UNIT II SYMMETRIC CIPHERS

UNIT II SYMMETRIC CIPHERS

Number theory – Algebraic Structures – Modular Arithmetic - Euclid‘s algorithm – Congruence and matrices – Group, Rings, Fields, Finite Fields

SYMMETRIC KEY CIPHERS: SDES – Block Ciphers – DES, Strength of DES –

Differential and linear cryptanalysis – Block cipher design principles – Block cipher mode of operation –

Evaluation criteria for AES – Pseudorandom Number Generators – RC4 – Key distribution.

2.1 ALGEBRAIC STRUCTURES

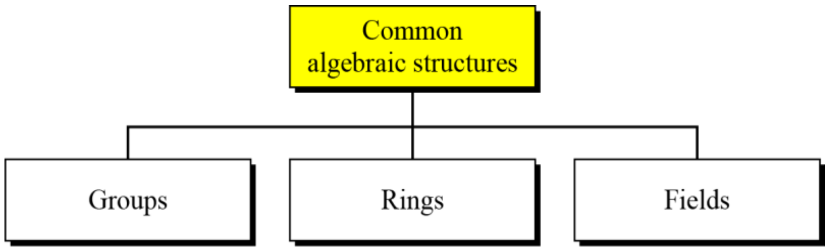


Figure 2.1 Common Algebraic Structures

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.

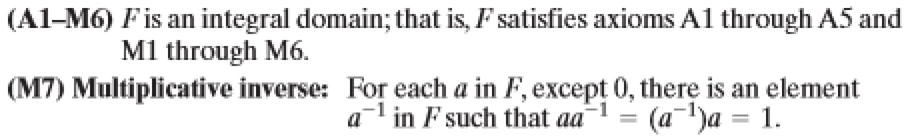
Next, we define an integral domain, which is a commutative ring that obeys the following axioms

(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 $b = 0$.

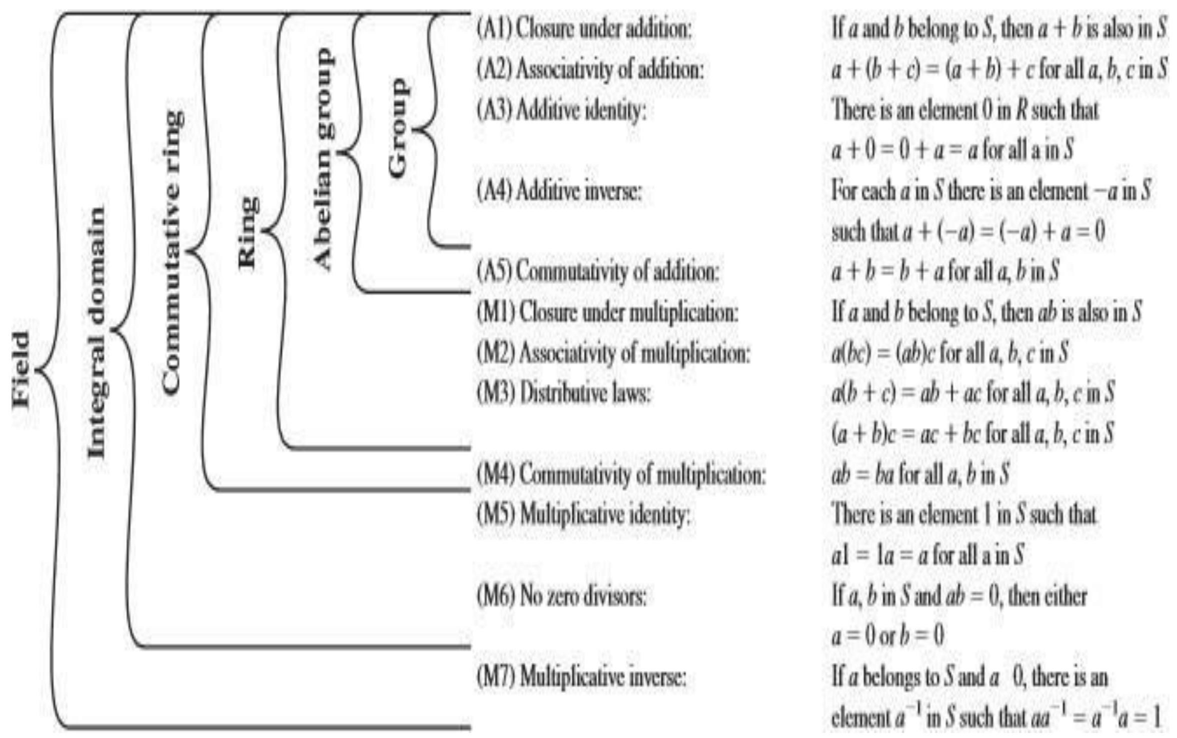
Fields

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 $aa^{-1} = (a^{-1})a = 1$.



(A1) Closure under addition:	If a and b belong to S, then $a + b$ is also in S
(A2) Associativity of addition:	$a + (b + c) = (a + b) + c$ for all a, b, c in S
(A3) Additive identity:	There is an element 0 in R such that $a + 0 = 0 + a = a$ for all a in S
(A4) Additive inverse:	For each a in S there is an element $-a$ in S such that $a + (-a) = (-a) + a = 0$
(A5) Commutativity of addition:	$a + b = b + a$ for all a, b in S
(M1) Closure under multiplication:	If a and b belong to S, then ab is also in S
(M2) Associativity of multiplication:	$a(bc) = (ab)c$ for all a, b, c in S
(M3) Distributive laws:	$a(b + c) = ab + ac$ for all a, b, c in S $(a + b)c = ac + bc$ for all a, b, c in S
(M4) Commutativity of multiplication:	$ab = ba$ for all a, b in S
(M5) Multiplicative identity:	There is an element 1 in S such that $a1 = 1a = a$ for all a in S
(M6) No zero divisors:	If a, b in S and $ab = 0$, then either $a = 0$ or $b = 0$
(M7) Multiplicative inverse:	If a belongs to S and a 0, there is an element a^{-1} in S 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

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

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

$$\boxed{11 \bmod 7 = 4; \qquad -11 \bmod 7 = 3}$$

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]

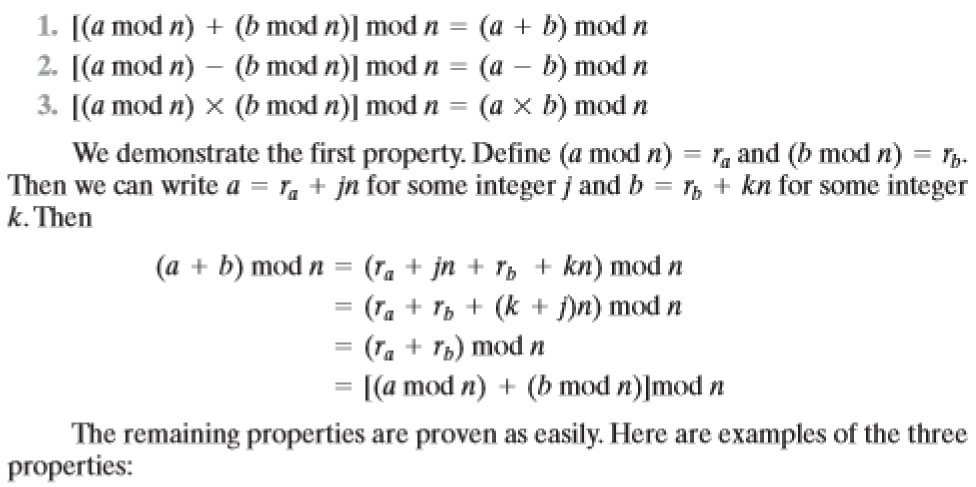
$$\boxed{73 \equiv 4 \pmod{23}; \qquad 21 \equiv -9 \pmod{10}}$$

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



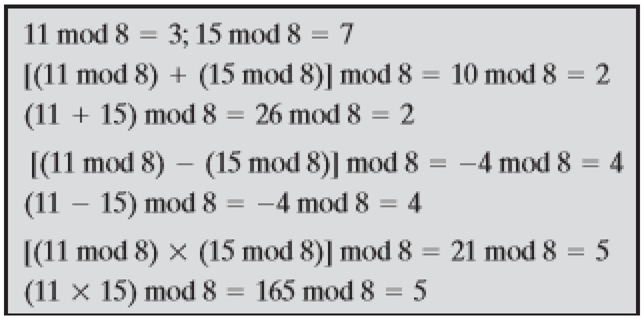
1. $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
2. $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
3. $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

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



$11 \bmod 8 = 3;\ 15 \bmod 8 = 7$

$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$

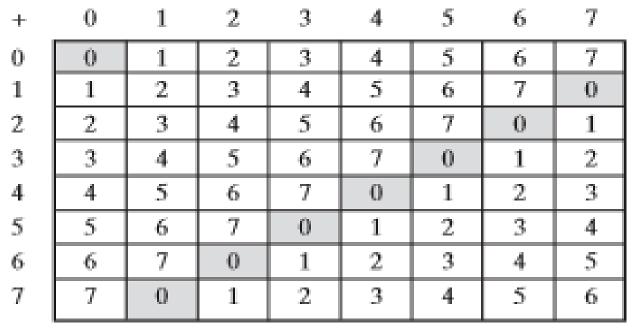
$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$

$[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = -4 \bmod 8 = 4$

$(11 - 15) \bmod 8 = -4 \bmod 8 = 4$

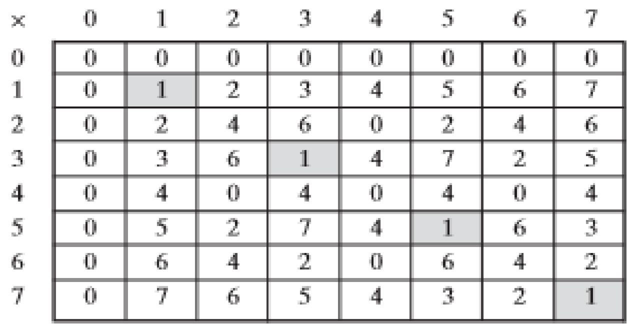
$[(11 \bmod 8) \times (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$

$(11 \times 15) \bmod 8 = 165 \bmod 8 = 5$



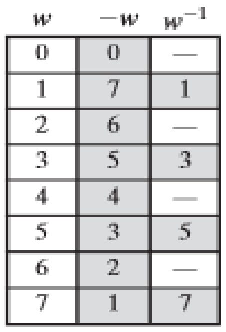
+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(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

Residue Classes

- Define the set Z_n as the set of nonnegative integers less than n: $Z_n = \{0, 1, \ldots\ldots\ldots\ldots (n - 1)\}$.

  This is referred to as the set of residues, or residue classes (mod n).
- To be more precise, each integer in Z_n represents a residue class.
- we can label the residue classes, (mod n) as [0], [1], [2], ... [n - 1], where

  $[r] = \{a: a$ is an integer, $a \equiv r \pmod{n}\}$

Residue Classes for mod 4

- Z4={0,1,2,3}

Residual Class [0] Residual Class [1] Residual Class [2]

- a mod n=r mod n

?

0 mod 4=0	1 mod 4=1	2 mod 4=2
4 mod 4=0	5 mod 4=1	6 mod 4=2
8 mod 4=0	9mod 4=1	10 mod 4=2
12 mod 4=0	13 mod 4=1	14 mod 4=2

The residue classes (mod 4) are

$$[0] = \{\ldots, -16, -12, -8, -4, 0, 4, 8, 12, 16, \ldots\}$$
$$[1] = \{\ldots, -15, -11, -7, -3, 1, 5, 9, 13, 17, \ldots\}$$
$$[2] = \{\ldots, -14, -10, -6, -2, 2, 6, 10, 14, 18, \ldots\}$$
$$[3] = \{\ldots, -13, -9, -5, -1, 3, 7, 11, 15, 19, \ldots\}$$

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)

- 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.
- More formally, the positive integer *c* is said to be the greatest common divisor of *a* and *b* if

1. *c* is a divisor of *a* and of *b*.

2. Any divisor of *a* and *b* is a divisor of *c*.

An equivalent definition is the following:

- gcd(a, b) = max[k, such that $k \mid a$ and $k \mid b$]

- we require that the greatest common divisor be positive, gcd(a, b) = gcd(a, -b) = gcd(-a, b) = gcd(a,b).
  - In general, gcd(a, b) = gcd($|a|$, $|b|$).

- We have integers a, b such that d = gcd (a, b).
- Now dividing a by b and applying the division algorithm, we can state:

$$\boldsymbol{a = q_1 b + r_1} \qquad \boldsymbol{0 \leq r_1 < b}$$

- If $r_1 = 0$, then $b \mid a$ and d = gcd(a, b) = b.
- But if $r_1 \neq 0$, we can divide b by r_1 i.e gcd(b, r_1) and apply the division algorithm to obtain

$$\boldsymbol{b = q_2 r_1 + r_2} \qquad \boldsymbol{0 \leq r_2 < r_1}$$

- if $r_2 \neq 0$, then d = gcd (r_1, r_2). The division process continues until some zero remainder appears
- a/b can be written as a=q b+r

a	dividend
b	divisor
q	quotient
r	remainder

- For e.g 7/4 can be written as 7=1 x 4+3

7	dividend
4	divisor
1	quotient
3	remainder

- We say, at the (n + 1)th stage where r_{n-1} is divided by r_n. The result is the following system of equations

$$\left.\begin{array}{ll}
a = q_1 b + r_1 & 0 < r_1 < b \\
b = q_2 r_1 + r_2 & 0 < r_2 < r_1 \\
r_1 = q_3 r_2 + r_3 & 0 < r_3 < r_2 \\
\quad\vdots & \quad\vdots \\
r_{n-2} = q_n r_{n-1} + r_n & 0 < r_n < r_{n-1} \\
r_{n-1} = q_{n+1} r_n + 0 & \\
d = \gcd(a, b) = r_n &
\end{array}\right\}$$

At each iteration, we have $d = \gcd(r_i, r_{i+1})$ until finally $d = \gcd(r_n, 0) = r_n$.

- **Example1:gcd(a,b)=gcd(24,12)**
  We have a=q1b+r1

  24=2x12+0

If r1=0 return b as result

Hence gcd(24,12) is 12

Example 2: 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 3:

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)

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 4:

Step 1: gcd(a,b)=gcd(1970,1066)

We have a=q1b+r1

1970=1x1066+ 904

Here r1≠0

step :2 so compute gcd(b,r1)=gcd(1066,904)

We have b=q2r1+r2

1066=1x904+162

Here r2 ≠0

Step :3 So compute gcd(r1,r2)=gcd(904,162)

We have r1=q3r2+r3904=5x162+94

Here r3 ≠0

Repeat the steps until remainder is 0.if remainder is 0 return previous remainder as result.

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)

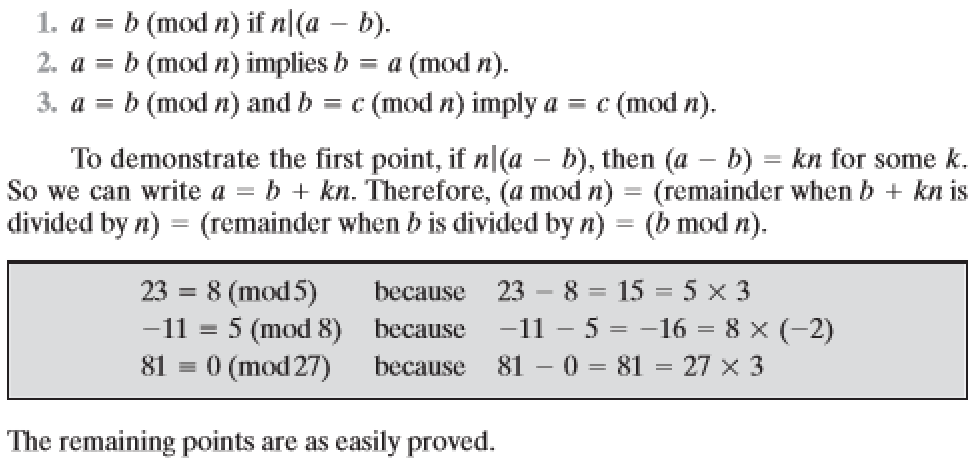
904	= 5 x 162 + 94	gcd (162, 94)
162	= 1 x 94 + 68	gcd (94, 68)
94	= 1 x 68 + 26	gcd (68, 26)
68	= 2 x 26 + 16	gcd (26, 16)
26	= 1 x 16 + 10	gcd (16, 10)
16	= 1 x 10 + 6	gcd (10, 6)
10	= 1 x 6 + 4	gcd (6, 4)
6	= 1 x 4 + 2	gcd (4, 2)
4	= 2 x 2 + 0	gcd (2, 0)

Therefore, gcd (1970, 1066) = 2

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 $a = c \pmod{n}$.

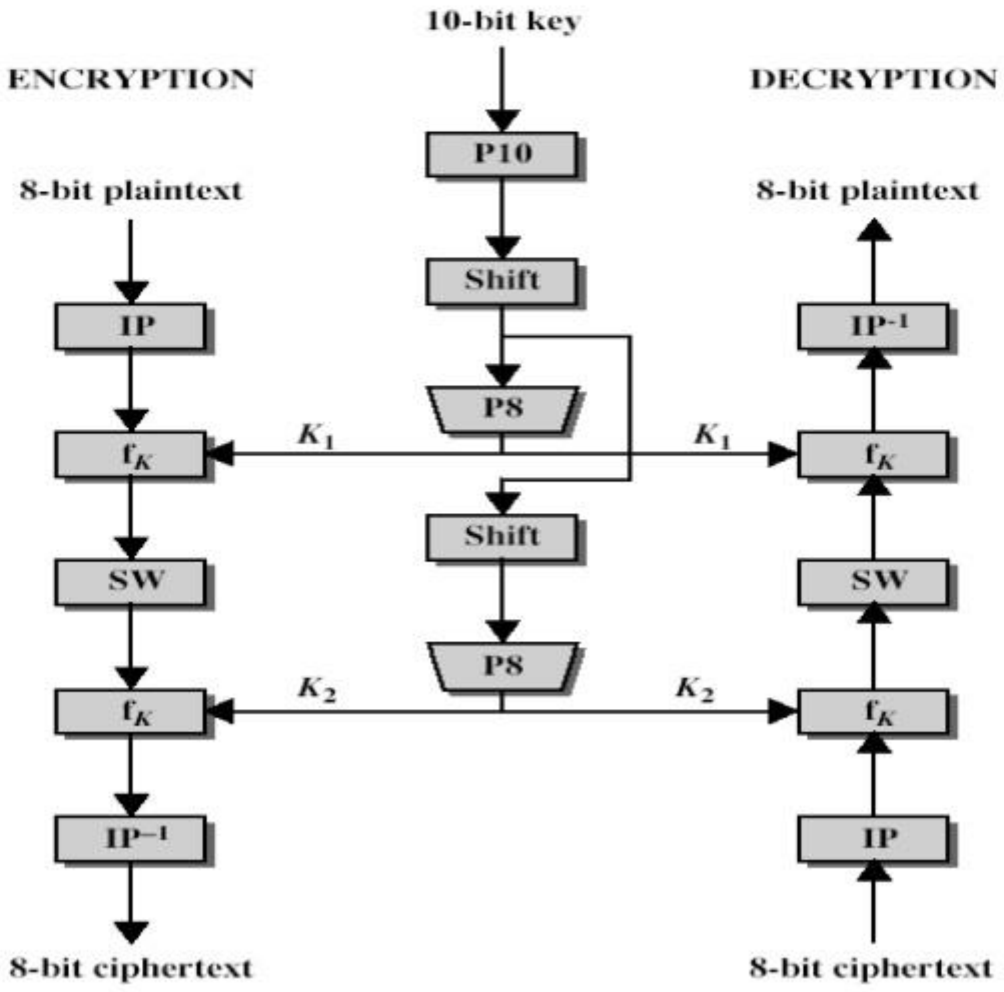
To demonstrate the first point, if $n|(a - b)$, then $(a - b) = kn$ for some k. So we can write $a = b + kn$. Therefore, $(a \bmod n)$ = (remainder when $b + kn$ is divided by n) = (remainder when b is divided by n) = $(b \bmod n)$.

$23 = 8 \pmod{5}$	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.

Simplified Data Encryption Standard (S-DES)

The overall structure of the simplified DES. 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.



The encryption algorithm involves five functions:

1. An initial permutation (IP)

2. A complex function labelled f_k, which involves both permutation and substitution operations and depends on a key input

3. A simple permutation function that switches (SW) the two halves of the data

4. The function f_k again

5. A permutation function that is the inverse of the initial permutation

The function f_k 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 of functions: $IP^{-1} \text{ o } f_{K2} \text{ o } SW \text{ o } f_{k1} \text{ o } IP$

Which can also be written as

$$\text{Ciphertext} = IP^{-1} (f_{K2} (SW (f_{k1} (IP (\text{plaintext})))))$$

Where

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

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

Decryption can be shown as

$$\text{Plaintext} = IP^{-1} (f_{K1} (SW (f_{k2} (IP (\text{ciphertext})))))$$

S-DES key generation

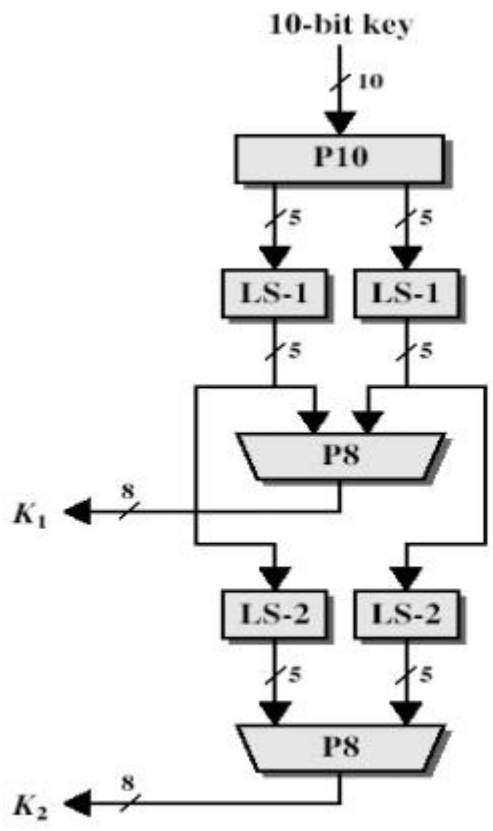


Figure: key generation for S-DES

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. 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. For example, the key (1010000010) is permuted to (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 we 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

The result is subkey 1 (K1). In our example, this yields (10100100). We 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 S-DES encryption

Encryption involves the sequential application of five functions.

Initial and Final 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

This retains all 8 bits of the plaintext but mixes them up.

Consider the plaintext to be 11110011.

Permuted output = 10111101

At the end of the algorithm, the inverse permutation is used:

IP –1							
4	1	3	5	7	2	8	6

The Function f_k

The most complex component of S-DES is the function f_k, 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

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

Where SK is a subkey and $\oplus$ is the bit-by-bit exclusive-OR function.

e.g., permuted output = 1011 1101 and suppose F (1101, SK) = (1110) for some key SK.

Then f K(10111101) = 1011$\oplus$1110, 1101

= 01011101

We 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

e.g., R= 1101

E/P output = 11101011

It is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey K1 = (k11, k12 , k13 , k14 , k15, k16 , k17 , k18) is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits: $\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$

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. The entry in that row and column, in base 2, is the 2-bit output. For example, if (p0,0 p0,3) =) (00) and (p0,1 p0,2) = (10), then the output is from row 0, column 2 of S0, which is 3, or (11) in) binary. Similarly, (p1,0 p1,3) and (p1,1 p1,2) are used to index into a row and column of S1 to produce an additional 2 bits. Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

The Switch Function

The function f K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext.

DATA ENCRYPTION STANDARD

DES Encryption

As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length. Actually, the function expects a 64-bit key as input. However, only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily.

General Depiction of DES Encryption Algorithm

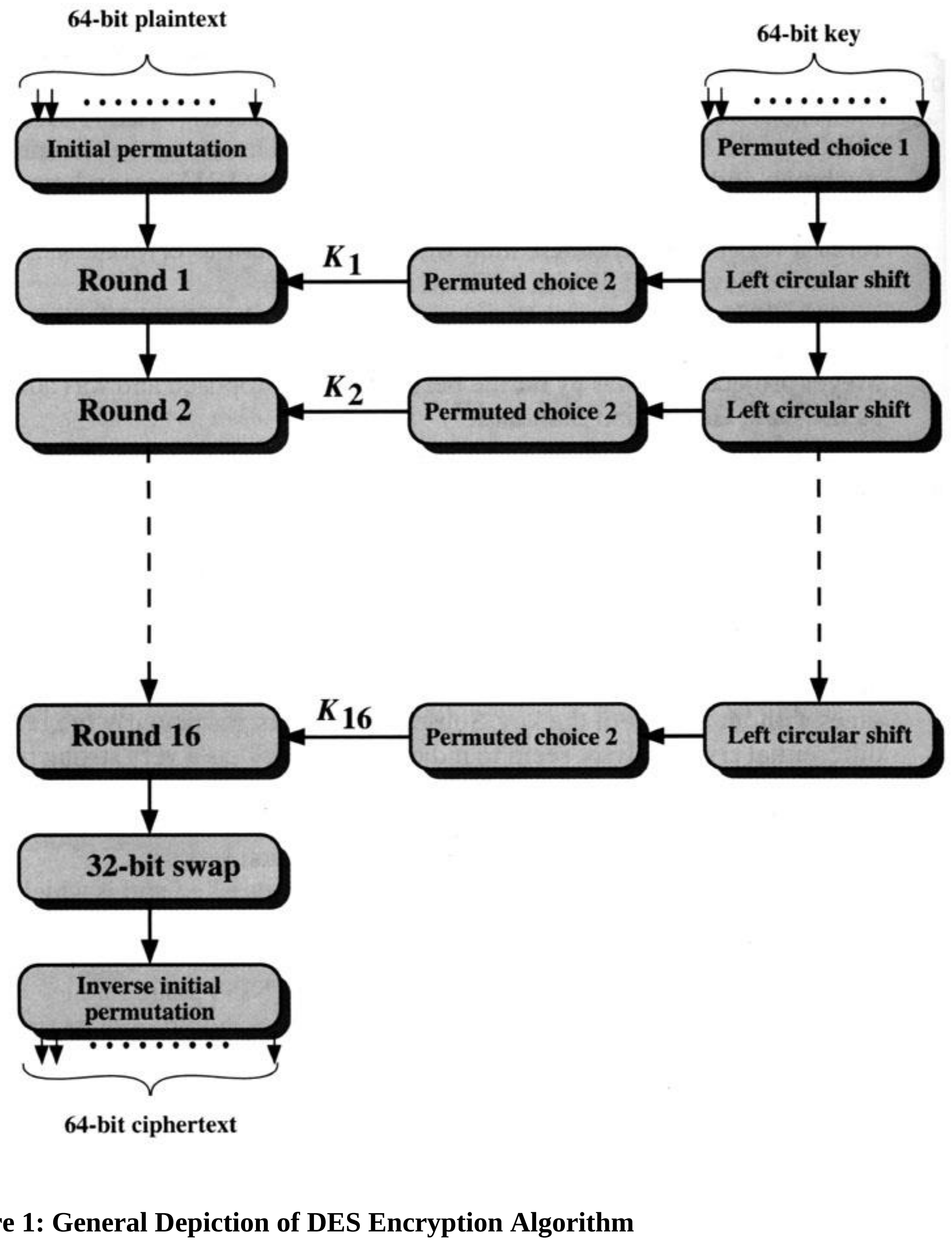


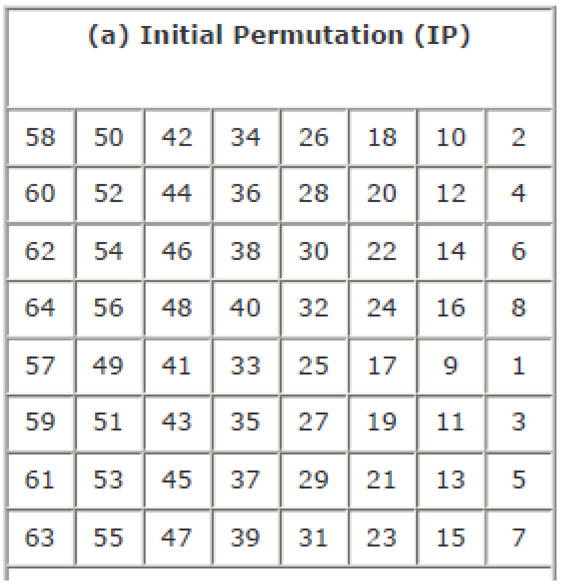
Figure 1: General Depiction of DES Encryption Algorithm

Looking at the left-hand side of the figure, 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*. 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**. 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. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

The right-hand portion of Figure 1 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 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.

Initial Permutation

The initial permutation and its inverse are defined by tables, as shown in Tables a and b respectively. The tables are to be interpreted as follows. 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.



(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

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

(b)Inverse Initial permutation (IP^{-1})

(c) 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	

(d) 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

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input *M*:

$$\begin{array}{cccccccc}
M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 & M_8 \\
M_9 & M_{10} & M_{11} & M_{12} & M_{13} & M_{14} & M_{15} & M_{16} \\
M_{17} & M_{18} & M_{19} & M_{20} & M_{21} & M_{22} & M_{23} & M_{24} \\
M_{25} & M_{26} & M_{27} & M_{28} & M_{29} & M_{30} & M_{31} & M_{32} \\
M_{33} & M_{34} & M_{35} & M_{36} & M_{37} & M_{38} & M_{39} & M_{40} \\
M_{41} & M_{42} & M_{43} & M_{44} & M_{45} & M_{46} & M_{47} & M_{48} \\
M_{49} & M_{50} & M_{51} & M_{52} & M_{53} & M_{54} & M_{55} & M_{56} \\
M_{57} & M_{58} & M_{59} & M_{60} & M_{61} & M_{62} & M_{63} & M_{64}
\end{array}$$

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

$$\begin{array}{cccccccc}
M_{58} & M_{50} & M_{42} & M_{34} & M_{26} & M_{18} & M_{10} & M_2 \\
M_{60} & M_{52} & M_{44} & M_{36} & M_{28} & M_{20} & M_{12} & M_4 \\
M_{62} & M_{54} & M_{46} & M_{38} & M_{30} & M_{22} & M_{14} & M_6 \\
M_{64} & M_{56} & M_{48} & M_{40} & M_{32} & M_{24} & M_{16} & M_8 \\
M_{57} & M_{49} & M_{41} & M_{33} & M_{25} & M_{17} & M_9 & M_1 \\
M_{59} & M_{51} & M_{43} & M_{35} & M_{27} & M_{19} & M_{11} & M_3 \\
M_{61} & M_{53} & M_{45} & M_{37} & M_{29} & M_{21} & M_{13} & M_5 \\
M_{63} & M_{55} & M_{47} & M_{39} & M_{31} & M_{23} & M_{15} & M_7
\end{array}$$

If we then take the inverse permutation *Y* = IP-1(*X*) = IP-1(IP(*M*)), it can be seen that the original ordering of the bits is restored.

Details of Single Round

Figure 2 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

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

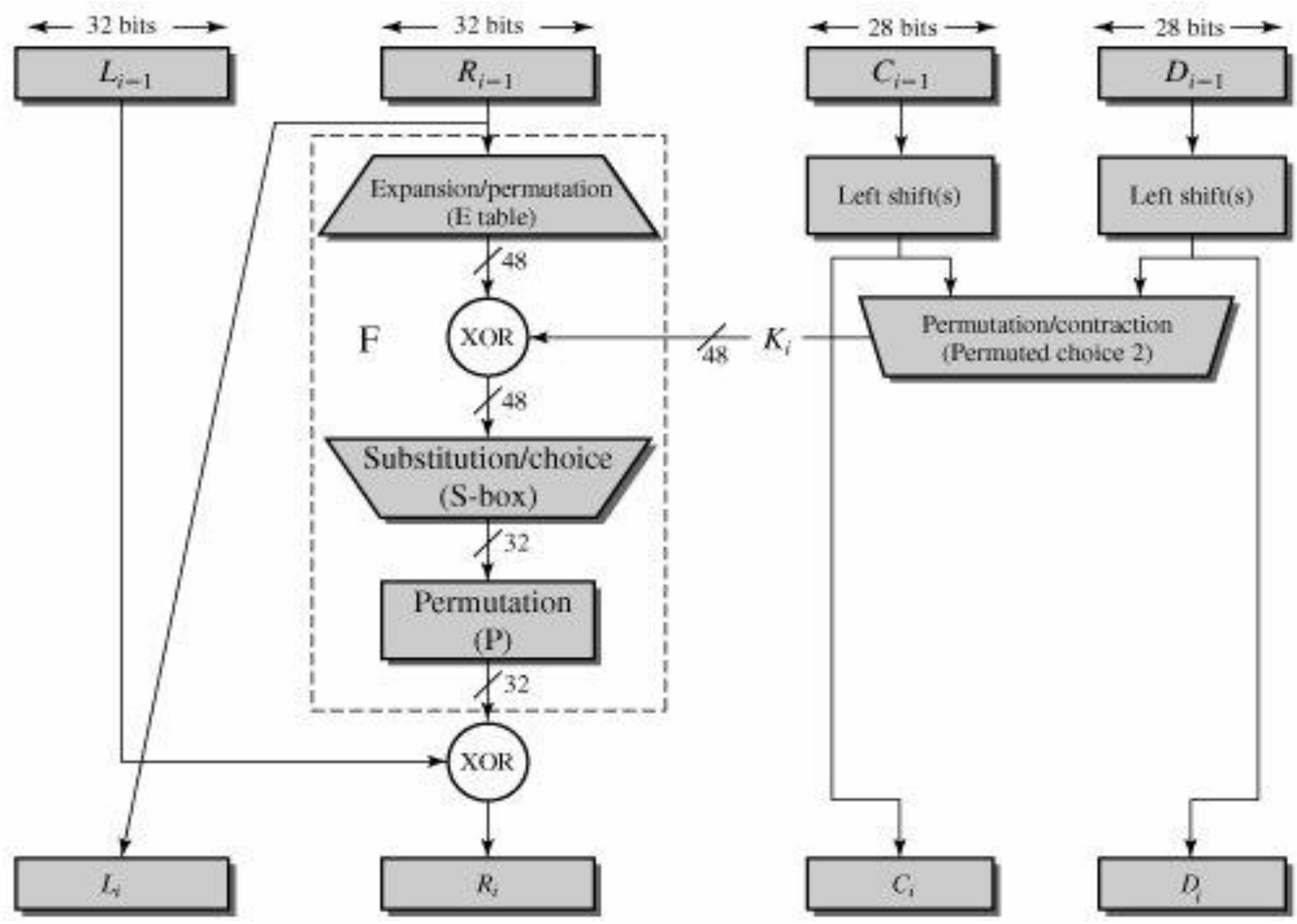


Figure 2. Single Round of DES Algorithm

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 (Table c). 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 permuted as defined by Table d. The role of the S-boxes in the function F is illustrated in Figure 3.

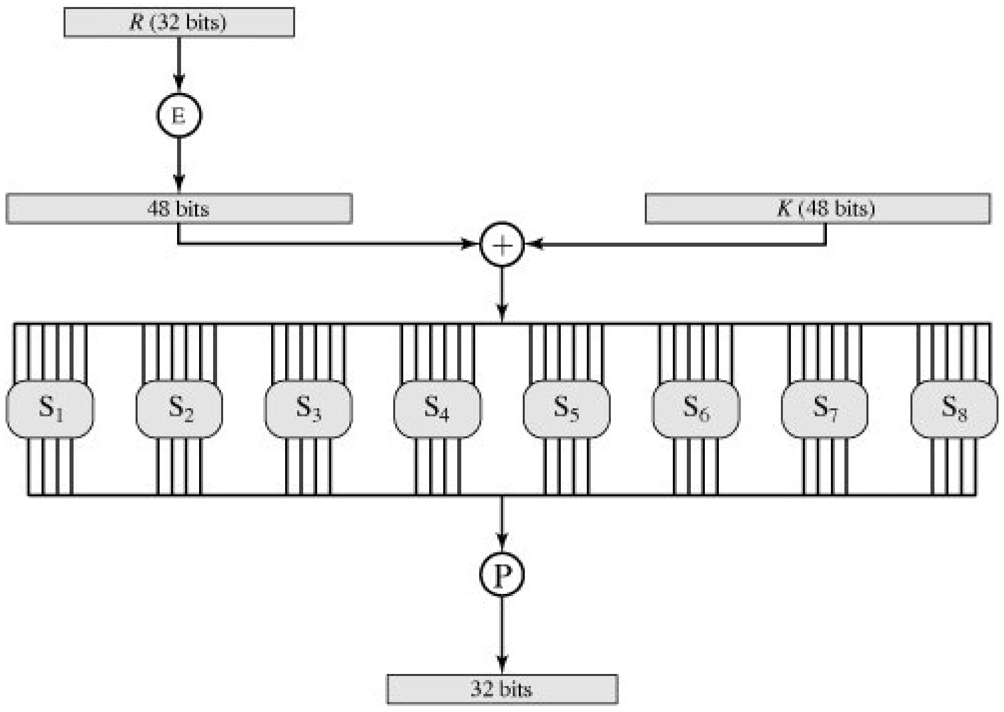


Figure 3: Calculation of F(R, K)

The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: 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. 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.

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3.3. Definition of DES S-Boxes

Each row of an S-box defines a general reversible substitution. Figure 3.1 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S1.The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (*Ki*). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

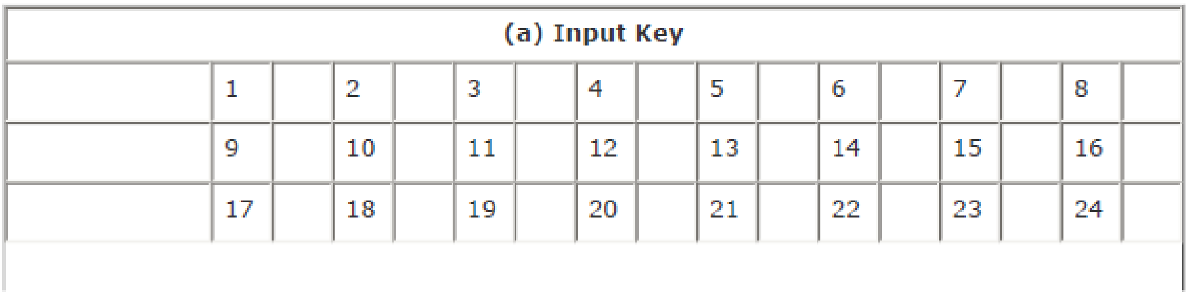
... efgh ijkl mnop ...

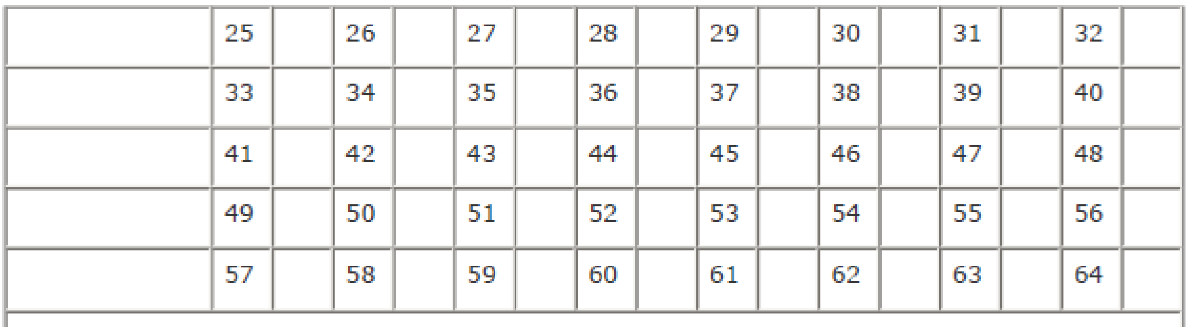
this becomes

... defghi hijklm lmnopq ...

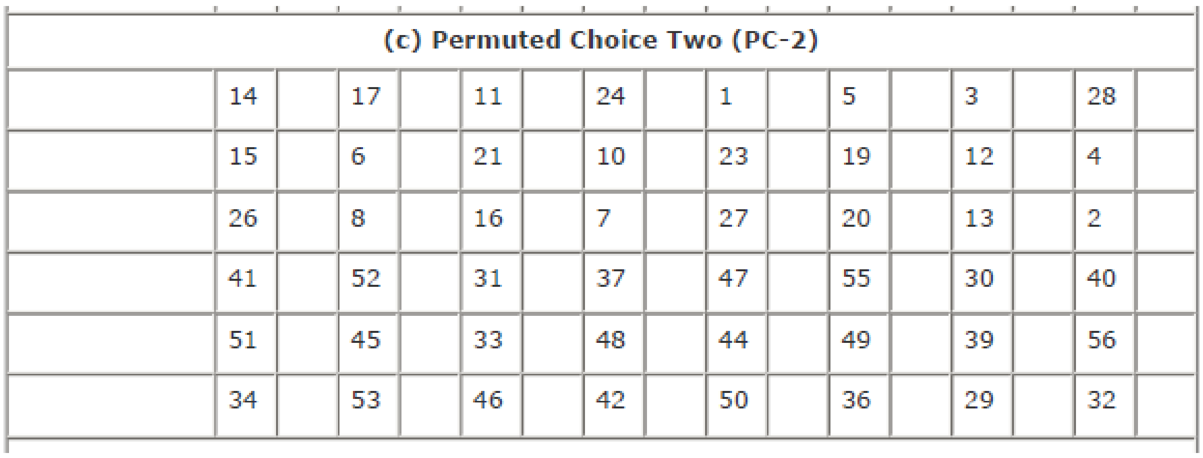
The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round the output from each S-box immediately affects as many others as possible.

Key Generation

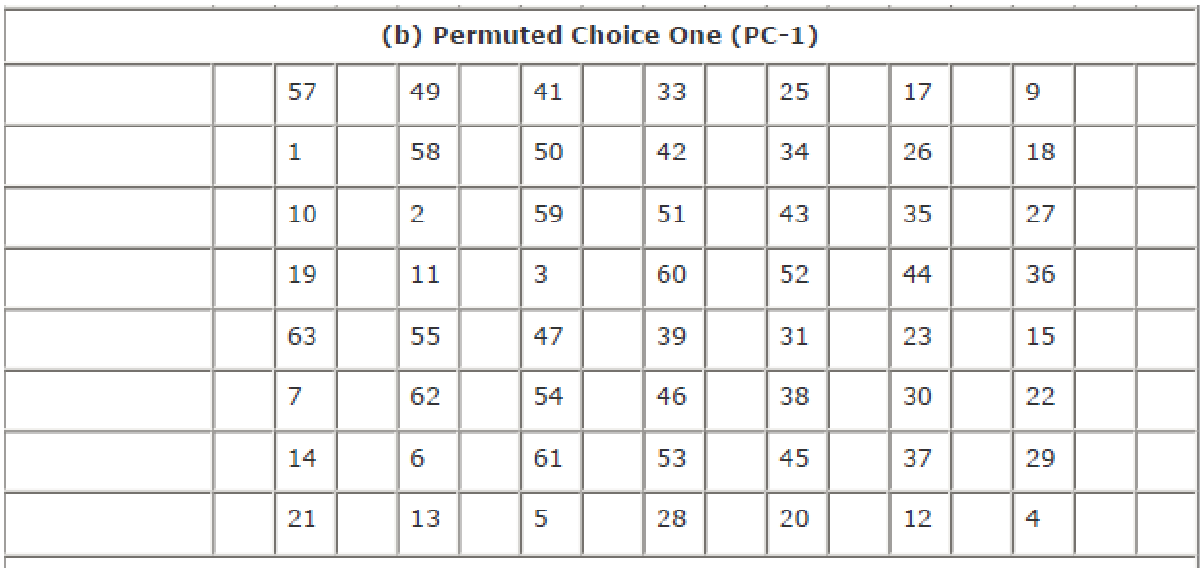




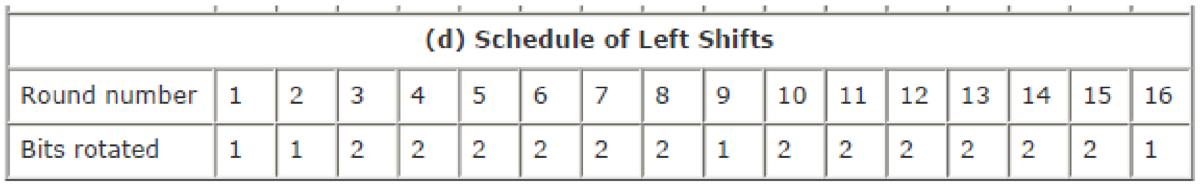
(a) Input Key																
	1		2		3		4		5		6		7		8	
	9		10		11		12		13		14		15		16	
	17		18		19		20		21		22		23		24	
	25		26		27		28		29		30		31		32	
	33		34		35		36		37		38		39		40	
	41		42		43		44		45		46		47		48	
	49		50		51		52		53		54		55		56	
	57		58		59		60		61		62		63		64	



(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	



(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		



(d) Schedule of Left Shifts																
Round number	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

DES Decryption

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

THE STRENGTH OF DES

The Use of 56-Bit Keys

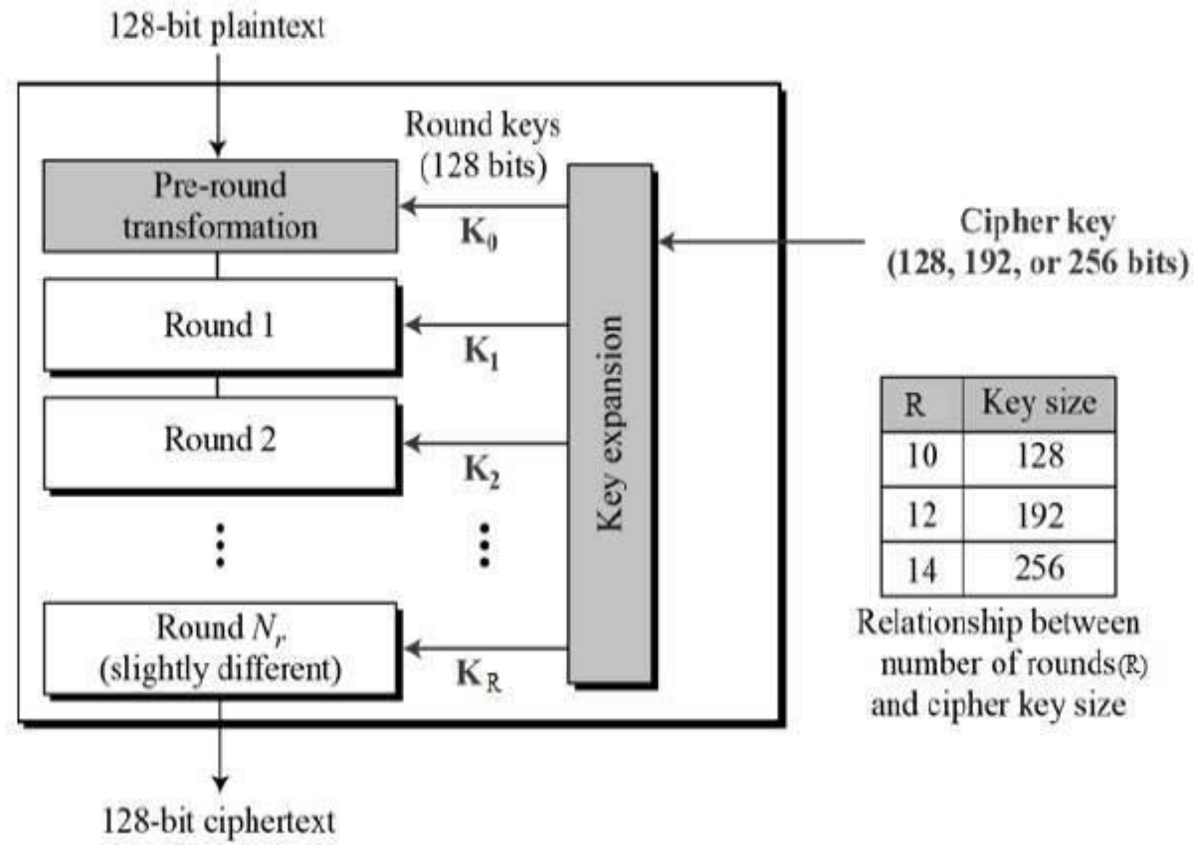
With a key length of 56 bits, there are 256 possible keys, which is approximately 7.2 X 1016 keys. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher. DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation

(EFF) announced that it had broken a DES encryption using a special-purpose “DES cracker” machine that was built for less than $250,000.

ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

The Advanced Encryption Standard (AES), also known by its original name Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.The Advanced Encryption Standard, or AES, is a symmetric block cipher chosen by the U.S. government to protect classified information and is implemented in software and hardware throughout the world to encrypt sensitive data.

- The more popular symmetric encryption algorithm is the Advanced Encryption Standard (AES).
- It is found at least six time faster than triple DES.
- A replacement for DES was needed as its key size was too small.
- With increasing computing power, it was considered vulnerable against exhaustive key search attack.
- Triple DES was designed to overcome this drawback but it was found slow.



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

- AES is an iterative rather than Feistel cipher.
- It is based on ‘substitution–permutation network’.
- AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix
- Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys.
- Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

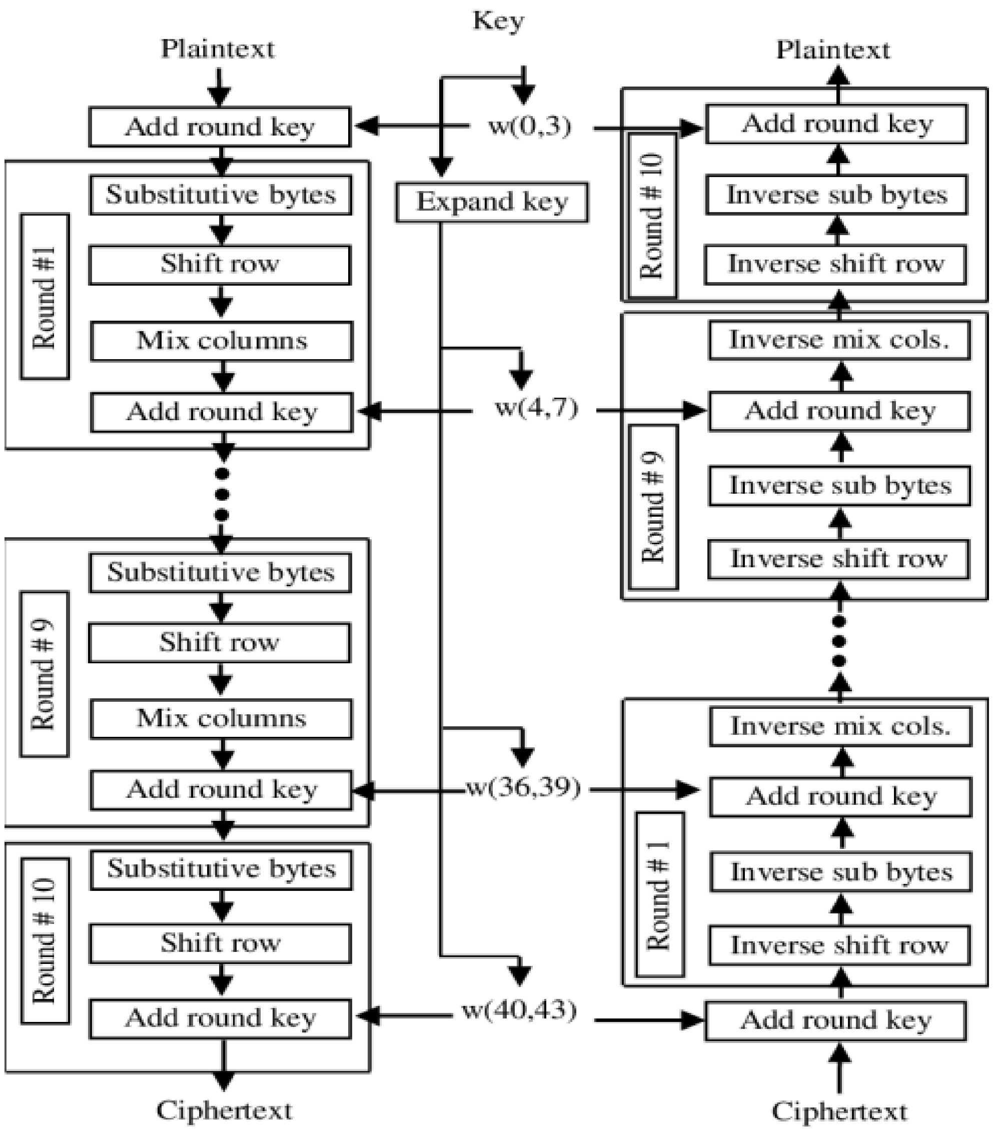


Fig 1.1 Basic structure of AES algorithm

Steps in the AES Encryption Process

The encryption process uses a set of specially derived keys called round keys. The AES steps of encryption for a 128-bit block:

1. Derive the set of round keys from the cipher key.
2. Initialize the state array with the block data (plaintext).
3. Add the initial round key to the starting state array.
4. Perform nine rounds of state manipulation.
5. Perform the tenth and final round of state manipulation.

6. Copy the final state array out as the encrypted data (ciphertext).

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES. A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution– permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations). Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix − Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128- bit round key, which is calculated from the original AES key.

Encryption Process

Encryption works by taking plain text and converting it into cipher text, which is made up of seemingly random characters. Only those who have the special key can decrypt it. AES uses symmetric key encryption, which involves the use of only one secret key to cipher and decipher information. Here, we restrict to description of a typical round of AES encryption. Each round comprises of four sub-processes.

- ByteSubstitution (SubBytes)
- ShiftRows
- MixColumns
- AddRoundKey

ByteSubstitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

ShiftRows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.

• Third row is shifted two positions to the left.

• Fourth row is shifted three positions to the left.

• The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

AddRoundKey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round. The AddRoundKey operation is the only phase of AES encryption that directly operates on the AES round key. In this operation, the input to the round is exclusive-ored with the round key.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

• Add round key

• Mix columns

• Shift rows

• Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches. However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

ADVANTAGES OF AES

One of the primary advantages of AES is its ubiquity. Since it is defined as the standard used by the US government, it is supported by most vendors. Also, it is relatively fast in both hardware and software. The three possible key lengths supported by AES allow users to pick a tradeoff between speed and security. Increased key length increases the execution time of both encryption and decryption. At this time, all three key lengths are considered secure and the best known attacks against AES reduce effective key length by at most three bits. AES uses a single S-Box for all bytes in all rounds. In contrast, DES uses eight distinct S-Boxes, which increases implementation requirements.

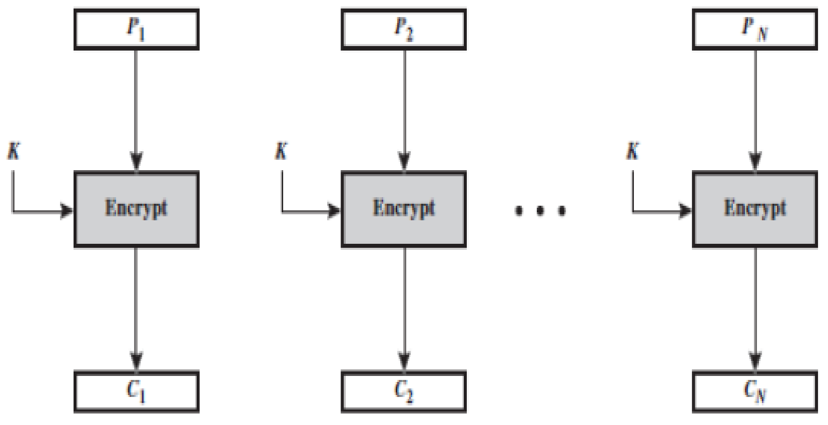
Block Ciphers Modes of Operation

Block cipher algorithm is a basic building block for providing data security. To apply a block cipher in a variety of applications, four "modes of operation" have been defined. In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. These modes are intended for use with any symmetric block cipher, including triple DES and AES. For different applications and uses, there are several modes of operations for a block cipher.

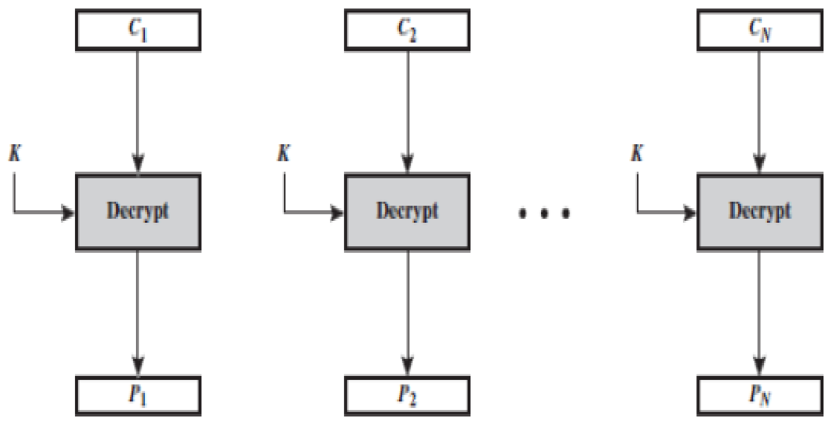
1. Electronic Code Book
2. Cipher Block Chaining Mode
3. Cipher Feedback Mode
4. Output Feedback Mode
5. Counter Mode

Electronic Codebook Mode

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



(a) Encryption

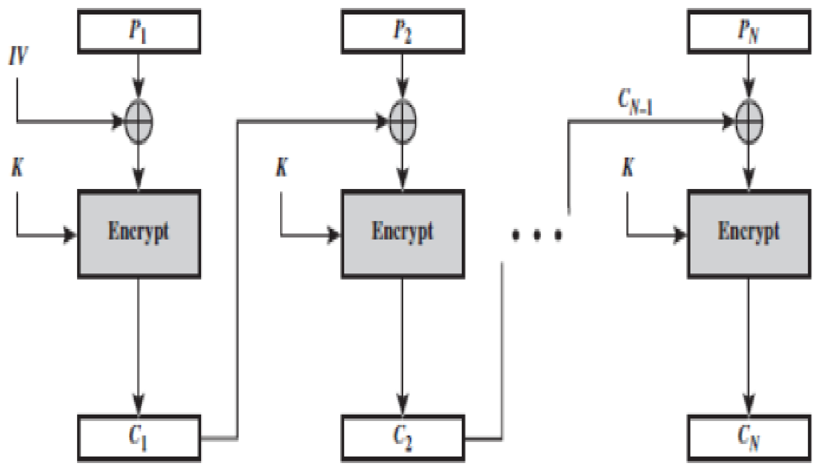


(b) Decryption

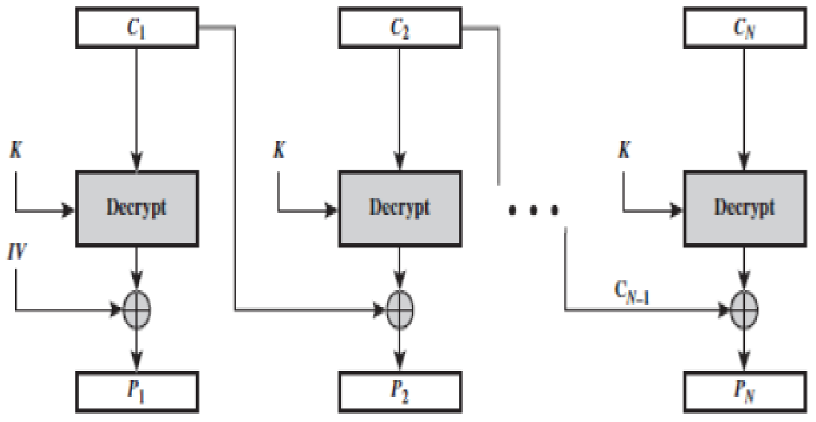
For a message longer than *b* bits, the procedure is simply to break the message into *b*-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure , the plaintext (padded as necessary) consists of a sequence of *b*-bit blocks, *P*1, *P*2,...,*PN*; the corresponding sequence of ciphertext blocks is *C*1, *C*2,..., *CN*. The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use. The most significant characteristic of ECB is that the same *b*-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext. For lengthy messages, the ECB mode may not be secure.

Cipher Block Chaining Mode

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks.



(a) Encryption

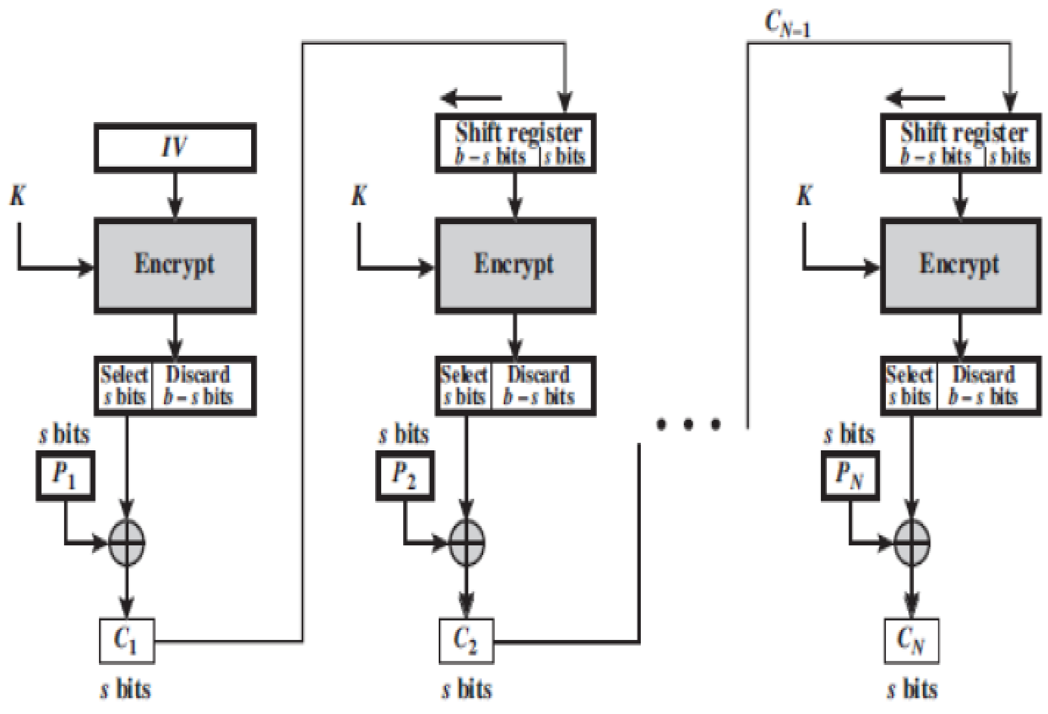


(b) Decryption

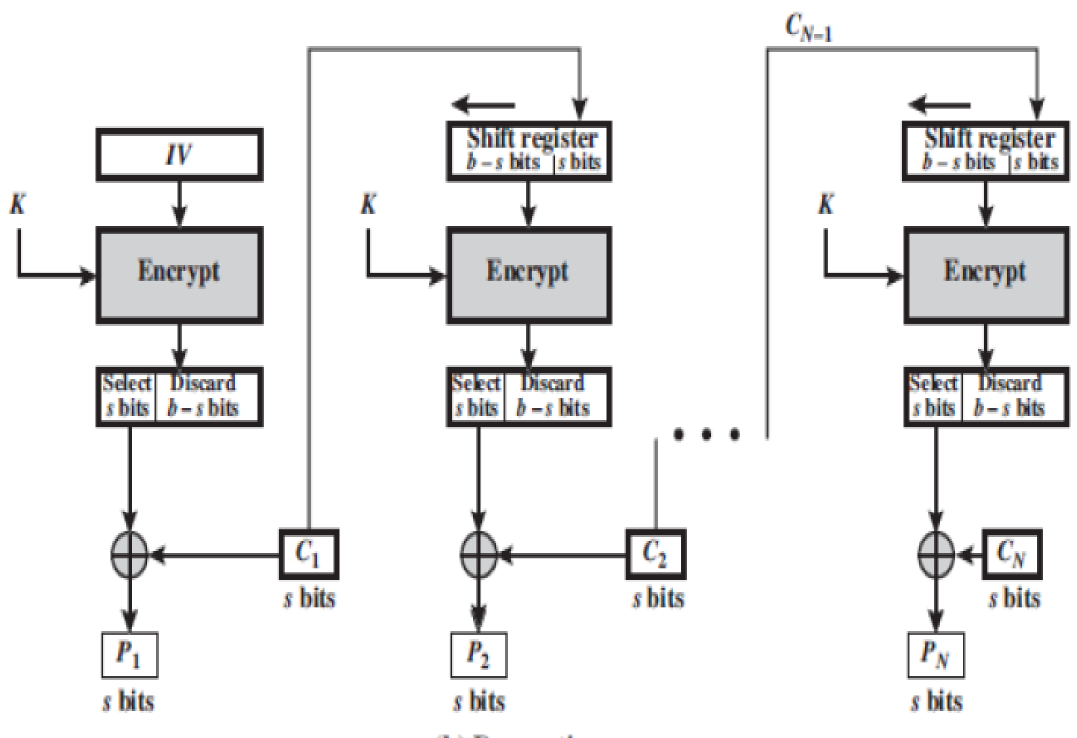
To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block. The IV must be known to both the sender and receiver but be unpredictable by a third party. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than *b* bits. In addition to its use to achieve confidentiality, the CBC mode can be used for authentication.

Cipher Feedback Mode

The DES scheme is essentially a block cipher technique that uses *b*-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. Figure depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is *s* bits; a common value is *s* = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than units of *b* bits, the plaintext is divided into *segments* of *s* bits.



(a) Encryption

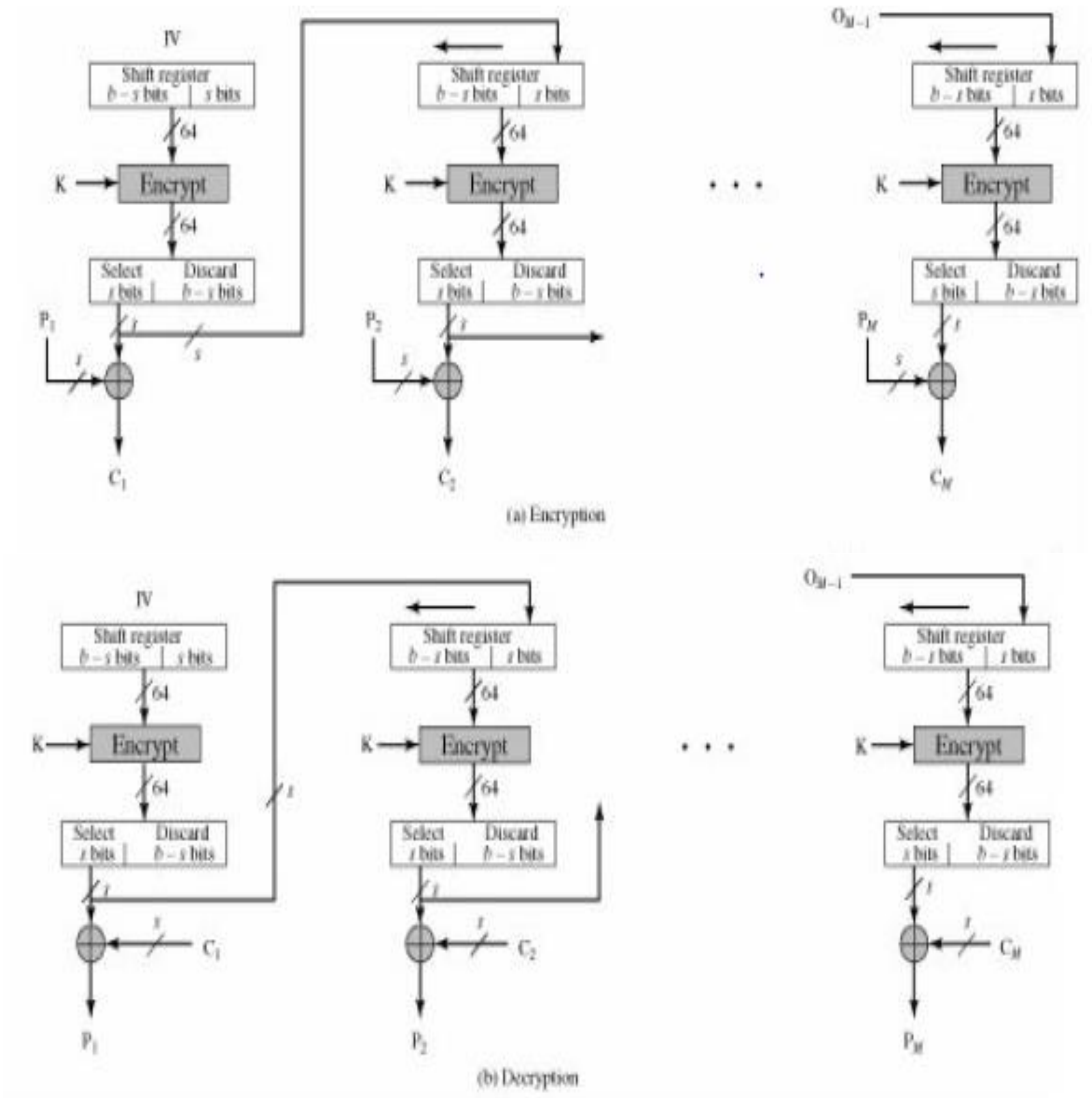


(b) Decryption

First, consider encryption. The input to the encryption function is a *b*-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) *s* bits of the output of the encryption function are XORed with the first segment of plaintext *P1* to produce the first unit of ciphertext *C1*, which is then transmitted. In addition, 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. For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.

Output Feedback Mode

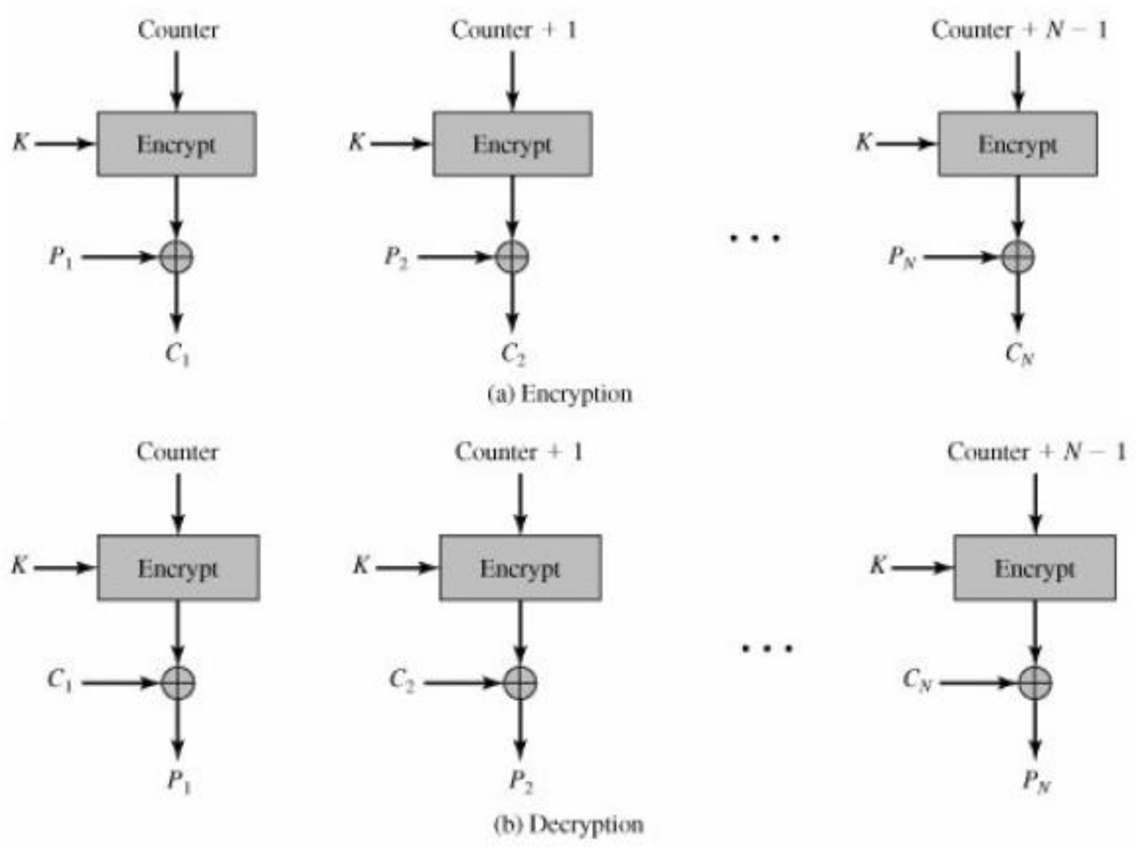
The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in Figure. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.



One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in *C*1 only the recovered value of is *P*1 affected; subsequent plaintext units are not corrupted.

Counter Mode

Figure depicts the CTR mode. A counter, equal to the plaintext block size is used. The only requirement stated is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2*b* where *b* is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.



RC4 (Ron’s Code)

An encryption technique and it uses symmetric key encryption algorithm and it was invented by Ron Rivest. RC stands for Ron’s Code .**RC4** is a stream cipher (i.e we generate streams of byte) and variable length key algorithm. This algorithm encrypts one byte at a time (or larger units on a time).
A key input is **pseudorandom bit generator,** that produces a stream of 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.

RC4 consist of two main parts

1. Key scheduling algorithm (KSA) and

2. Psuedo-Random generation algorithm (PRGA)

Basic Steps in RC4

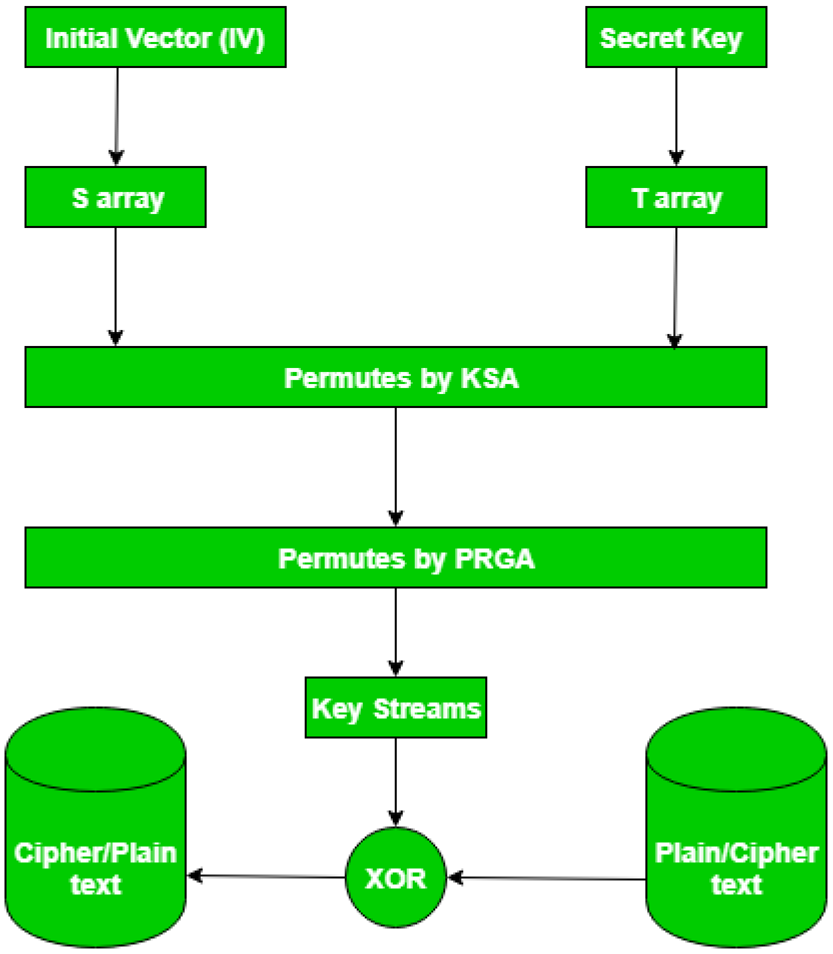
1. Initialize an array of 256 bytes

2. Run the KSA on them

3. Run the PRGA on the KSA output to get the stream

4. XOR the data with the key stream

Block diagram



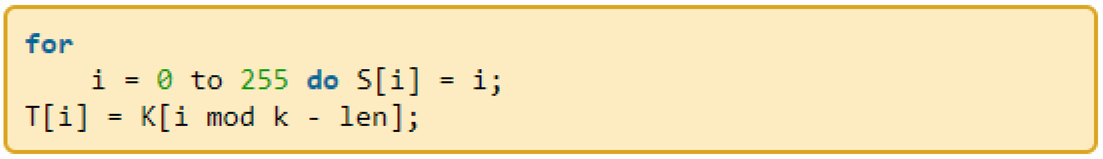
Key-GenerationAlgorithm –

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.

Key-Scheduling Algorithm:

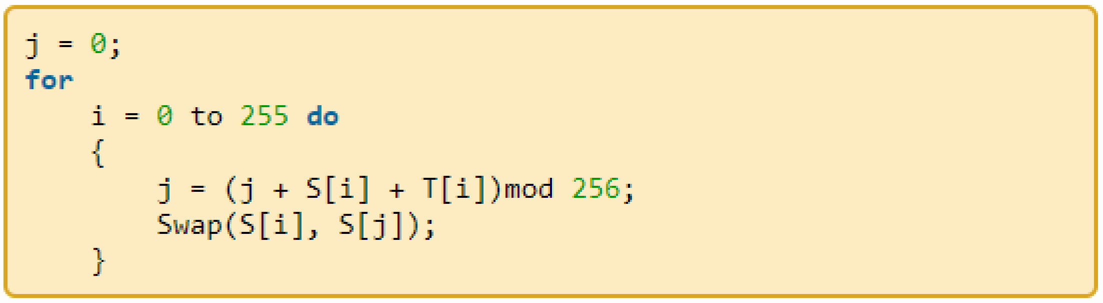
1.Initialization: The entries of S are set equal to the values from 0 to 255 in ascending order. A temporary vector T, is created.
If the length of the key k is 256 bytes, then k is assigned to T. Otherwise, for a key with length(k-len) bytes, the first k-len elements of T as copied from K and then K is repeated as many times as necessary to fill T. The idea is illustrated as follow:



```
for
    i = 0 to 255 do S[i] = i;
T[i] = K[i mod k - len];
```

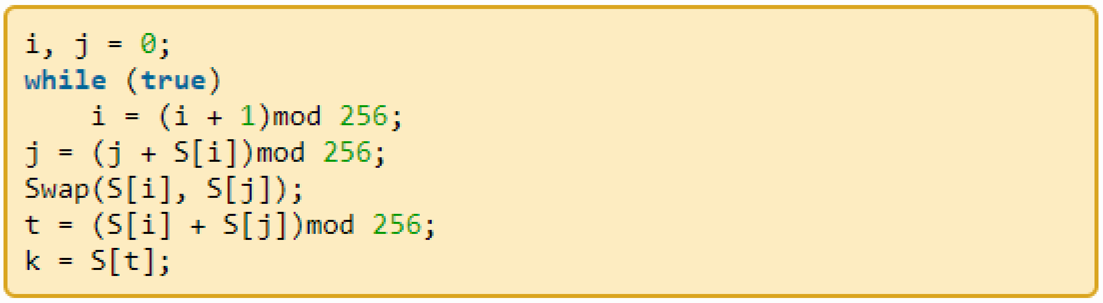
2. Key scheduling algorithm (KSA)

we 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 :



```
j = 0;
for
    i = 0 to 255 do
    {
        j = (j + S[i] + T[i])mod 256;
        Swap(S[i], S[j]);
    }
```

2.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



```
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];
```

3.Encrypt using X-Or():

<u>Summary of Encryption using RC4</u>

1. Choose a secret key

2. Run the KSA and PRGA using the key to generate a key stream

3. XOR key stream with the data to generate encrypted stream

4.Transmit Encrypted key stream

Key Distribution

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.

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.

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, Ka, known only to itself and the KDC; similarly, B shares the master key Kb with the KDC (Figure 2.36). The following steps occur:

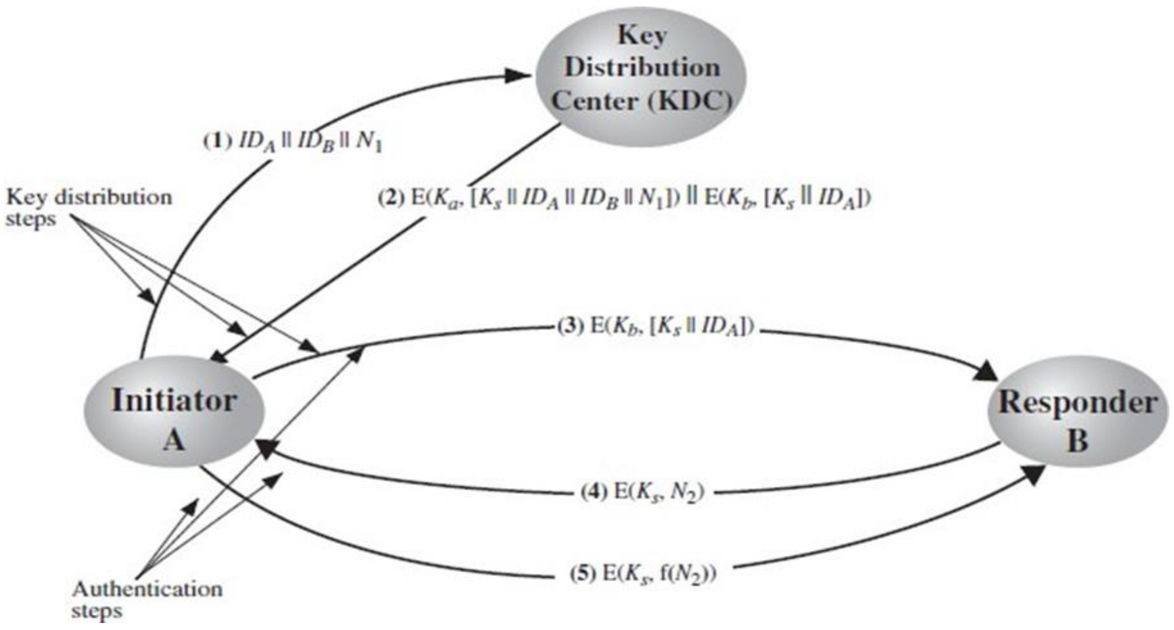

Key Distribution Center (KDC)
(1) $ID_A \| ID_B \| N_1$
Key distribution steps
(2) $E(K_a, [K_s \| ID_A \| ID_B \| N_1]) \| E(K_b, [K_s \| ID_A])$
(3) $E(K_b, [K_s \| ID_A])$
Initiator A
Responder B
(4) $E(K_s, N_2)$
(5) $E(K_s, f(N_2))$
Authentication steps

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, N1, 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 (Kb, [Ks || IDA]). Because this information is encrypted with Kb, it is protected from eavesdropping. B now knows the session key (Ks), knows that the other party is A (from IDA), and knows that the information originated at the KDC (because it is encrypted using Kb). 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, N2, to A.

5.Also using Ks, A responds with f(N2), where f is a function that performs some transformation on N2 (e.g., adding one).

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.