21a. Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words.
- Below Figure 2.20 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. Note that the ordering of bytes within a matrix is by column. The first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix. The second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix. The 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 2.3).
- The first *N* - 1 round consist of four distinct transformation functions: Sub Bytes, Shift Rows, Mix Columns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is an 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 cipher text.

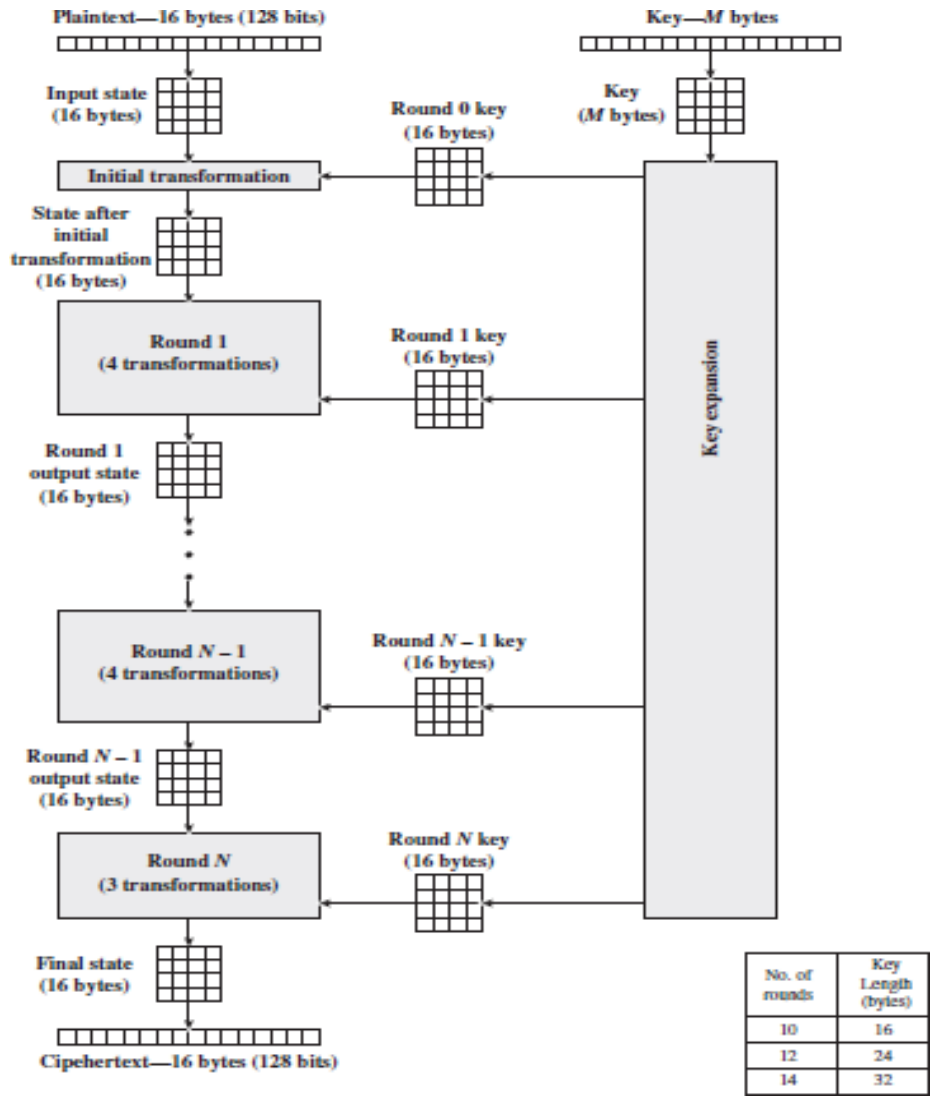


Figure 2.20 AES Encryption Process

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

Table 2.3 AES Parameters

2.12.3 Detailed Structure

Below Figure 2.20 shows the AES cipher shows the sequence of transformations in each round and showing the corresponding decryption function.

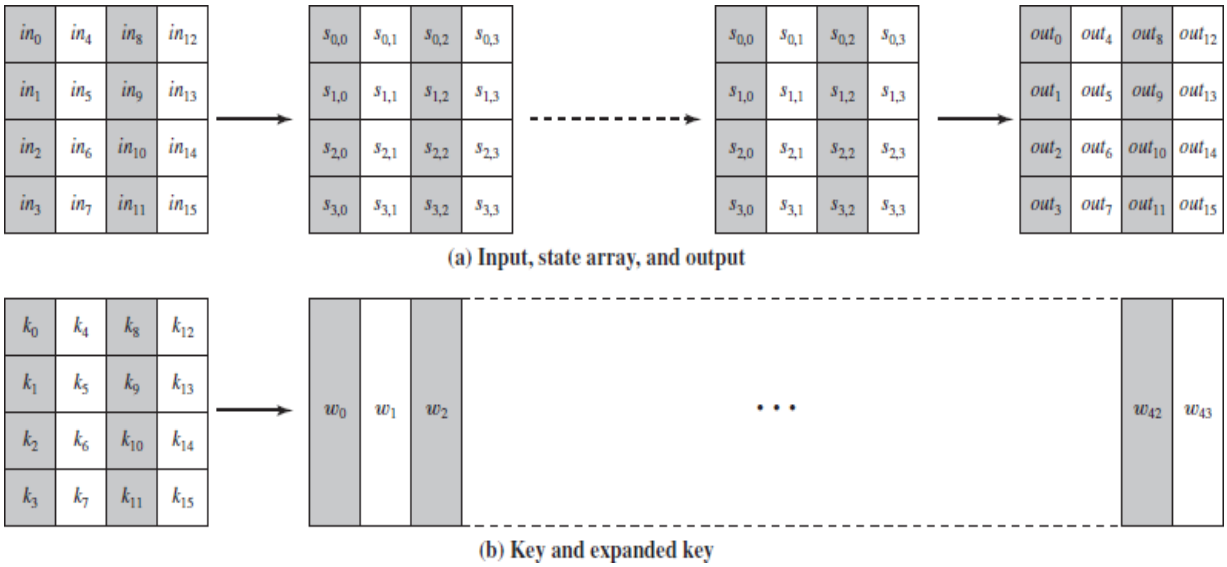


Fig: 2.21 Detail AES structure

Overall detail about AES structure.

1. 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 instead processes the entire data block as a single matrix during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bitwords, **w**[*i*]. Four distinct words (128 bits) serve as a round key for each round as shown in figure 2.22;
3. 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 ofthe block
   - **ShiftRows:** A simple permutation
   - **MixColumns:** A substitution that makes use of arithmetic over GF(28)
   - **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption as shown in figure 2.22, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.
5. Only the AddRoundKey stage makes use of the key. The AddRoundKey stage would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

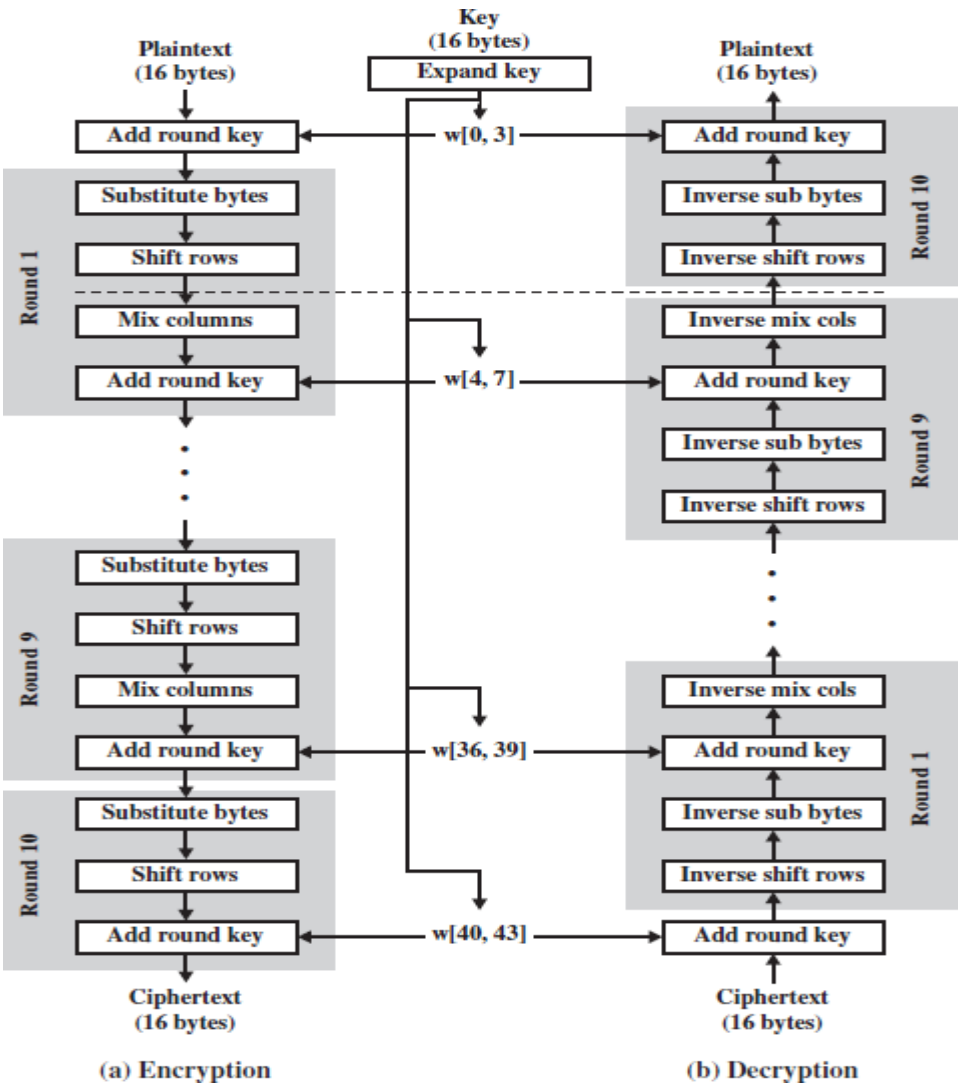


Fig 2.22 AES Encryption and Decryption

6. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that.

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

7. The decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

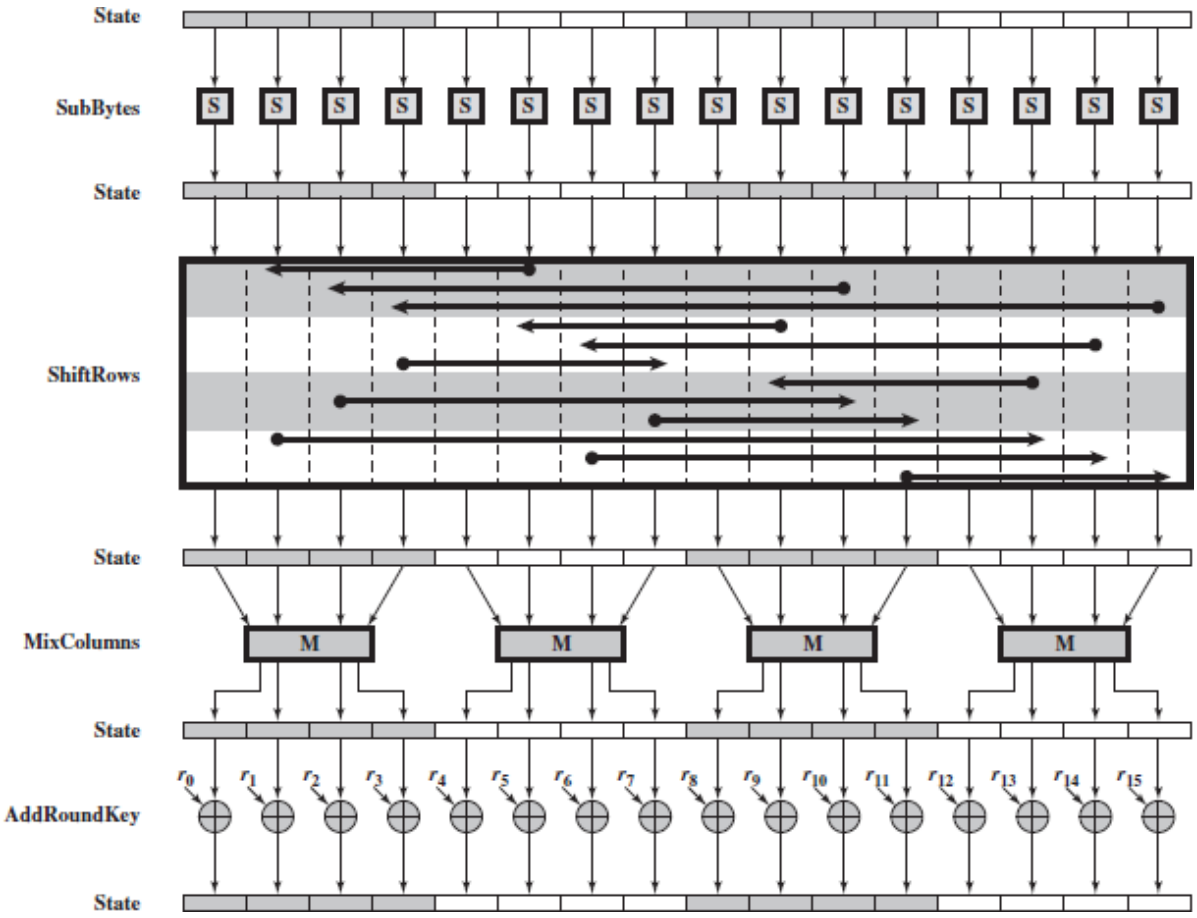


Fig 2.23 AES Encryption Round

8. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.
9. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required, to make the cipher reversible

2.12.4 AES Transformation Functions

Four transformations used in AES. For each stage, we describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage.

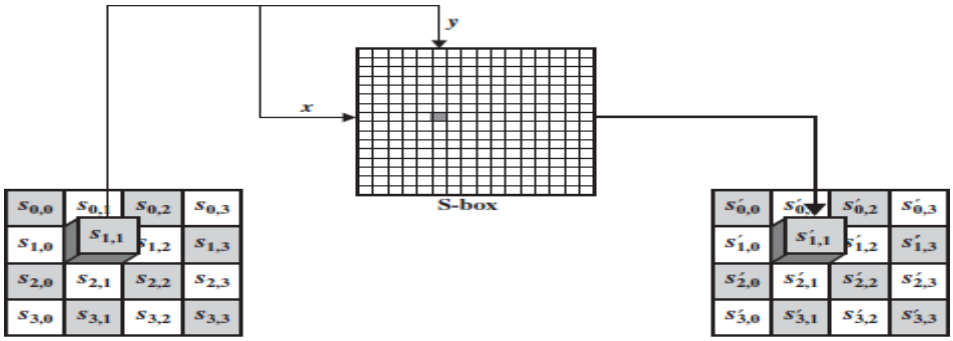
Substitute Bytes Transformation

Type 1: Forward and Inverse Transformations*:*

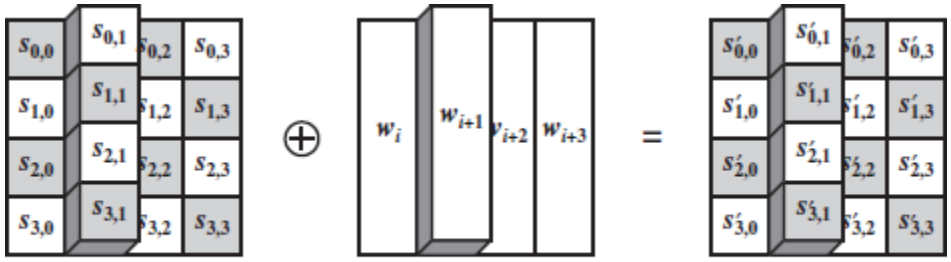
***The* forward substitute byte transformation**, called Sub Bytes, is a simple table lookup (Figure 2.24a). AES defines a 16 * 16 matrix of byte values, called an S-box that contains a permutation of all possible 256 8-bit values.

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 unique8-bit output value as shown in figure 2.25.

For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.



(a) Substitute byte transformation



(b) Add round key transformation

Figure 2.24 AES Byte level Operations

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

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

(b) Inverse S-box

Figure 2.25 AES S-Boxes

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

The S-box is constructed in the following fashion (Figure 2.26a).

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, c, {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row *y*, column *x* is {*yx*}.

2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(28); the value {00} is mapped to itself.

3. Consider that each byte in the S-box consists of 8 bits labeled (*b*7, *b*6, *b*5, *b*4, *b*3,*b*2, *b*1, *b*0). Apply the following transformation to each bit of each byte in the S-box:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Where *ci* is the *i*th bit of byte *c* with the value {63}; that is, ($c_7c_6c_5c_4c_3c_2c_1c_0$) = (01100011). The prime („) indicates that the variable is to be updated by the value on the right.

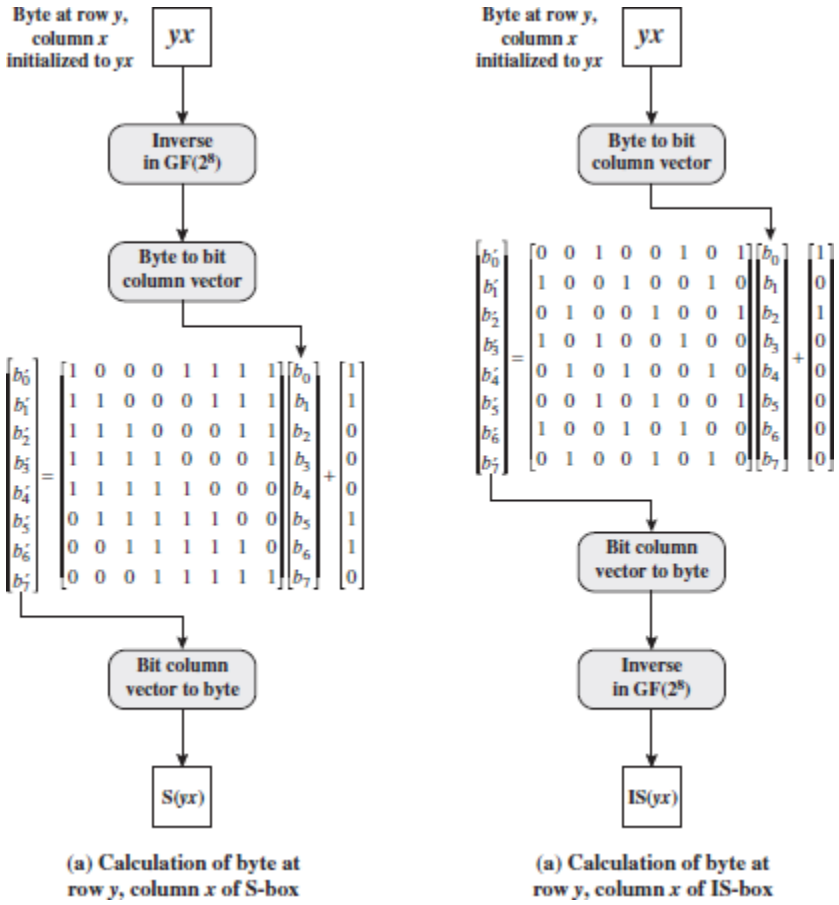


Figure 2.26 Construction of S-Box and IS-Box

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The AES standard depicts this transformation in matrix form as follows.

- In ordinary matrix multiplication, each element in the product matrix is the sum of products of the elements of one row and one column. Each element in the product matrix is the bitwise XOR of products of elements of one row and one column.
- As an example, consider the input value {95}. The multiplicative inverse in GF(28) is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary. Using above Equation

$$\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\1\\0\\1\\0\\0\\0\\1\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}=\begin{bmatrix}1\\0\\0\\1\\0\\0\\1\\0\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}=\begin{bmatrix}0\\1\\0\\1\\0\\1\\0\\0\end{bmatrix}$$

The result is {2A}, which should appear in row {09} column {05} of the S-box.

Type 2: Inverse Substitute Byte Transformation:

The **inverse substitute byte transformation**, called InvSubBytes, For example, that the input {2A}produces the output {95}, and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation is followed by taking the

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

multiplicative inverse in GF(28). The inverse transformation is

where byte *d* = {05}, or 00000101. We can depict this transformation as follows.

$$\begin{bmatrix}b'_0\\b'_1\\b'_2\\b'_3\\b'_4\\b'_5\\b'_6\\b'_7\end{bmatrix}=\begin{bmatrix}0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\\0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}+\begin{bmatrix}1\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$$

InvSubBytes is the inverse of Sub Bytes, label the matrices in sub Bytes and InvSubBytes as **X** and **Y**, respectively, and the vector versions of constants c and d as **C** and **D**, respectively. For some 8-bit vector **B,** becomes $\mathbf{B'} = \mathbf{XB} \oplus \mathbf{C}$. We need to show that $\mathbf{Y(XB} \oplus \mathbf{C)} \oplus \mathbf{D} = \mathbf{B}$**.** To multiply out, we must show$\mathbf{YXB} \oplus \mathbf{YC} \oplus \mathbf{D} = \mathbf{B}$**.** This becomes

$$\begin{bmatrix}0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\\0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}\oplus$$

$$\begin{bmatrix}0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\\0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0\end{bmatrix}\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}\oplus\begin{bmatrix}1\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}=$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

We have demonstrated that **YX** equals the identity matrix, and the **YC** = **D**,so that **YC** $\oplus$ **D** equals the null vector.

Type 3: Shift Rows Transformation

Forward and Inverse Shift RowsTransformations*:*

The forward shift row transformation, called Shift Rows, is depicted in Figure 2.27. 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-bytecircular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of Shift Rows

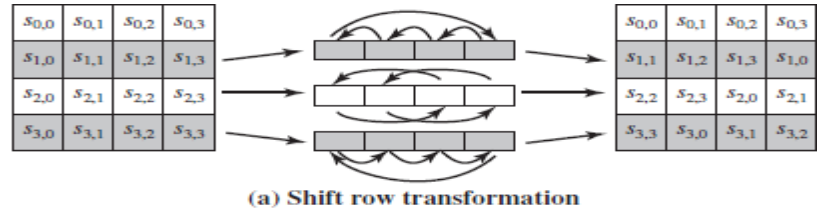
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

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

Figure 2.27 Forward Shift Row Transformation

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 as shown in figure 2.28



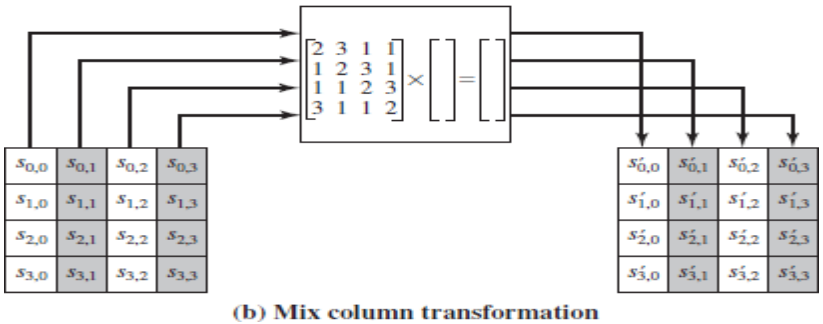


Figure 2.28 AES Row and Column Operations

Type 4: Mix Columns Transformation

Forward and Inverse Transformations*: 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. The transformation can be defined by the following matrix multiplication on **State**

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

Each element in the product matrix is the sum of products of elements of one rowand one column. In this case, the individual additions and multiplications are performed in GF(2^8).

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j} \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$
$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

The MixColumns transformation on a single column of **State** can be expressed as

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

The MixColumns transformation on the first column, we need to show that

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

For the first equation, we have {02}.{87} =(0000 1110) $\oplus$(0001 1011) =(0001 0101) and

{03}. {6E} = {6E}$\oplus$({02}. {6E}) = (0110 1110)$\oplus$(1101 1100) = (1011 0010) then

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

The **inverse mix column transformation**, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

The **inverse** of Equation need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of above Equation.

For the first equation, we have {0E}.{02} =00011100 and {09}.{03} ={09}$\oplus${09}.{02} = 00001001 $\oplus$ 00010010 =00011011then

$$\begin{aligned} \{0E\} \cdot \{02\} &= 00011100 \\ \{0B\} &= 00001011 \\ \{0D\} &= 00001101 \\ \{09\} \cdot \{03\} &= \underline{00011011} \\ & \quad 00000001 \end{aligned}$$

The encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes only encryption is used.

2. AES can be used to construct a message authentication code and for this, only encryption is used.

Type 5: AddRoundKey Transformation

Forward and Inverse Transformations

In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128bits of the round key.

The operation is viewed as a column wise operation between the 4 bytes of a **State** column and one word of the roundkey; it can also be viewed as a byte-level operation.

The following is an example ofAddRoundKey:

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

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

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

The Figure 2.29 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

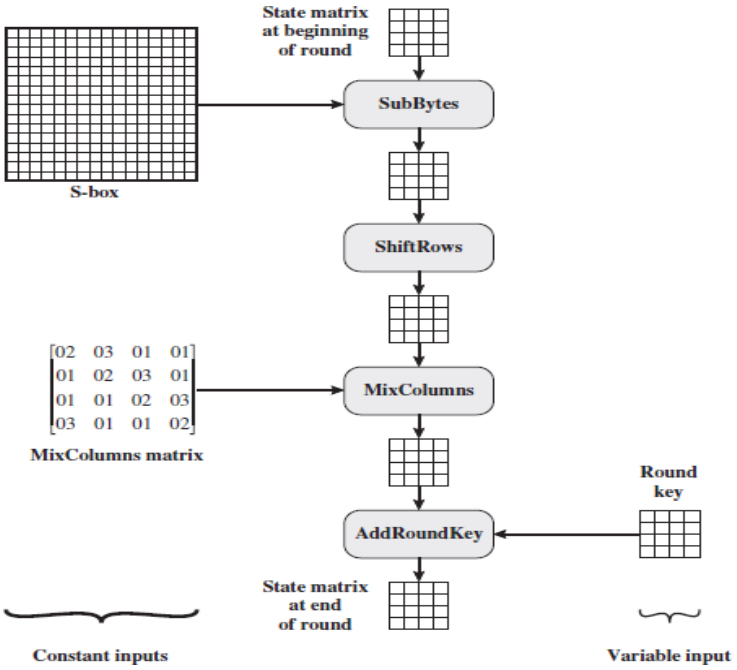


Fig 2.29 AES Key Expansion

Type 6: Key Expansion Algorithm

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.

Each added word **w**[i]depends on the immediately preceding word, **w**[i - 1], and the word four positions back, **w**[i - 4]. 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.

Figure 2.30 illustrates the generation of the expanded key, using the symbol g to represent that complex function. The function g consists of the following sub functions

```
KeyExpansion (byte key[16], word w[44])
{
     word temp
     for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                        key[4*i+2],
                                        key[4*i+3]);

      for (i = 4; i < 44; i++)
      {
       temp = w[i - 1];
       if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                                  ⊕ Rcon[i/4];
       w[i] = w[i-4] ⊕ temp
      }
}
```

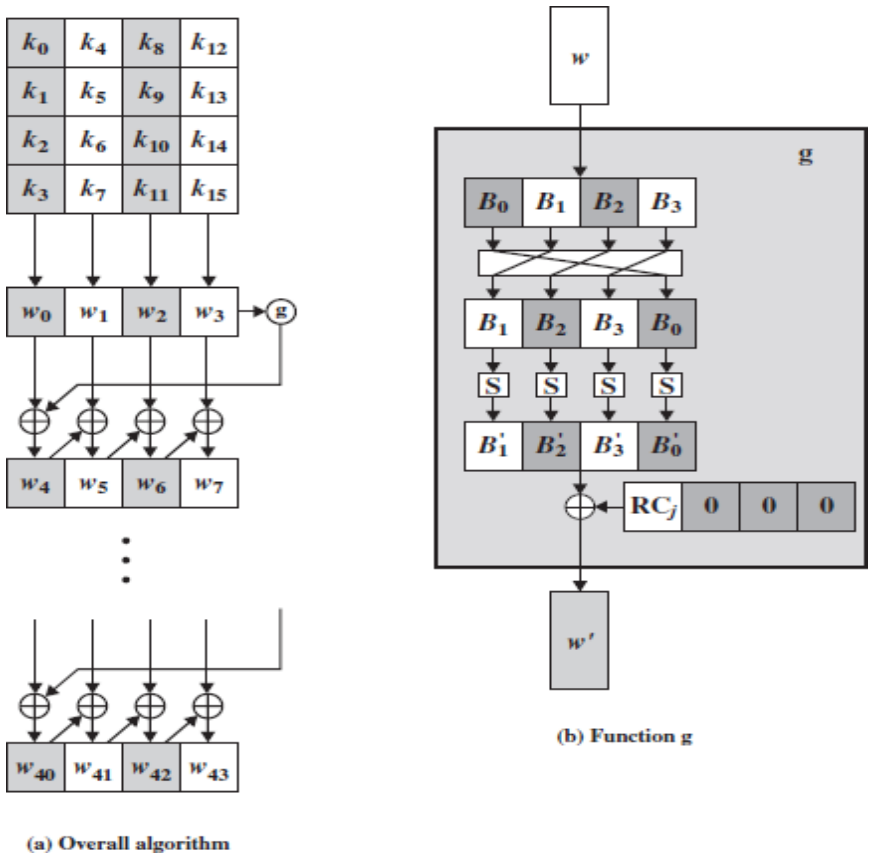


Figure 2.30 Key Expansion Algorithm

1. RotWord performs a one-byte circular left shift on a word. This means that a 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].

The round constant is a word in which the three rightmost bytes are always 0.Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 # RC[j-1] and with multiplication defined over the field GF(28). The values of RC[j] in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated asfollows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i−4]	w[i] = temp ⊕ w[i−4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

An AES Example

For this example, the plaintext is a hexadecimal palindrome. The plaintext,key, and resulting ciphertext are

Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	ff0b844a0853bf7c6934ab4364148fb9

Results

Table 2.4 shows the expansion of the 16-byte key into 10 round keys. The process is formed word by word, with each four-byte word occupying one column of the word round-key matrix.

Key Words	Auxiliary Function
w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad d6 w3 = af 7f 67 98	RotWord(w3) = 7f 67 98 af = x1 SubWord(x1) = d2 85 46 79 = y1 Rcon(1) = 01 00 00 00 y1 ⊕ Rcon(1) = d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord(w7) = 81 15 a7 38 = x2 SubWord(x2) = 0c 59 5c 07 = y2 Rcon(2) = 02 00 00 00 y2 ⊕ Rcon(2) = 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord(w11) = ff d3 c6 e6 = x3 SubWord(x3) = 16 66 b4 83 = y3 Rcon(3) = 04 00 00 00 y3 ⊕ Rcon(3) = 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0	RotWord(w15) = ae 7e c0 b1 = x4 SubWord(x4) = e4 f3 ba c8 = y4 Rcon(4) = 08 00 00 00 y4 ⊕ Rcon(4) = ec f3 ba c8 = 4

Table 2.4 Expansion of the 16-byte key into 10 round keys

The left-hand column shows the four round-key words generated for each round. The right-hand column shows the steps used to generate the auxiliary word used in key expansion. The key itself serving as the round key for round 0.

Next, Table 2.5 shows the progression of **State** through the AES encryption process. The first column shows the value of **State** at the start of a round. For the first row, **State** is just the matrix arrangement of the plaintext. The second, third, and fourth columns show the value of **State** for that round after the SubBytes, ShiftRows,andMixColumns transformations, respectively. The fifth column shows the roundkey.

Key Words	Auxiliary Function
w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord(w19) = 8c dd 50 43 = x5 SubWord(x5) = 64 c1 53 1a = y5 Rcon(5) = 10 00 00 00 y5 ⊕ Rcon(5) = 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord (w23) = 40 46 bd 4c = x6 SubWord (x6) = 09 5a 7a 29 = y6 Rcon(6) = 20 00 00 00 y6 ⊕ Rcon(6) = 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord (w27) = a5 a9 ef cf = x7 SubWord (x7) = 06 d3 bf 8a = y7 Rcon (7) = 40 00 00 00 y7 ⊕ Rcon(7) = 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord (w31) = 7d a1 4a f7 = x8 SubWord (x8) = ff 32 d6 68 = y8 Rcon (8) = 80 00 00 00 y8 ⊕ Rcon(8) = 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w32 = 3c be 0b 3	RotWord (w35) = be 0b 38 3c = x9 SubWord (x9) = ae 2b 07 eb = y9 Rcon (9) = 1B 00 00 00 y9 ⊕ Rcon (9) = b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord (w39) = 6b 41 56 f9 = x10 SubWord (x10) = 7f 83 b1 99 = y10 Rcon (10) = 36 00 00 00 y10 ⊕ Rcon (10) = 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	

Table 2.5 progression of State through the AES encryption process

2.13 RC4 ALGORITHM

RC4 is an encryption algorithm created in 1987 by Ronald Rivest of RSA Security. It is a stream cipher (figure 2.31), which means that each digit or character is encrypted one at a time. A cipher is a message that has been encoded.

A key input is pseudorandom bit generator that produces a stream 8-bit number that is unpredictable without knowledge of input key.

The output of the generator is called key-stream, is combined one byte at a time with the plaintext stream cipher using X-OR operation.

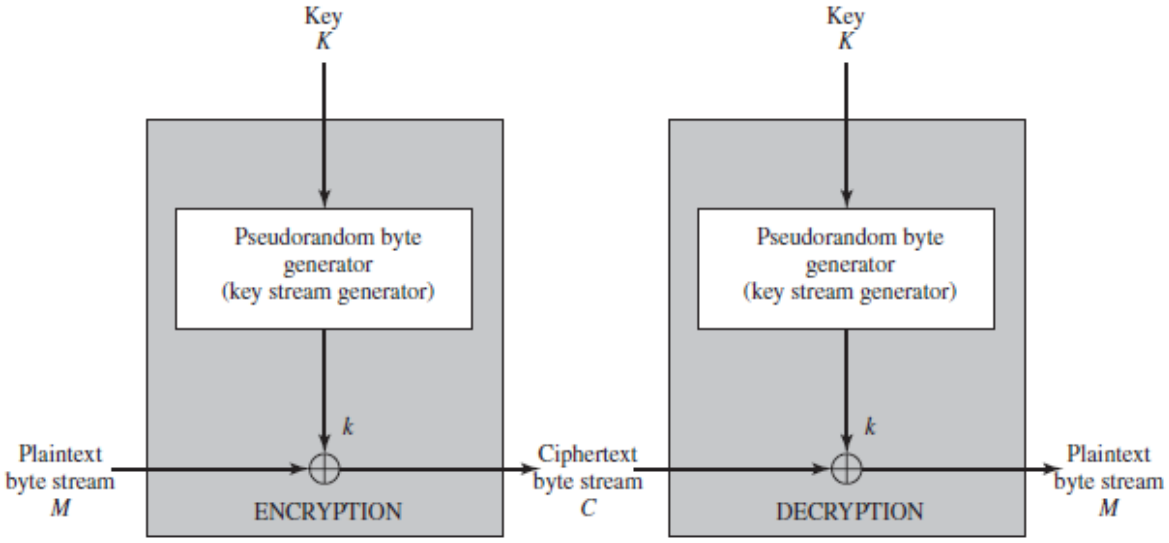


Figure 2.31 Stream Cipher Diagram

Example

RC4 Encryption			RC4 Decryption	
	10011000	Plaintext	11001000	Ciphertext
⊕	01010000	Key Stream	⊕ 01010000	Key Stream
	11001000	Ciphertext	10011000	Plaintext

2.13.1 Key Generation Algorithm

A variable-length key from 1 to 256 byte is used to initialize a 256-byte state vector S, with elements S[0] to S[255]. For encryption and decryption, a byte k is generated from S by selecting one of the 255 entries in a systematic fashion, then the entries in S are permuted again (Figure 2.32).

Initialization of S

The entries of S are set equal to the values from 0 to 255 in ascending orders, a temporary vector T, is created. If the length of the key k is 256 bytes, then k is assigned to T. Otherwise, for a key with length(klen) bytes, the first klen elements of T as copied from K and then K is repeated as many times as necessary to fill T.

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

Next, use T to produce the initial permutation of S. Starting with S[0] to S[255], and for each S[i] algorithm swap it with another byte in S according to a scheme dictated by T[i], but S will still contain values from 0 to 255:

```
// 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]);
 }
```

Pseudo random generation algorithm (Stream Generation)

Once the vector S is initialized, the input key will not be used. In this step, for each S[i] algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S. After reaching S[255] the process continues, starting from S[0] again

//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];

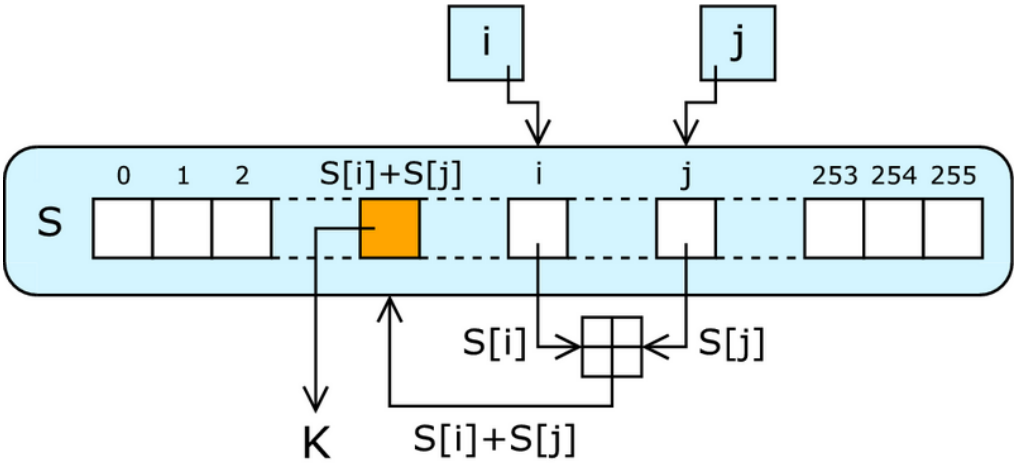


Figure 2.32 PRGA Algorithm

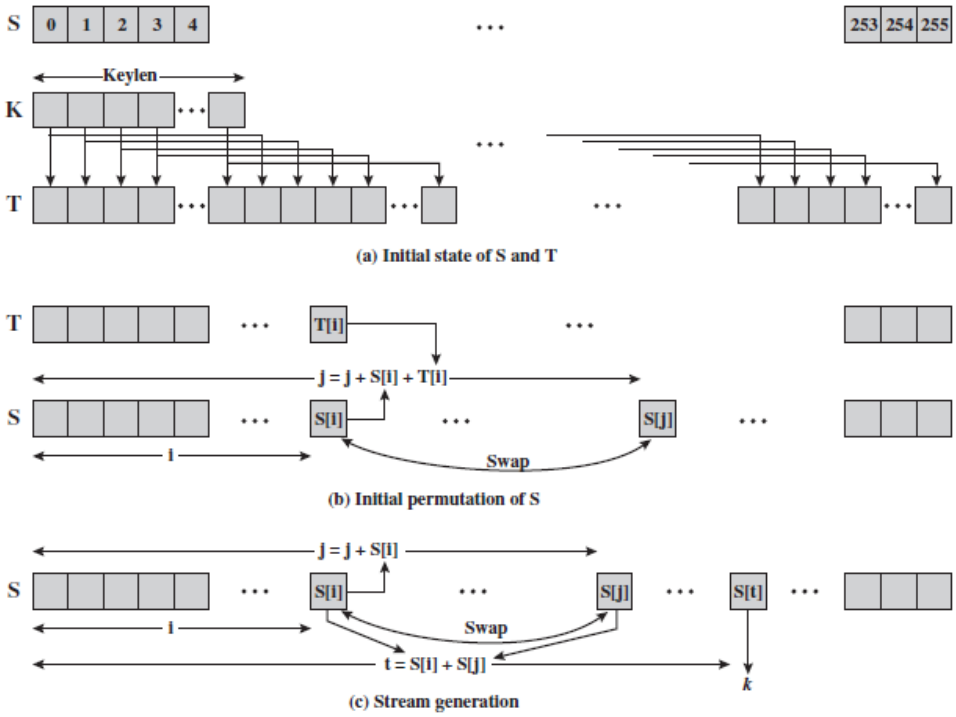


Figure 2.33 RC4 Algorithm

Encrypt using XOR

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

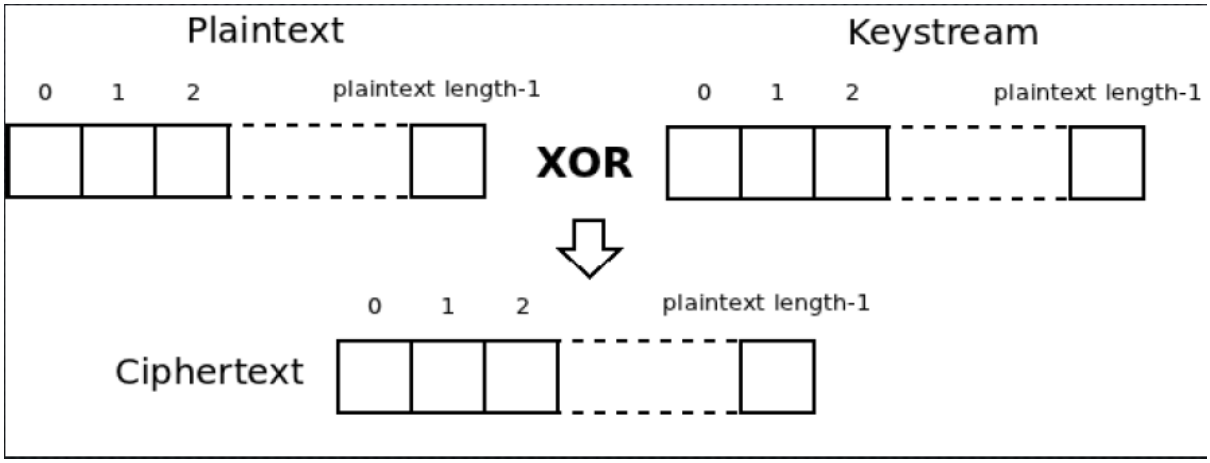


Figure 2.34 RC4 Encryption

Decrypt using XOR

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

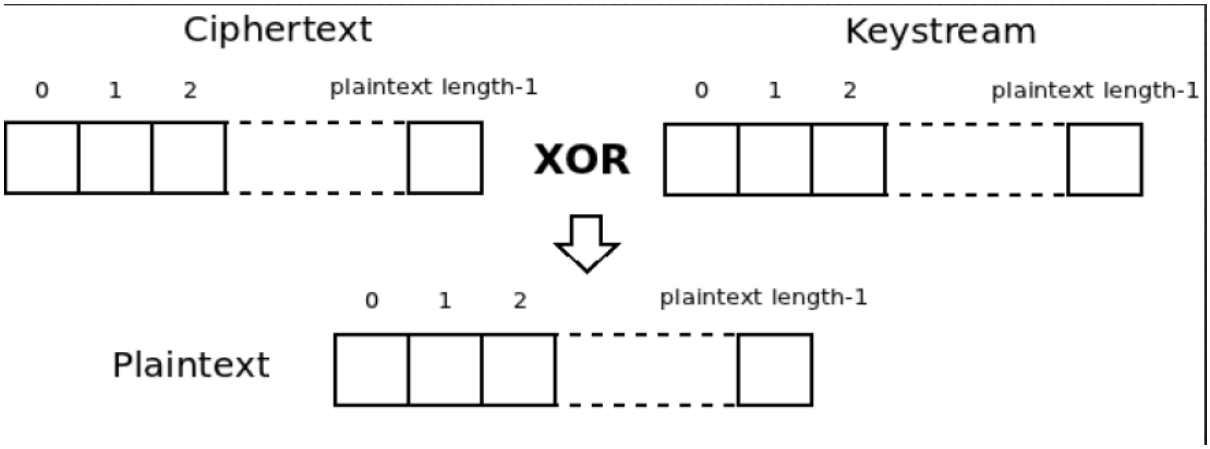


Figure 2.35 RC4 Decryption

Advantage

- It is faster and more suitable for streaming application

2.14 Key Distribution

2.14.1 Symmetric Key Distribution Using Symmetric Encryption

- In Symmetric key encryption, the two parties to an exchange must share the same key, and that key must be protected from access by others. Therefore, the term that refers to the means of delivering a key to two parties who wish to exchange data, without allowing others to see the key.
- For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can 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 can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third-party C, C can deliver a key on the encrypted links to A and B.

- Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 are mostly based on 1 or 2 occurring first.
- A third party, whom all parties trust, can be used as a trusted intermediary to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As numbers of parties grow, some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

2.14.2 Key Distribution Centre

- The use of a key distribution centre is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used.
- Communication between end systems is encrypted using a temporary key, often referred to as a Session key.
- Typically, the session key is used for the duration of a logical connection and then discarded
- Master key is shared by the key distribution centre and an end system or user and used to encrypt the session key.

2.14.3 Key Distribution Scenario

- Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, K_a, known only to itself and the KDC; similarly, B shares the master key K_b with the KDC (Figure 2.36). The following steps occur:

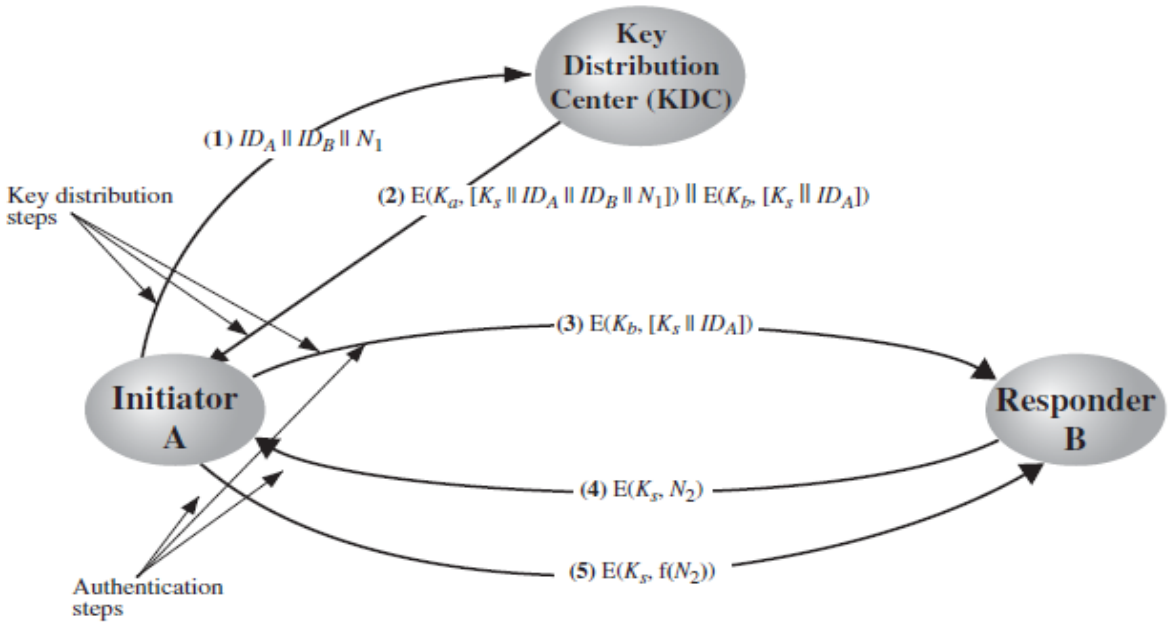


Figure 2.36 Key Distribution Scenarios

1. An issue a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1, for this transaction, which we refer to as a nonce. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using Ka Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
   - The one-time session key, Ks, to be used for the session
   - The original request message, including the nonce, to enable A to match this response with the appropriate request

   Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request. In addition, the message includes two items intended for B:
   - The one-time session key, Ks to be used for the session
   - An identifier of A (e.g., its network address), IDA

   These last two items are encrypted with Kb (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A store the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, E (K_b, [K_s || ID_A]). Because this information is encrypted with K_b, it is protected from eavesdropping. B now knows the session key (K_s), knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b). At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:
4. Using the newly minted session key for encryption, B sends a nonce, N_2, to A.
5. Also using K_s, A responds with f(N_2), where f is a function that performs some transformation on N_2 (e.g., adding one).

2.14.4 Session Key Lifetime

- The distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.
- For connection-oriented protocols, one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session.
- If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.
- For a connectionless protocol, such as a transaction-oriented protocol, there is no explicit connection initiation or termination.
- Thus, it is not obvious how often one needs to change the session key. The most secure approach is to use a new session key for each exchange.
- A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.