UNIT- II:
Symmetric Encryption
Mathematics of Symmetric Key Cryptography, Introduction to Modern Symmetric Key Ciphers, Data Encryption Standard, Advanced Encryption Standard.

Ch-4 - Mathematics of Symmetric Key Cryptography: Algebraic Structures

4.1 Algebraic structures

In previous chapter we discussed some set of numbers, such as Z, Z_n, Z_n^*, Z_p, Z_p^*.

- Cryptography requires sets of integers and specific operations that are defined for those sets.
- The combination of the set and the operations that are applied to that elements of the set is called an **algebraic structure**.
- In this chapter, we will define three common algebraic structures: groups, rings, and fields. (Figure 4.1)

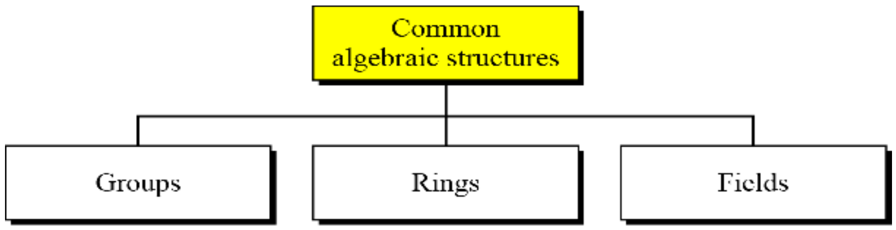


Figure 4.1 Common algebraic structures

4.2 GROUP

A **Group** (**G**) is a set of elements with a binary operation "**.**" that satisfies four properties (or axioms). A **commutative group**, also called an **abelian group**, is a group in which the operator satisfies the four properties for group plus an extra property, commutativity. The four properties for groups plus commutativity are defined as follows:

1. **Closure:** For all a, b ∈ G, then a • b ∈ G
2. **Associativity:** For all a, b, c ∈ G, then a • (b • c) = (a • b) • c
3. **Commutativity:** For all a, b ∈ G, then a • b = b • a
4. **Existence of identity:** For all element a in G, there exists and element e, identity element, s. t. e • a = a • e = a
5. **Existence of inverse:** For each a there exists a', inverse of a, so that a • a' = a' • a = e

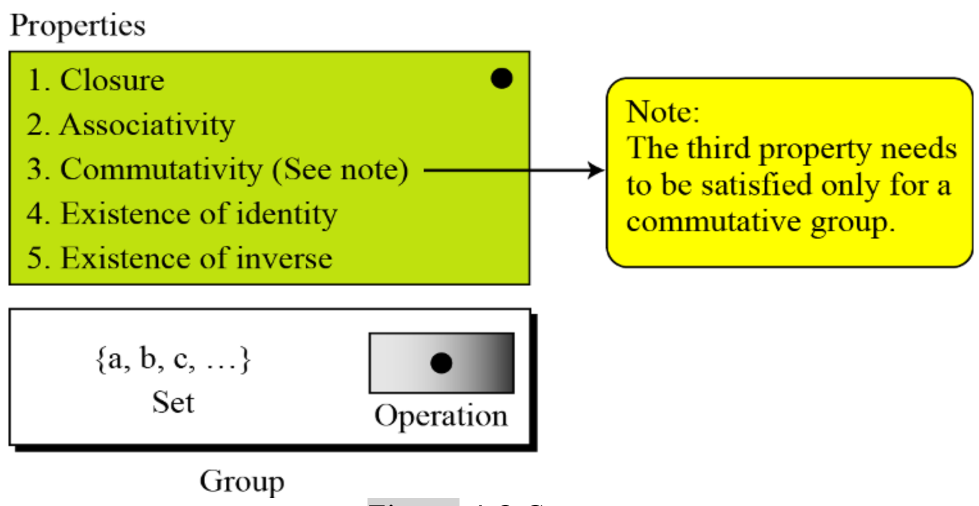


Figure 4.2 Group

Figure 4.2 shows the concept of a group and Abelian group.

Application:

- Although a group involves a single operation, the properties imposed on the operation allow the use of a pair of operations as long as they are inverses of each other.
- For example, if the defined operation is addition, the group supports both addition and subtraction, because subtraction is addition using the additive inverse. This is also true for multiplication and division. However, a group can support only addition/subtraction or multiplication/division operations, but not the both at the same time.

EXAMPLE The set of residue integers with the addition operator, $G = <Z_n, +>$, is a commutative group. We can perform addition and subtraction on the elements of this set without moving out of the set. Let us check the properties.

1. Closure is satisfied. The result of adding two integers in Z_n is another integer in Z_n.
2. Associativity is satisfied. The result of $4 + (3 + 2)$ is the same as $(4 + 3) + 2$.
3. Commutativity is satisfied. We have $3 + 5 = 5 + 3$.
4. The identify element is 0. We have $3 + 0 = 0 + 3 = 3$.
5. Every element has an additive inverse. The inverse of an element is its complement. For example, the inverse of 3 is -3 ($n - 3$ in Z_n) and the inverse of -3 is 3. The inverse allows us to perform subtraction on the set.

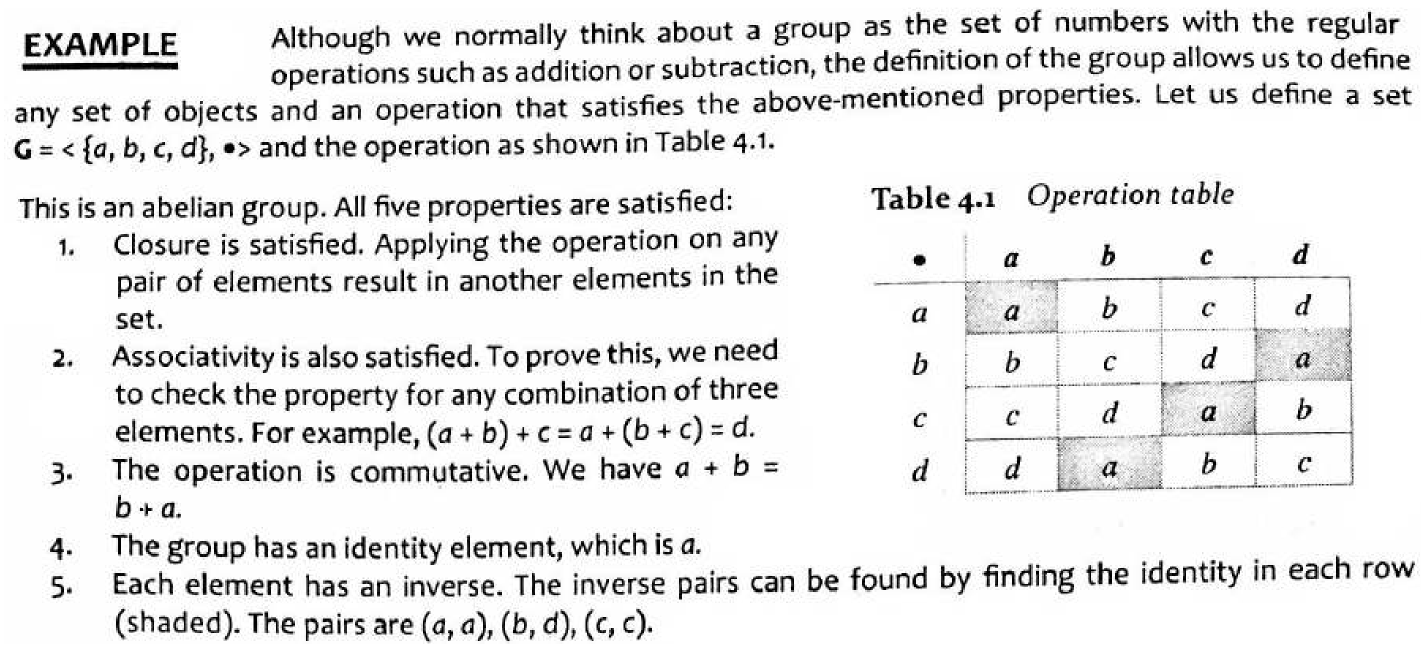

EXAMPLE Although we normally think about a group as the set of numbers with the regular operations such as addition or subtraction, the definition of the group allows us to define any set of objects and an operation that satisfies the above-mentioned properties. Let us define a set **G** = < {a, b, c, d}, •> and the operation as shown in Table 4.1.

Table 4.1 *Operation table*

•	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

This is an abelian group. All five properties are satisfied:

1. Closure is satisfied. Applying the operation on any pair of elements result in another elements in the set.
2. Associativity is also satisfied. To prove this, we need to check the property for any combination of three elements. For example, $(a + b) + c = a + (b + c) = d$.
3. The operation is commutative. We have $a + b = b + a$.
4. The group has an identity element, which is a.
5. Each element has an inverse. The inverse pairs can be found by finding the identity in each row (shaded). The pairs are (a, a), (b, d), (c, c).

Finite Group:

A group is called a finite group if the set has a finite number of elements; otherwise, it is an infinite group.

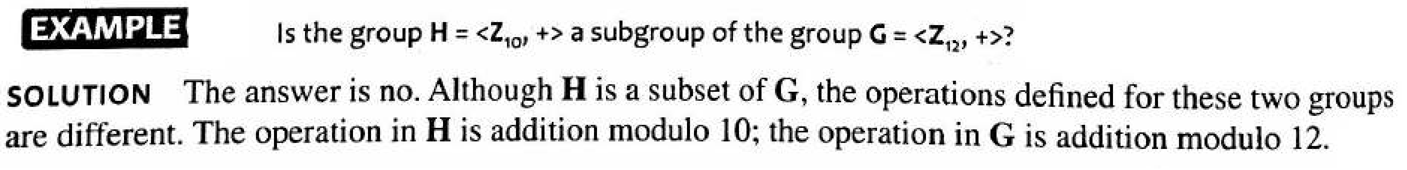
Order of a group:

The order of a group, |G|, is the number of elements in the group. If the group is not finite, its order is infinite; if the group is finite, the order is finite.

Subgroups:

A subset H of a group G is a subgroup of G if H itself is a group with respect to the operation on G. In other words, if G = < S, **.**> is a group , H = <T, **.**> is a group under the same operation, and T is a nonempty subset of S, then H is a subgroup of G. The above definition implies that:

1. If a and b are members of both groups, then c=a **.** b is also a member of both groups.
2. The group share the same identity element.
3. If a is a member of both groups, the inverse of a is also a member of both groups.
4. The group made of the identity element of G, H=< {e}, . >, is a subgroup of G.
5. Each group is a subgroup of itself.


EXAMPLE Is the group **H** = <$\mathbf{Z}_{10}$, +> a subgroup of the group **G** = <$\mathbf{Z}_{12}$, +>?

SOLUTION The answer is no. Although **H** is a subset of **G**, the operations defined for these two groups are different. The operation in **H** is addition modulo 10; the operation in **G** is addition modulo 12.

If a subgroup of a group can be generated using the power of an element, the subgroup is called the **cyclic subgroup.** The term *power* here means repeatedly applying the group operation to the element:

$$a^n \rightarrow a \,.\, a \ldots\ldots . a \quad \text{(n times)}$$

The set made from this process is referred to as <a>. Note that the duplicate elements must be discarded. Note that $a^0 = e$.

Example:

Find all cyclic subgroups of group G = <Z_6, +>.

$0^0 \bmod 6 = 0$

$1^0 \bmod 6 = 0$
$1^1 \bmod 6 = 1$
$1^2 \bmod 6 = (1 + 1) \bmod 6 = 2$
$1^3 \bmod 6 = (1 + 1 +1) \bmod 6 = 3$
$1^4 \bmod 6 = (1 + 1 + 1 + 1) \bmod 6 = 4$
$1^5 \bmod 6 = (1 + 1 + 1 + 1+ 1) \bmod 6 = 5$

$2^0 \bmod 6 = 0$
$2^1 \bmod 6 = 2$
$2^2 \bmod 6 = (2 + 2) \bmod 6 = 4$

$3^0 \bmod 6 = 0$
$3^1 \bmod 6 = 3$

$4^0 \bmod 6 = 0$
$4^1 \bmod 6 = 4$
$4^2 \bmod 6 = (4 + 4) \bmod 6 = 2$

$5^0 \bmod 6 = 0$
$5^1 \bmod 6 = 5$
$5^2 \bmod 6 = 4$
$5^3 \bmod 6 = 3$
$5^4 \bmod 6 = 2$
$5^5 \bmod 6 = 1$

Four cyclic subgroups can be made from the group G = <Z_6, +>.
They are:
H_1 = <{0}, +>,

H_2 = <{0, 2, 4}, +>,

H_3 = <{0, 3}, +>,

H_4 = G.

Example:

Find all cyclic subgroups of group G = <Z_{10}*, x>.

G has only four elements: 1, 3, 7, and 9

$1^0 \bmod 10 = 1$

$3^0 \bmod 10 = 1$
$3^1 \bmod 10 = 3$
$3^2 \bmod 10 = 9$
$3^3 \bmod 10 = 7$

$7^0 \bmod 10 = 1$
$7^1 \bmod 10 = 7$
$7^2 \bmod 10 = 9$
$7^3 \bmod 10 = 3$

$9^0 \bmod 10 = 1$
$9^1 \bmod 10 = 9$

The cyclic subgroups are H_1 = <{1}, ×>, H_2 = <{1, 9}, ×>, and H_3 = G.

Cyclic Groups:
A Cyclic group is a group that is its own cyclic subgroup. The group **G** has a cyclic subgroup H5 = G. This means that the group G is a cyclic group.

- In this case, the element that generates the cyclic subgroup can also generate the group itself. This element is referred to as a **generator**.
- If g is a generator, the element in a finite cyclic group can be written as,

$\{e, g, g^2, g^3, \ldots\ldots, g^{n-1}\}$, where $g^n = e$.
Note that a cyclic group can have many generators.

Example:

a. The group $G = <Z_6, +>$ is a cyclic group with two generators, $g = 1$ and $g = 5$.

b. The group $G = <Z_{10}^*, \times>$ is a cyclic group with two generators, $g = 3$ and $g = 7$.

Lagrange's Theorem:

- Lagrange's theorem relates the order of a group to the order of its subgroup.
- Assume that G is a group, and H is a subgroup of G. If the order of G and H are |G| and |H|, respectively, then based on this theorem, **|H| divides |G|**.
- In the above example, |G| = 6, the orders of the groups are |H1| = 1, |H2| = 3, |H3| = 2.

Order of an element:

The order of an element a in a group, ord(a), is the smallest integer n such that $a^n = e$. In other words the order of an element is the order of the cyclic group it generates.

Example:

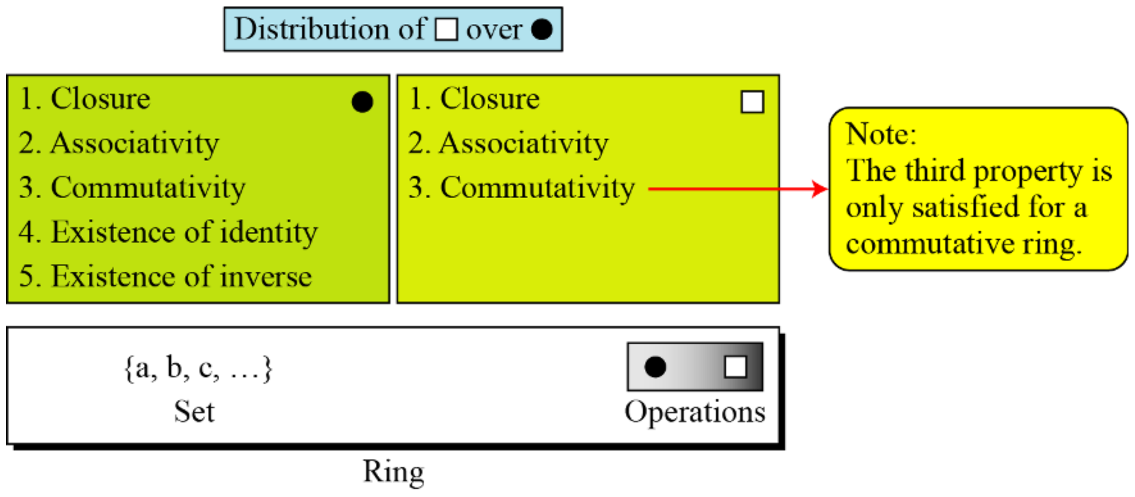
a. In the group $G = <Z_6, +>$, the orders of the elements are: ord(0) = 1, ord(1) = 6, ord(2) = 3, ord(3) = 2, ord(4) = 3, ord(5) = 6.

b. In the group $G = <Z_{10}*, \times>$, the orders of the elements are: ord(1) = 1, ord(3) = 4, ord(7) = 4, ord(9) = 2.

4.3. Ring

- A ring, denoted as R = <{ ......}, **•** **, □ > ,** is an algebraic structure with two operations.
- The first operation must satisfy all 5 properties required for an abelian group.
- The second operation must satisfy only the first two.
- In addition, the second operation must be distributed over the first.
- **Distributivity** means that for all a, b, and c elements of R, we have

  a □ (b ● c) = (a □ b) ●(a □c) and (a ● b) □ c =(a □ c) ● (b □ c).
- A **commutative ring** is a ring in which the commutative property is also satisfied for the second operation.
- The following figure shows a ring and a commutative ring.



Example:

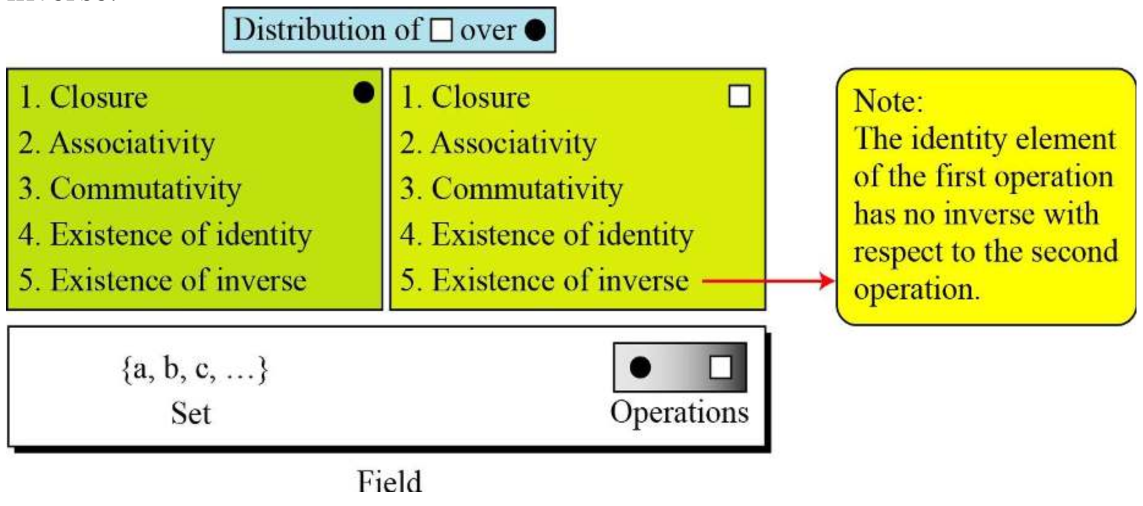
- The set Z with two operations, addition and multiplication, is a commutative ring.
- We show it by R = <Z, +, ×>.
- Addition satisfies all of the five properties; multiplication satisfies only three properties.
- Multiplication also distributes over addition.

$$(5 \times (3 + 2) = (5 \times 3) + (5 \times 2)$$

Application: A ring involves two operations. However the second operation can fail to satisfy the third and fourth properties. In other words, the first operation is actually a pair of operation such as addition and subtraction; the second operation is a single operation, such as multiplication, but not division.

4.4 Field

A **field**, denoted by F= <{……}, ● **,** □ > is a commutative ring in which the second operation satisfies all five properties defined for the first operation except that the identity of the first operation has no inverse.



Application:

A field is a structure that supports two pairs of operations that we have used in mathematics: addition/subtraction and multiplication/division. There is an exception: division by zero is not allowed.

Finite fields

- Although we have fields of infinite order, only finite fields extensively used in cryptography.
- A **finite field**, a field with a finite number of elements, are very important structures in cryptography.
- Galois showed that for a field to be finite, the number of elements should be p^n, where p is prime and n is a positive integer.

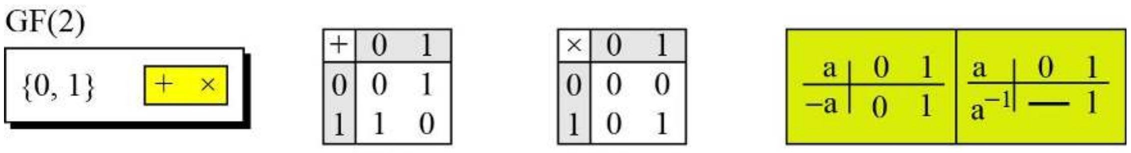
The finite fields are usually called as **Galois fields** and denoted as **GF(p^n).**

GF(P) Fields

- When n=1, we have GF(p) field. The field can be the set Zp, {0, 1, 2, ... , p-1}, with two arithmetic operations (addition and multiplication).
- Recall that each element has an additive inverse and that nonzero elements have a multiplicative inverse (no multiplicative inverse for 0).

Example1:

A very common field in this category is GF(2) with the set {0, 1} and two operations, addition and multiplication, as shown in Figure below.



+	0	1
0	0	1
1	1	0

Addition

×	0	1
0	0	0
1	0	1

Multiplication

a	0	1
−a	0	1

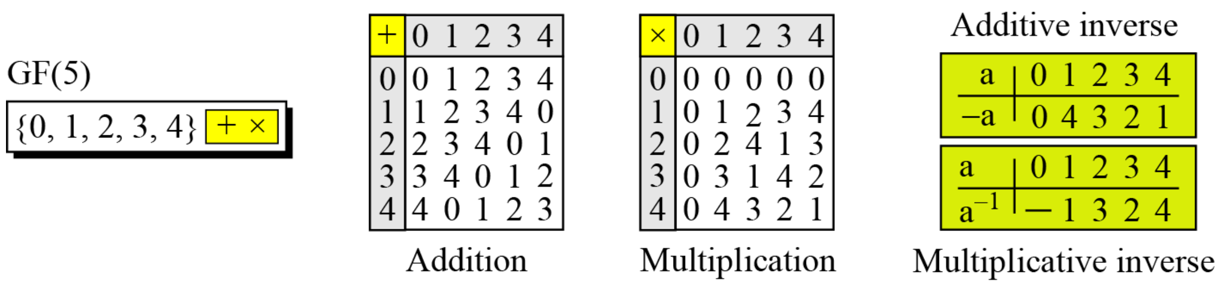
a	0	1
a^{-1}	—	1

Inverses

- There are several things to notice about this field. First, the set has only two elements, which are binary digits or bits (0 and 1).
- Second, the addition operation is actually the exclusive-or (XOR) operation we use on two binary digits.
- Third, the multiplication operation is the AND operation we use on two binary digits.
- Fourth, addition and subtraction are the same (XOR).
- Fifth, multiplication and division are the same (AND operation).

Example2:

We can define GF(5) on the set Z_5 (5 is a prime) with addition and multiplication operators as shown in Figure below.


GF(5)

{0, 1, 2, 3, 4} + ×

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Addition

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Multiplication

Additive inverse

a	0	1	2	3	4
−a	0	4	3	2	1

a	0	1	2	3	4
a^{-1}	−	1	3	2	4

Multiplicative inverse

Although we can use the extended Euclidean algorithm to find the multiplicative inverses of elements in GF(5), it is simpler to find each pair with the product equal to 1. They are (1,1), (2,3), (3,2), (4,4).

GF(p^n) Fields

In addition to GF(p) fields, we are also interested in GF(p^n) fields in cryptography. In the next section we show GF(2^n) is a very useful field in cryptography.

5.4 GF(2^n) FIELDS

In cryptography, we often need to use four operations (addition, subtraction, multiplication, and division). In other words, we need to use fields. We can work in GF(2^n) and uses a set of 2^n elements. The elements in this set are n-bit words.

1. We can use GF(p) with the set Zp, where p is the largest prime number less than 2^n. For example, if n=4, the largest prime less than 2^4 is 13. This means that we cannot use integers 13, 14, 15.
2. We can work in GF(2^n) and uses a set of 2^n elements. The elements in this set are n-bit words, for example, if n=3, the set is

   {000, 001, 010, 011, 100, 101, 110, 111}

5.4.1 Polynomials

Although we can directly define the rules for addition and multiplication operations on n-bit words that satisfy the properties in GF(2^n). It is easier to work with a representation of n-bit words, a polynomial of degree n-1. A polynomial of degree n-1 is an expression of the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x^1 + a_0x^0$$

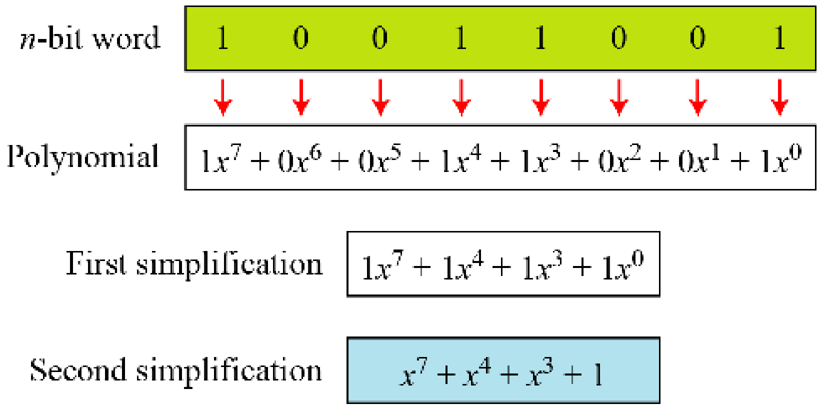
where x^i is called the ith term and a_i is called coefficient of the ith term. Although we are familiar with polynomials in algebra, to represent an n-bit word by a polynomial we need to follow some rules:

a. The power of x defines the position of the bit in the n-bit word. This means the leftmost bit is at position zero (related to x^0); the rightmost bit is at position $n - 1$ (related to x^{n-1}).

b. The coefficients of the terms define the value of the bits. Because a bit can have only a value of 0 or 1, our polynomial coefficients can be either 0 or 1.

Example:

Figure show how we can represent the 8-bit word (10011001) using a polynomials.



Example

Find the 8-bit word related to the polynomial $x^6 + x^3 + x$

x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
0	1	0	0	1	0	1	0

This is related to the 8-bit word 01001010

Note: Polynomial representing n-bit words use two fields: GF(2) and GF(2^n)

Modulus

- Before defining the operations on polynomials, we need to talk about the modulus polynoials.
- Addition of two polynomials will never create a polynomial out of the set.
- However multiplication of two polynomials may create a polynomial with degree more than n-1.
- This means we need to divide the result by a modulus and keep only the remainder as in modular arithmetic.
- For the sets of polynomials in GF(2^n) , a group of polynomials of degree n is defined as the modulus.
- The modulus in this case acts as a **prime polynomial**, which means that no polynomials in the set can divide this polynomial.
- **A prime polynomial cannot be factored** into a polynomial with degree of less than n. Such polynomials are referred to as **irreducible polynomial**.

Degree	*Irreducible Polynomials*
1	$(x + 1)$, (x)
2	$(x^2 + x + 1)$
3	$(x^3 + x^2 + 1)$, $(x^3 + x + 1)$
4	$(x^4 + x^3 + x^2 + x + 1)$, $(x^4 + x^3 + 1)$, $(x^4 + x + 1)$
5	$(x^5 + x^2 + 1)$, $(x^5 + x^3 + x^2 + x + 1)$, $(x^5 + x^4 + x^3 + x + 1)$, $(x^5 + x^4 + x^3 + x^2 + 1)$, $(x^5 + x^4 + x^2 + x + 1)$

Table: ***List of irreducible polynomials***

Note:
Addition and subtraction operations on polynomials are the same operation.

Example

$(x^2 + x^2) = 2x^2$ (Coefficients should belongs to GF(2))

$(x^2 + x^2) \bmod 2$

$= 2x^2 \bmod 2$

$= [\ (2 \bmod 2) \times (x^2 \bmod 2)\ ] \bmod 2$

$= [\ 0 \times (x^2 \bmod 2)\ ] \bmod 2$

$= 0 \bmod 2$

$= 0$

$(x^2 - x^2) \bmod 2$

$= 0 \bmod 2$

$= 0$

$$\boxed{(x^2 + x^2) = (x^2 - x^2) = 0}$$

Example
Let us do $(x^5 + x^2 + x) \otimes (x^3 + x^2 + 1)$ in GF(2^8).

We use the symbol $\otimes$ to show that we mean polynomial addition. The following shows the procedure:

$$\begin{array}{lcl}
0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0 & \oplus & \\
0x^7 + 0x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0 & & \\
\text{---} & & \\
0x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 0x^2 + 1x^1 + 1x^0 & \rightarrow & x^5 + x^3 + x + 1
\end{array}$$

There is also another short cut. Because the addition in GF(2) means the exclusive-or (XOR) operation. So we can exclusive-or the two words, bits by bits, to get the result. In the previous example, $x^5 + x^2 + x$ is 00100110 and $x^3 + x^2 + 1$ is 00001101. The result is 00101011 or in polynomial notation $x^5 + x^3 + x + 1$.

Polynomial Multiplication

1. The coefficient multiplication is done in GF(2).

2. $x^i \times x^j = x^{i+j}$.

3. The multiplication may create terms with degree more than $n - 1$, which means the result needs to be reduced using a modulus polynomial.

4. The modulus used in this case is called as **Irreducible Polynomial (IP)**.

Example

Find the result of $(x^5 + x^2 + x) \otimes (x^7 + x^4 + x^3 + x^2 + x)$ in GF(2^8) with irreducible polynomial $(x^8 + x^4 + x^3 + x + 1)$. Note that we use the symbol $\otimes$ to show the multiplication of two polynomials.

Solution

$$P_1 \otimes P_2 = x^5(x^7 + x^4 + x^3 + x^2 + x) + x^2(x^7 + x^4 + x^3 + x^2 + x) + x(x^7 + x^4 + x^3 + x^2 + x)$$

$$P_1 \otimes P_2 = x^{12} + x^9 + x^8 + x^7 + x^6 + x^9 + x^6 + x^5 + x^4 + x^3 + x^8 + x^5 + x^4 + x^3 + x^2$$

$$P_1 \otimes P_2 = (x^{12} + x^7 + x^2) \bmod (x^8 + x^4 + x^3 + x + 1) = x^5 + x^3 + x^2 + x + 1$$

To find the final result, divide the polynomial of degree 12 by the polynomial of degree 8 (the modulus) and keep only the remainder. Figure shows the process of division.

$$\begin{array}{r|l} & x^4 + 1 \\ \hline x^8 + x^4 + x^3 + x + 1 & x^{12} + x^7 + x^2 \\ & x^{12} + x^8 + x^7 + x^5 + x^4 \\ \hline & x^8 + x^5 + x^4 + x^2 \\ & x^8 + x^4 + x^3 + x + 1 \\ \hline \text{Remainder} & \boxed{x^5 + x^3 + x^2 + x + 1} \end{array}$$

Figure: Polynomial division with quotients in GF(2)

Example

Let us define a GF(2^2) field in which the set has four 2-bit words: {00, 01, 10, 11}. We can redefine addition and multiplication for this field in such a way that all properties of these operations are satisfied, as shown below.

GF (2^2) = < {00, 01, 10, 11}, $\oplus$, $\otimes$ >

Addition

$\oplus$	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

Identity: 00

$$\begin{array}{r} 1\,0 \\ \oplus\ 0\,0 \\ \hline 1\,0 \end{array} \qquad \begin{array}{r} 1\,0 \\ \oplus\ 0\,1 \\ \hline 1\,1 \end{array} \qquad \begin{array}{r} 1\,0 \\ \oplus\ 1\,0 \\ \hline 0\,0 \end{array} \qquad \begin{array}{r} 1\,0 \\ \oplus\ 1\,1 \\ \hline 0\,1 \end{array}$$

Multiplication

$\otimes$	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01

The set is {00, 01, 10, 11}
The corresponding Polynomials are 0 1 x x+1

$$\begin{array}{r|l} & \quad 1 \\ x^2+x+1 & \cancel{x^2} + 0 + \cancel{1} \\ & (-)\ \cancel{x^2} + x + \cancel{1} \\ \hline & \quad -x \end{array}$$

$$\Rightarrow -x = +x$$
$$= x$$
$$= 10$$

$$(x+1)(x+1) = x^2+x+x+1$$
$$= x^2+2x+1$$
$$= x^2+1$$

Reduced with irreducible polynomial of 2.

More calculations

$$10 \times 10$$
$$x \times x$$
$$= x^2$$

$$10 \times 11$$
$$= x \times (x+1)$$
$$= x^2+x$$

$$\begin{array}{r|l} & \quad 1 \\ x^2+x+1 & x^2 + 0 + 0 \\ & (-)\ x^2 + x + 1 \\ \hline & \quad -x-1 \end{array}$$

$$-x-1 = +x+1$$
$$= x+1$$
$$= 11$$

$$\begin{array}{r|l} & \quad 1 \\ x^2+x+1 & \cancel{x^2} + \cancel{x} + 0 \\ & (-)\ \cancel{x^2} + \cancel{x} + 1 \\ \hline & \quad -1 \end{array}$$

$$-1 = +1$$
$$= 1$$
$$= 01$$

Addition

⊕	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

Identity: 00

Multiplication

⊗	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Identity: 01

Each word is a additive inverse of itself.

Every word except 00 has a multiplicative inverse.

Addition and multiplication are defined in terms of polynomials.

To find multiplicative inverse of a polynomial, the Extended Euclidean algorithm must be applied to the modulus and the polynomial. The process is exactly the same as for integers.

Example

In GF (2^4), find the inverse of (x^2 + 1) modulo (x^4 + x + 1).

The answer is (x^3 + x + 1) as shown in Table below.

q	r_1	r_2	r	t_1	t_2	t
(x^2+1)	(x^4+x+1)	(x^2+1)	(x)	(0)	(1)	(x^2+1)
(x)	(x^2+1)	(x)	(1)	(1)	(x^2+1)	(x^3+x+1)
(x)	(x)	(1)	(0)	(x^2+1)	(x^3+x+1)	(0)
	(1)	(0)		(x^3+x+1)	(0)	

[($x^2 + 1$) ⊗ ($x^3 + x + 1$)] mod ($x^4 + x + 1$) = 1

Example

In GF(2^8), find the inverse of (x^5) modulo ($x^8 + x^4 + x^3 + x + 1$).

The answer is ($x^5 + x^4 + x^3 + x$) as shown in Table below

q	r_1	r_2	r	t_1	t_2	t
(x^3)	$(x^8+x^4+x^3+x+1)$	(x^5)	(x^4+x^3+x+1)	(0)	(1)	(x^3)
$(x+1)$	(x^5)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)
(x)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)	$(x^5+x^4+x^3+x)$
(x^3+x^2+1)	(x^3+x^2+1)	(1)	(0)	(x^4+x^3+1)	$(x^5+x^4+x^3+x)$	(0)
	(1)	(0)		$(x^5+x^4+x^3+x)$	(0)	

Multiplication using Computer

The computer implementation uses a better algorithm, repeatedly multiplying a reduced polynomial by x. For example instead of finding **x^2 ⊗ P_2** the program finds the result of **(x ⊗ (x ⊗ P_2))**.

Example

Find the result of multiplying $P_1 = (x^5 + x^2 + x)$ by $P_2 = (x^7 + x^4 + x^3 + x^2 + x)$ in GF(2^8) with irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$) using the algorithm described above.

The process is shown in Table. We first find the partial result of multiplying x^0, x^1, x^2, x^3, x^4, and x^5 by P_2. Note that although only three terms are needed, the product of $x^m \otimes P_2$ for m from 0 to 5 because each calculation depends on the previous result.

Powers	*Operation*	*New Result*	*Reduction*
$x^0 \otimes P_2$		$x^7+x^4+x^3+x^2+x$	No
$x^1 \otimes P_2$	$\boldsymbol{x} \otimes (x^7+x^4+x^3+x^2+x)$	x^5+x^2+x+1	**Yes**
$x^2 \otimes P_2$	$\boldsymbol{x} \otimes (x^5+x^2+x+1)$	$x^6+x^3+x^2+x$	No
$x^3 \otimes P_2$	$\boldsymbol{x} \otimes (x^6+x^3+x^2+x)$	$x^7+x^4+x^3+x^2$	No
$x^4 \otimes P_2$	$\boldsymbol{x} \otimes (x^7+x^4+x^3+x^2)$	x^5+x+1	**Yes**
$x^5 \otimes P_2$	$\boldsymbol{x} \otimes (x^5+x+1)$	x^6+x^2+x	No
$\mathbf{P_1} \times \mathbf{P_2} = (x^6+x^2+x) + (x^6+x^3+x^2+x) + (x^5+x^2+x+1) = \boldsymbol{x^5+x^3+x^2+x+1}$			

We can design a simple algorithm to find each partial result

1. If the most significant bit of the previous result is 0,
   - Just shift the previous result one bit to the left.
2. If the most significant bit of the previous result is 1,

- Shift it one bit to the left, and
- Exclusive-or it with the modulus without the most significant bit.

Example

Repeat the above Example using bit patterns of size 8.

We have P1 = 000100110, P2 = 10011110, modulus = 100011011 (nine bits). We show the exclusive or operation by ⊗.

Powers	*Shift-Left Operation*	*Exclusive-Or*
$x^0 \otimes P_2$		10011110
$x^1 \otimes P_2$	00111100	(00111100) ⊕ (00011010) = **<u>00100111</u>**
$x^2 \otimes P_2$	01001110	**<u>01001110</u>**
$x^3 \otimes P_2$	10011100	10011100
$x^4 \otimes P_2$	00111000	(00111000) ⊕ (00011010) = 00100011
$x^5 \otimes P_2$	01000110	**<u>01000110</u>**
$P_1 \otimes P_2$ = (00100111) ⊕ (01001110) ⊕ (01000110) = 00101111		

Multiplication using a generator

Sometimes it is easier to define the elements ofthe GF(2^n) field using a generator.

If the irreducible polynomial is f(x), then

∀ a ∈ F such that f(a) = 0

If g is the generator element then f(g) = 0

It can be proved that the elements can be generated as:

$$\boxed{\{0, g^0, g^1, g^2, \ldots, g^N\}, \text{ where } N = 2^n - 2}$$

Example

Generate the elements of the field GF(2^4)using the irreducible polynomial *f*(*x*) = $x^4 + x + 1$.

The elements 0, g^0, g^1, $g2$, and g^3 can be easily generated, because they are the 4-bit representations of 0, 1, x^2, and x^3.

g^3	g^2	g^1	g^0
1	1	1	1

$0 = 0000$

$g^0 = 0001$

$g^1 = 0010$

$g^2 = 0100$

$g^3 = 1000$

Example

Elements g^4 through g^{14}, which represent x^4 though x^{14} need to be divided by the irreducible polynomial.

To avoid the polynomial division, the relation

$f(g) = g^4 + g + 1 = 0$ can be used.

$$g^4 + g + 1 = 0 \rightarrow g^4 = -g - 1$$
$$= g + 1$$

0	$=$	0	$=$	0	$=$	0	$\longrightarrow$	0	$=$	(0000)
g^0	$=$	g^0	$=$	g^0	$=$	g^0	$\longrightarrow$	g^0	$=$	(0001)
g^1	$=$	g^1	$=$	g^1	$=$	g^1	$\longrightarrow$	g^1	$=$	(0010)
g^2	$=$	g^2	$=$	g^2	$=$	g^2	$\longrightarrow$	g^2	$=$	(0100)
g^3	$=$	g^3	$=$	g^3	$=$	g^3	$\longrightarrow$	g^3	$=$	(1000)
g^4	$=$	g^4	$=$	g^4	$=$	$g + 1$	$\longrightarrow$	g^4	$=$	(0011)
g^5	$=$	$g\,(g^4)$	$=$	$g\,(g + 1)$	$=$	$g^2 + g$	$\longrightarrow$	g^5	$=$	(0110)
g^6	$=$	$g\,(g^5)$	$=$	$g\,(g^2 + g)$	$=$	$g^3 + g^2$	$\longrightarrow$	g^6	$=$	(1100)
g^7	$=$	$g\,(g^6)$	$=$	$g\,(g^3 + g)$	$=$	$g^3 + g + 1$	$\longrightarrow$	g^7	$=$	(1011)
g^8	$=$	$g\,(g^7)$	$=$	$g\,(g^3 + g + 1)$	$=$	$g^2 + 1$	$\longrightarrow$	g^8	$=$	(0101)
g^9	$=$	$g\,(g^8)$	$=$	$g\,(g^2 + 1)$	$=$	$g^3 + g$	$\longrightarrow$	g^9	$=$	(1010)
g^{10}	$=$	$g\,(g^9)$	$=$	$g\,(g^3 + g)$	$=$	$g^2 + g + 1$	$\longrightarrow$	g^{10}	$=$	(0111)
g^{11}	$=$	$g\,(g^{10})$	$=$	$g\,(g^2 + g + 1)$	$=$	$g^3 + g^2 + g$	$\longrightarrow$	g^{11}	$=$	(1110)
g^{12}	$=$	$g\,(g^{11})$	$=$	$g\,(g^3 + g^2 + g)$	$=$	$g^3 + g^2 + g + 1$	$\longrightarrow$	g^{12}	$=$	(1111)
g^{13}	$=$	$g\,(g^{12})$	$=$	$g\,(g^3 + g^2 + g + 1)$	$=$	$g^3 + g^2 + 1$	$\longrightarrow$	g^{13}	$=$	(1101)
g^{14}	$=$	$g\,(g^{13})$	$=$	$g\,(g^3 + g^2 + 1)$	$=$	$g^3 + 1$	$\longrightarrow$	g^{14}	$=$	(1001)

The main idea is to reduce terms g^4 to g^{14} to a combination of the terms 1, g, g^2, and g^3, using the relation $g^4 = g + 1$. For example,

$$g^{12} = g\,(g^{11}) = g\,(g^3 + g^2 + g) = g^4 + g^3 + g^2 = g^3 + g^2 + g + 1$$

Additive inverses:

Additive inverse of each element is the element itself because addition and subtraction in this field are same.

$$-g^3 = g^3$$

Multiplicative inverses:

To find the multiplicative inverses g^3

$$(g^3)^{-1} = g^{-3} = g^{-3 \bmod 15} = g^{12} = g^3 + g^2 + g + 1$$

Note the exponents are calculated as modulo $2^n-1 = 15$.

See $g^3 \times g^{12} = g^{15} = g^0 = 1$

So g^3 and g^{12} are multiplicative inverses of each other.

The following show the results of addition and subtraction operations:

a. $g^3 + g^{12} + g^7 = g^3 + (g^3 + g^2 + g + 1) + (g^3 + g + 1) = g^3 + g^2 \rightarrow (1100)$

b. $g^3 - g^6 = g^3 + g^6 = g^3 + (g^3 + g^2) = g^2 \rightarrow (0100)$

The following show the result of multiplication and division operations:

a. $g^9 \times g^{11} = g^{20} = g^{20 \bmod 15} = g^5 = g^2 + g \rightarrow (0110)$

b. $g^3 / g^8 = g^3 \times g^7 = g^{10} = g^2 + g + 1 \rightarrow (0111)$

g^3/g^8

$= g^3 \times g^{-8}$

$= g^3 \times g^{-8 \bmod 15}$

$= g^3 \times g^7$

.

Ch-5 – Introduction to Modern Symmetric-Key Ciphers

5.1 Modern block ciphers

A symmetric-key modern block cipher encrypts an n-bit block of plaintext or decrypts an n-bit block of cipher text. The encryption or decryption algorithm uses a k-bit key.

If the message is fewer than n bits, padding must be added to make it an n-bit block; if more than n bits then it should be divided into n-bit blocks and the appropriate padding must be added. The common values for n are 64, 128, 256, or 512

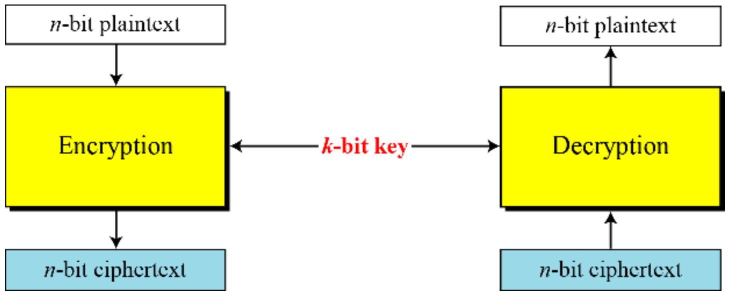


Figure 5.1 A modern block cipher

Example:

How many padding bits must be added to a message of 100 characters if 8-bit ASCII is used for encoding and the block cipher accepts blocks of 64 bits?

Solution:

Encoding 100 characters using 8-bit ASCII results in an 800-bit message. The plaintext must be divisible by 64. If |M| and |Pad| are the length of the message and the length of the padding,

$$|M| + |Pad| = 0 \bmod 64 \quad \rightarrow \quad |Pad| = -800 \bmod 64 \quad \rightarrow \quad 32 \bmod 64$$

5.1.1 Substitution or Transposition

- A modern block cipher can be designed to act as a substitution cipher or a transposition cipher.
- If the cipher is designed as substitution cipher, a 1-bit or a 0-bit can be replaced by either a 0 or a 1 bit. This means that the plain text and the cipher text can have different number of 1's.
- If the cipher is designed as transposition cipher, the bits are only reordered (transposed), there is same number of 1's in the plaintext or ciphertext.

Example:

Suppose that we have a block cipher where *n* = 64. If there are 10 1's in the ciphertext, how many trial-and-error tests does Eve need to do to recover the plaintext from the intercepted ciphertext in each of the following cases?

a. **The cipher is designed as a substitution cipher.**
b. **The cipher is designed as a transposition cipher.**

Solution:

a. In the first case, Eve has no idea how many 1's are in the plaintext. Eve needs to try all possible **2^{64} 64-bit blocks** to find one that makes sense. It would take hundreds of years.

b. In the second case, Eve knows that there are exactly 10 1's in the plaintext. Eve can launch an exhaustive-search attack using only those 64-bit blocks that have exactly 10 1's. There are only **64! / [10! * 54!]** out of 2^{64} 64-bit words. It can take less than 3 minutes.

Keyless transposition ciphers A keyless (or fixed-key) transposition cipher (or unit) can be thought of as a prewired transposition cipher when implemented in hardware.

Keyless substitution ciphers A keyless (or fixed-key) substitution cipher (or unit) can be thought of as a predefined mapping from the input to the output.

5.2 Components of a modern block cipher

Modern block ciphers normally are keyed substitution ciphers in which the key allows only partial mappings from the possible inputs to the possible outputs.

5.2.1 D-Boxes

A D-box (diffusion box) parallels the traditional transposition cipher for characters. It transposes bits.

We can find 3 types of D-boxes in modern block ciphers

1. Straight S-boxes
2. Expansion D-boxes
3. Compression D-boxes

as shown in figure 5.2.

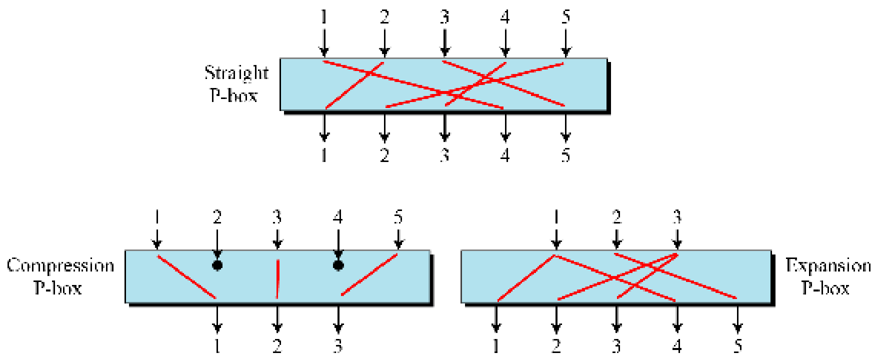


Figure 5.2 Three types of P-boxes

Invertibility A straight D-box is invertible. This means that we can use a straight D-box in the encryption cipher and its inverse in the decryption cipher. The mapping defined by a straight D-box is a permutation and thus may be referred to as **P-box**.

A straight D-box is invertible, but compression and expansion D-boxes are not.

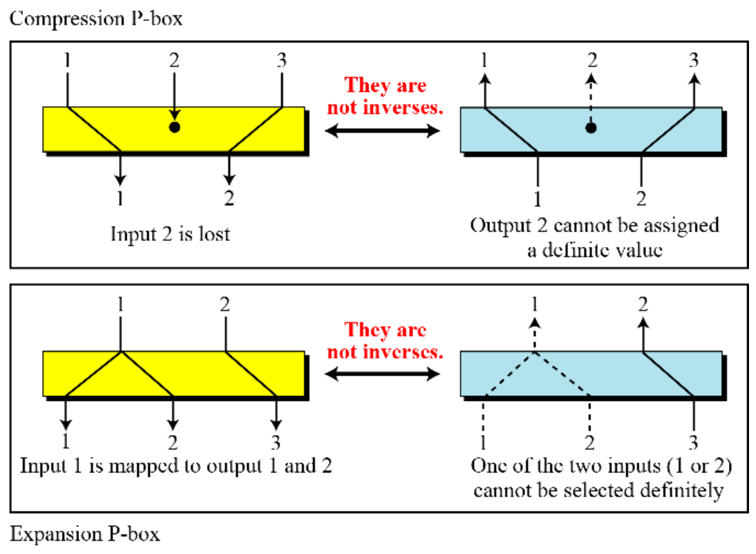

Compression P-box
1
2
3
They are not inverses.
1
2
3
1
2
Input 2 is lost
1
2
Output 2 cannot be assigned a definite value
1
2
They are not inverses.
1
2
1
2
3
Input 1 is mapped to output 1 and 2
1
2
3
One of the two inputs (1 or 2) cannot be selected definitely
Expansion P-box

Figure 5.3 Compression and expansion D-boxes are non-invertible components

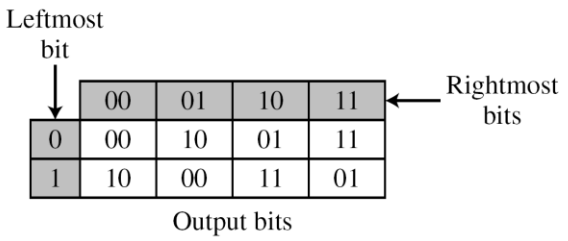
5.2.2 S-Boxes

An S-box (substitution box) can be thought of as a miniature substitution cipher. However, an S-box can have a different number of inputs and outputs.

An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same.

Example:

The following table defines the input/output relationship for an S-box of size 3 × 2. The leftmost bit of the input defines the row; the two rightmost bits of the input define the column. The two output bits are values on the cross section of the selected row and column.


Leftmost bit
Rightmost bits
Output bits

	00	01	10	11
0	00	10	01	11
1	10	00	11	01

Based on the table, an input of 010 yields the output 01. An input of 101 yields the output of 00.

Exclusive-Or (XOR)

An important component in most block ciphers is the exclusive-or operation.

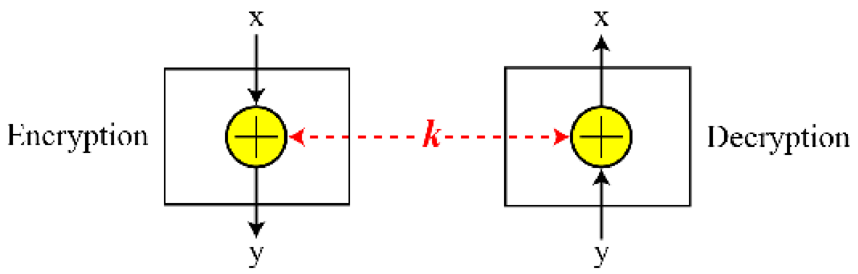


Figure 5.4 Invertibility of the exclusive-or operation

Circular Shift
Another component found in some modern block ciphers is the circular shift operation.

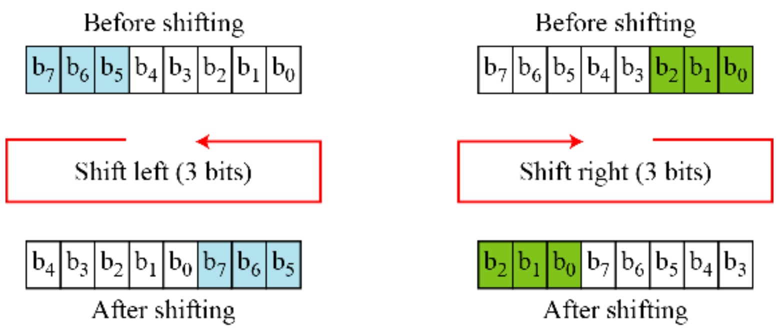

Before shifting
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
Shift left (3 bits)
b_4 b_3 b_2 b_1 b_0 b_7 b_6 b_5
After shifting
Before shifting
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
Shift right (3 bits)
b_2 b_1 b_0 b_7 b_6 b_5 b_4 b_3
After shifting

Figure 5.5 Circular shifting an 8-bit word to the left or right

Invertibility A circular left-shift operation is the invers of the circular right-shift operation. If one is used in the encryption cipher, the other can be used in the decryption cipher.

Swap
The swap operation is a special case of the circular shift operation where k = n/2. This means this operation is valid only if n (no of bits) is even number.

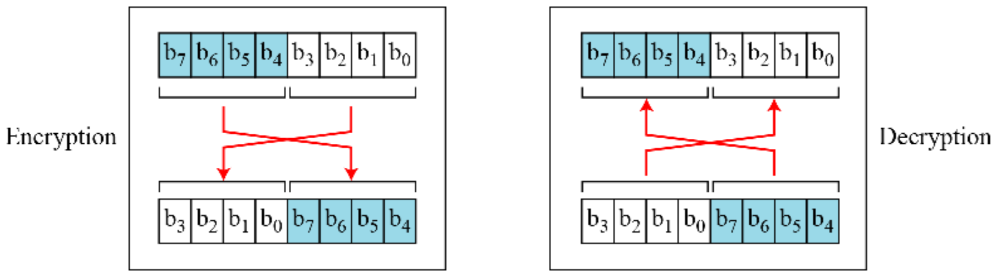

Encryption
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
b_3 b_2 b_1 b_0 b_7 b_6 b_5 b_4
Decryption

Figure 5.6 Swap operation on an 8-bit word

Split and Combine
Two other operations found in some block ciphers are split and combine.

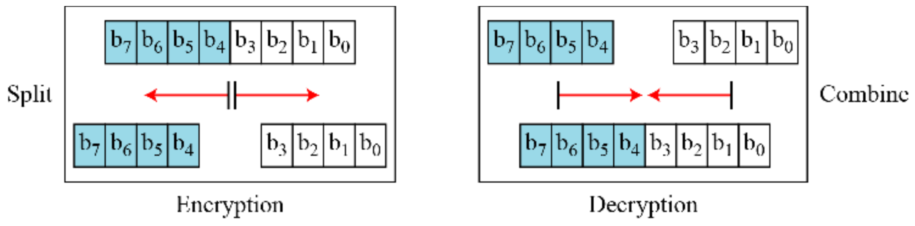


Figure 5.7 Split and combine operations on an 8-bit word

5.2.3 Product Ciphers

Product cipher is a combination of substitution, permutation and other components discussed in previous sections.
Claude Shannon introduced the concept of product cipher. He added two new important properties: diffusion, confusion.

Diffusion: Diffusion hides the relationship between ciphertext and plaintext. If a single symbol in the plaintext is changed, several or all symbols in the ciphertext will also be changed.

Diffusion hides the relationship between ciphertext and plaintext.

Confusion: Confusion hides the relationship between ciphertext and key. If a single bit in the key will be changed the most or all the bits in the ciphertext will be changed.

Confusion hides the relationship between ciphertext and key.

Rounds
Diffusion and confusion can be achieved by iterated product cipher. Iterated product cipher is combination of S-box, D-box and other components. Each iteration considered as **round.**

- Here the mapping of output according to input for both S-box, D-box has to be predetermined.
- The product cipher used only the straight D-box. Because it is invertible. So, decryption can be done by it.
- Modern block cipher use a **round key generator** for each round.

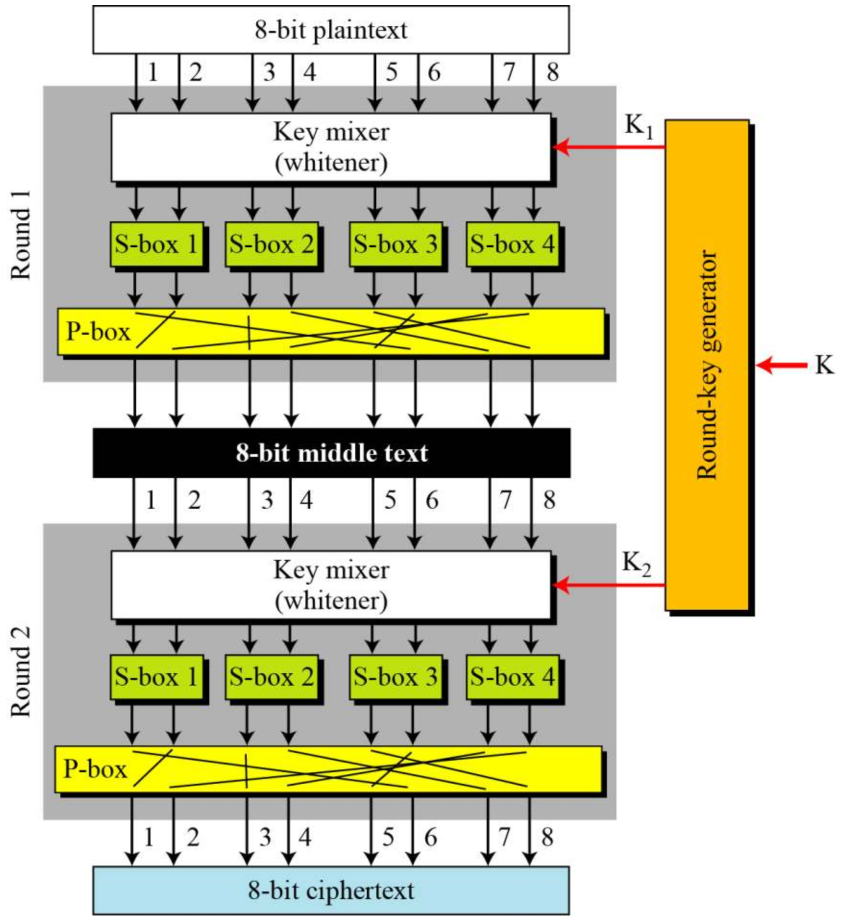


Figure 5.8 A product cipher made of two rounds

Diffusion and Confusion
The primitive design of Figure 5.8 shows how a product with the combination of S-boxes and D-boxes can guarantee diffusion and confusion. Figure 5.9 shows how changing a single bit in the plaintext affects many bits in the ciphertext as well as 8th bit K1 and bit 2 and 4 in K2 effects many bits in ciphertext.

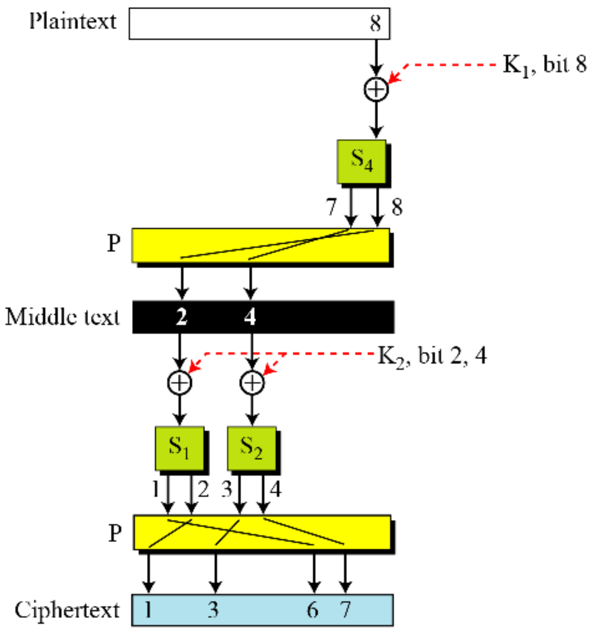


Figure 5.9 Diffusion and confusion in a block cipher

5.3 Two Classes of Product Ciphers

Modern block ciphers are all product ciphers, but they are divided into two classes.

1. Feistel ciphers
2. Non-Feistel ciphers

- Feistel cipher use both invertible and noninvertible components. This block cipher is DES.
- Non Feistel cipher use only invertible components. This type of block cipher is AES.

A Feistel cipher combines all noninvertible components in an unit and uses the same unit in encryption and decryption algorithm. The encryption and decryption algorithm are inverse of each other (using the same unit of noninvertible components).

First thought: In encryption, a noninvertible Function f(k),accepts the key as input. Then the output of the function is exclusive- ored with the plaintext. This combination is called **Mixer.** The output of this combination becomes ciphertext. It is sent to the receiver.

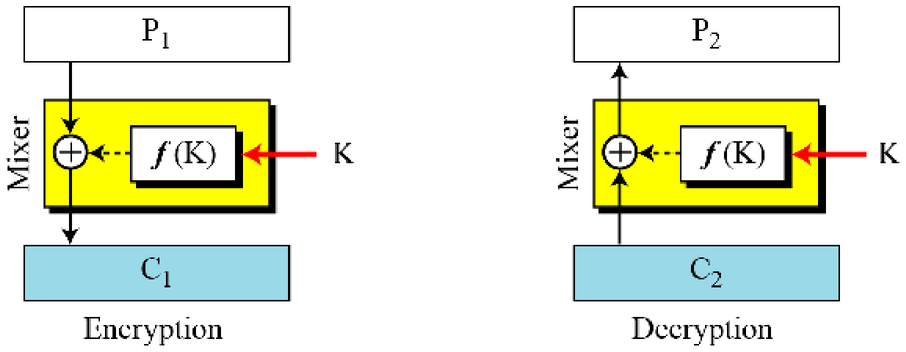


Figure 5.10 The first thought in Feistel cipher design

The input of decryption is same as the output of encryption algorithm. And the key function is also same.

That's mean C1=C2.

Encryption : $C1=P1 \oplus f(k)$

Decryption : $P2 = C2 \oplus f(k)$

$= C1 \oplus f(K)$ [C1=C2]

$= P1 \oplus f(k) \oplus f(k)$ [$X \oplus X=0$]

$= P1 \oplus 0 = P1$

The plaintext are same. So, encryption and decryption algorithm are inverse.

Improvement of Feistel cipher

Some improvements are done on the first thought of Feistel cipher to secure the text.

- In noninvertible function two inputs are considered. Function is to be a part of plaintext in encryption and ciphertext in decryption. And key can be the second input of the function.

By doing this function can be a complex element.
To achieve this goal, divide the plaintext into two equal length blocks as left and right.
Left block is represented as L and right one is R.
The inputs of the function must be same in both algorithm.

- In encryption the right section of the plaintext and key(k) goes to function as input. The output of this function exclusive-ored with left section of plaintext. And produce the left section of the cipher text.
- The right section same as the plaintext.
- In decryption algorithm reverse approach is used. The right section of the received cipher text goes to the function.

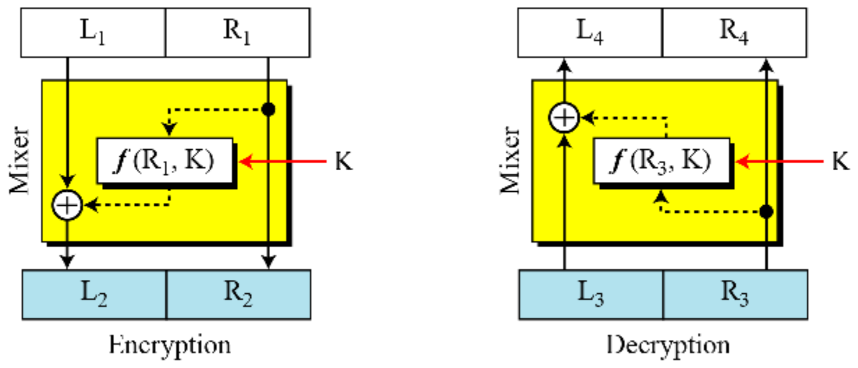


Figure 5.11 Improvement of the previous Feistel design

The input of decryption is same as the output of encryption algorithm. And the key function is also same. That's mean R2=R3, L2=L3.

Rules: Li = L(i-1) ⊕ f(R(i-1) , k)

Encryption : L2= L1 ⊕ f(R1,k) ; R1=R2 ;

Decryption : L4 = L3 ⊕ f(R3,k)

= L2 ⊕ f(R2,K) =L2 ⊕ f(R1,K) [R3=R2=R1]

= L1 ⊕ f(R1,K) ⊕ f(R1,K)

= L1 ⊕ 0 = L1

And R4=R3,

As L4=L1 and R1=R2=R3=R4 the plaintext are correctly decrypted.

Final design of Feistel cipher

But the improvement has one problem. The right half section of the plaintext never changes. So, it is very easy for the attacker to identify the left half section of the plain text. The design needs more improvement.

- First increase the number of rounds.
- Add a new element to each round: **swapper**. It allows to swap the left and right halves in each other.

Consider a two rounds Feistel cipher number of keys are K1, K2. Keys are used in reverse order in encryption and decryption. Between two rounds a middle text is created. This middle text has to be same in encryption and decryption.

The mixer and swapper are inverse.

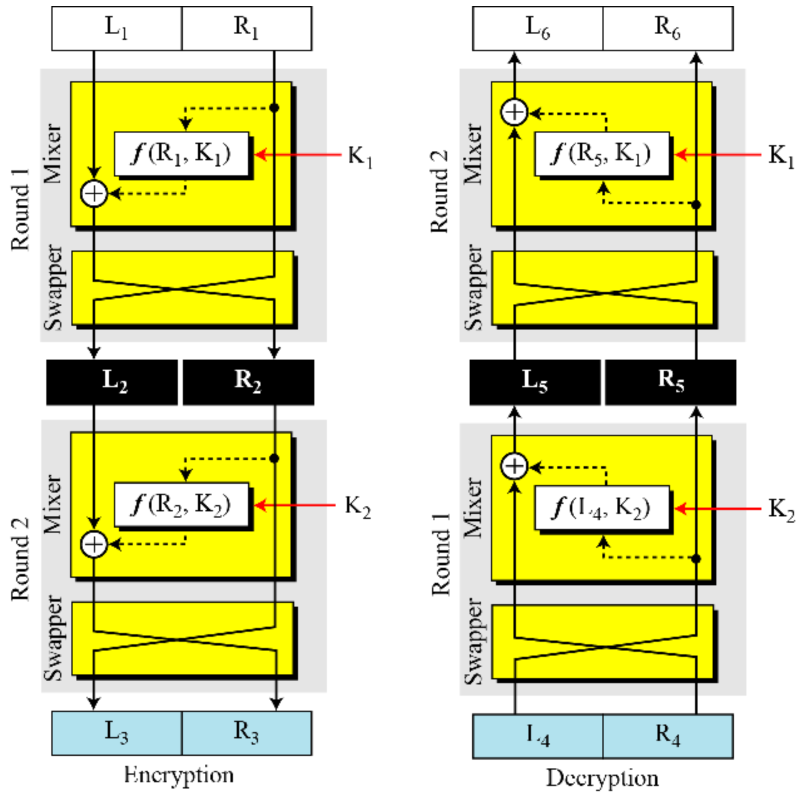


Figure 5.12 Final design of a Feistel cipher with two rounds

Non Feistel cipher

- Non Feistel cipher only used invertible components. That means S-boxes, need to have equal number of input and output, Compression and expansion D boxes are not allowed. Only can use straight D-boxes.
- In non Feistel cipher, there is no need to divide the plaintext into two equal parts.

Ch-6 – Data Encryption Standard (DES)

6.1 DATA ENCRYPTION STANDARD (DES)

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

Overview

Stream Ciphers and Block Ciphers

- A **stream cipher** is one that encrypts a digital data stream *one bit or one byte* at a time.
- A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a *block size of 64 or 128 bits* is used.
- DES is a block cipher, as shown in Figure 6.1.
- At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

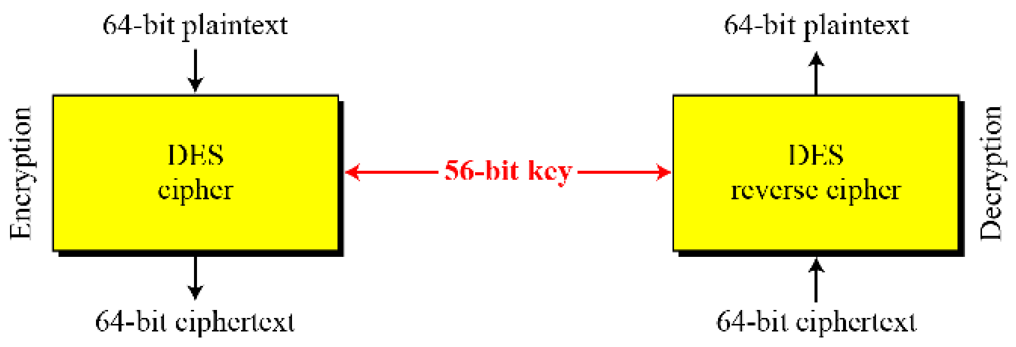


Figure 6.1 Encryption and decryption with DES

6.2 DES structure

Let us concentrate on encryption; later we will discuss decryption. The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm described later in the chapter. Figure 6.2 shows the elements of DES cipher at the encryption site.

6.2.1 Initial and Final Permutations

- The below diagram shows the initial and final permutations (P-boxes). Each of these permutations takes a 64-bit input and permutes them according to a predefined rule.
- We have shown only a few input ports and the corresponding output ports.
- These permutations are keyless straight permutations that are the inverse of each other.
- For example, in the initial permutation, the 58th bit in the input becomes the 1st bit in the output. Similarly, in the final permutation, the 1st bit in the input becomes the 58th bit in the output.

- The initial and final permutation tables are shown in table 6.1.

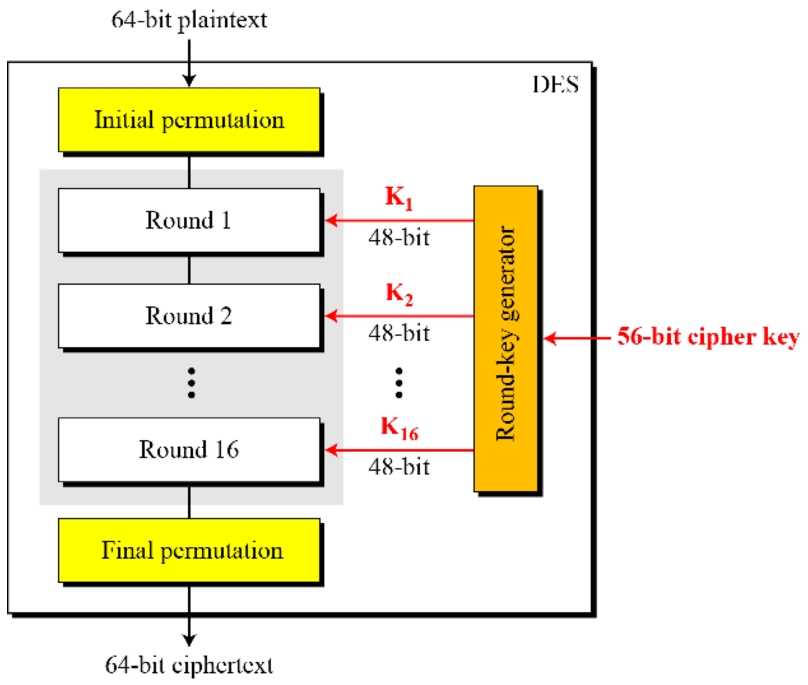


Figure 6.2 General structure of DES

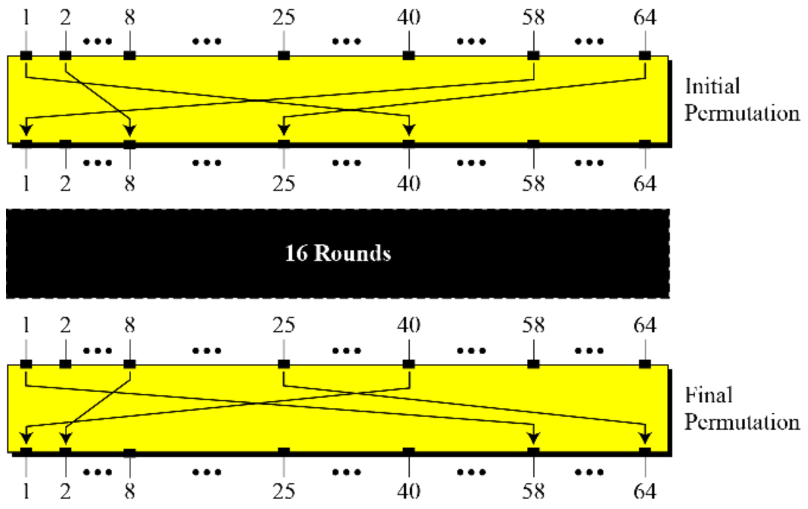


Figure 6.3 Initial and final permutation steps in DES

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

Table 6.1 Initial and final permutation tables

6.2.2 Rounds

- DES uses 16 rounds. Each round of DES is a Feistel cipher as shown in figure 6.4.
- The round takes L_{I-1} and R_{I-1} from previous round (or the initial permutation box) and creates L_I and R_I, which go to the next round (or final permutation box).
- We can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible.
- The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation.
- All noninvertible elements are collected inside the function ***f*** (R_{I-1}, K_I).

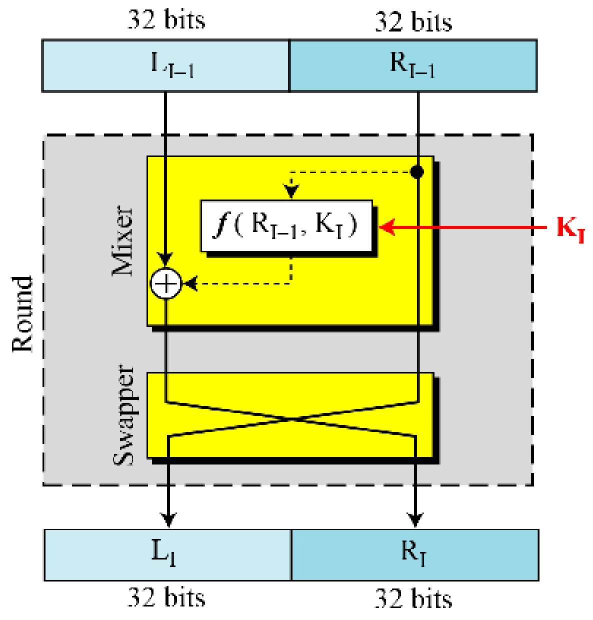


Figure 6.4 A round in DES (encryption site)

6.2.3 DES Function

- The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output.
- This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box as shown in figure 6.5.

Expansion D-box

- Since R_{I-1} is a 32-bit input and K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits. R_{I-1} is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits. This expansion permutation follows a predetermined rule.
- For each section, input bits 1, 2, 3, and 4 are copied to output bits 2, 3, 4, and 5, respectively.
- Output bit 1 comes from bit 4 of the previous section; output bit 6 comes from bit 1 of the next section.
- If sections 1 and 8 can be considered adjacent sections, the same rule applies to bits 1 and 32. Figure 6.6 shows the input and output in the expansion permutation.

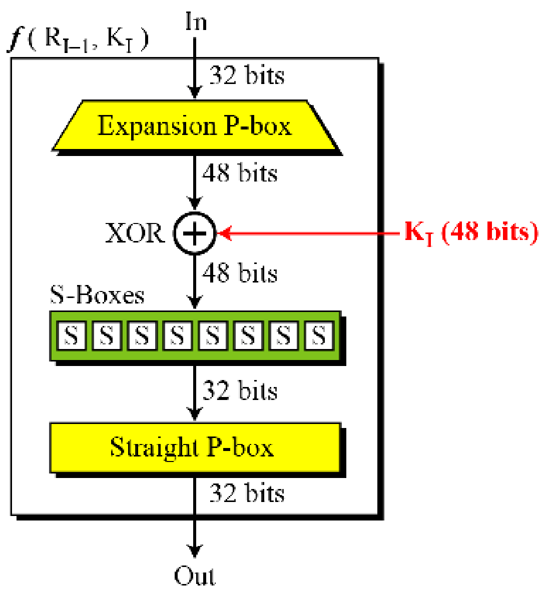


Figure 6.5 DES function

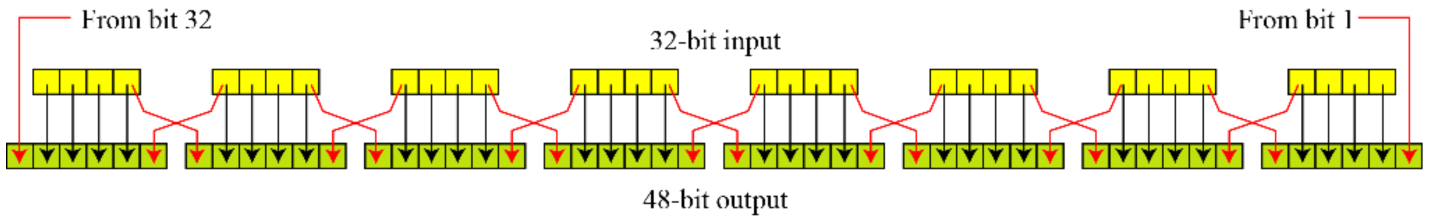


Figure 6.6 Expansion permutation

Whitener (XOR) After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. See figure 6.7.

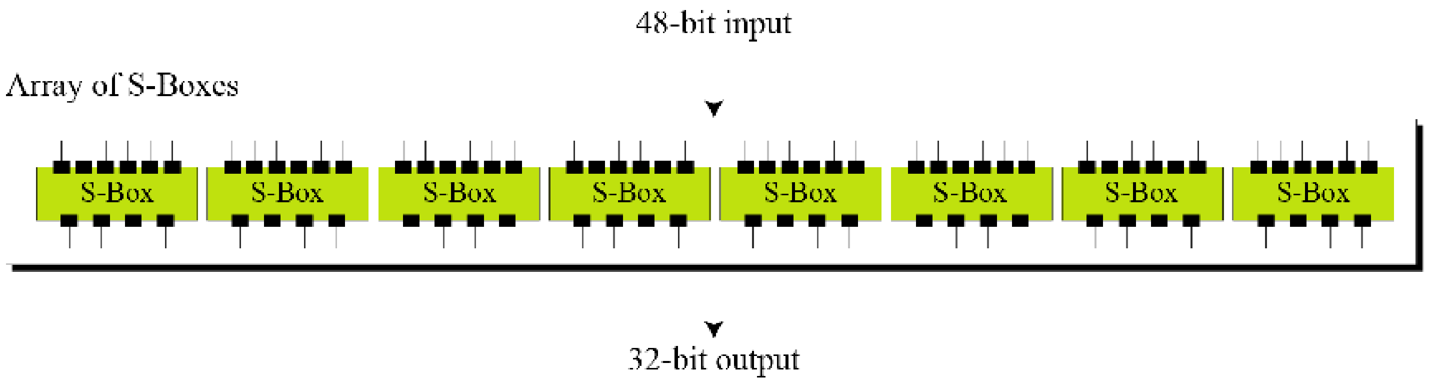


Figure 6.7 S-boxes

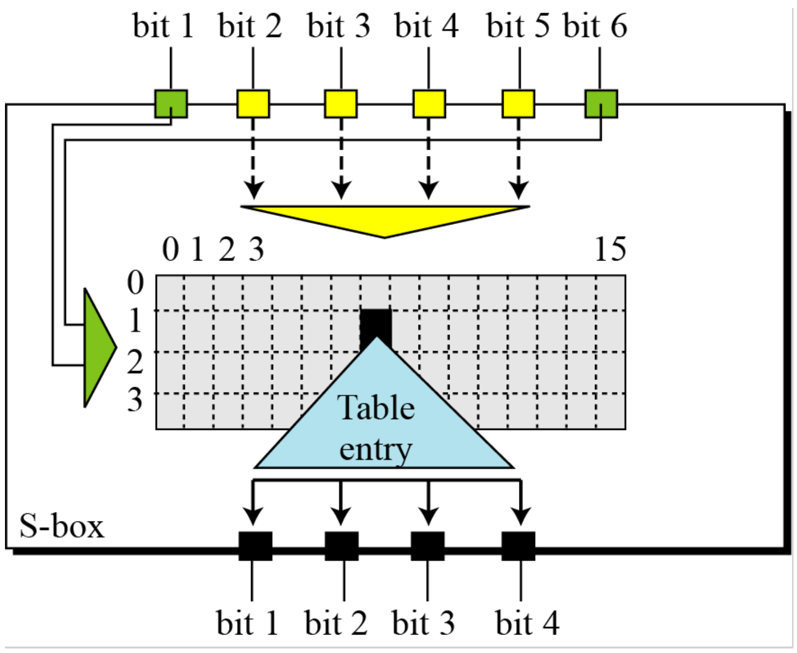


Figure 6.8 S-box rule

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Table 6.3 S-box 1

Table 6.3 shows the permutation for S-box 1. For the rest of the boxes see the textbook.

Example
The input to S-box 1 is 100011. What is the output?

Solution
If we write the first and the sixth bits together, we get 11 in binary, which is 3 in decimal. The remaining bits are 0001 in binary, which is 1 in decimal. We look for the value in row 3, column 1, in Table 6.3 (S-box 1). The result is 12 in decimal, which in binary is 1100. So the input **100011** yields the output **1100**.

Final Permutation

- The last operation in the DES function is a permutation with a 32-bit input and a 32-bit output.
- The input/output relationship for this operation is shown in Table 6.4 and follows the same general rule as previous tables.
- For example, the seventh bit of the input becomes the second bit of the output.

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Table 6.4 Straight permutation table

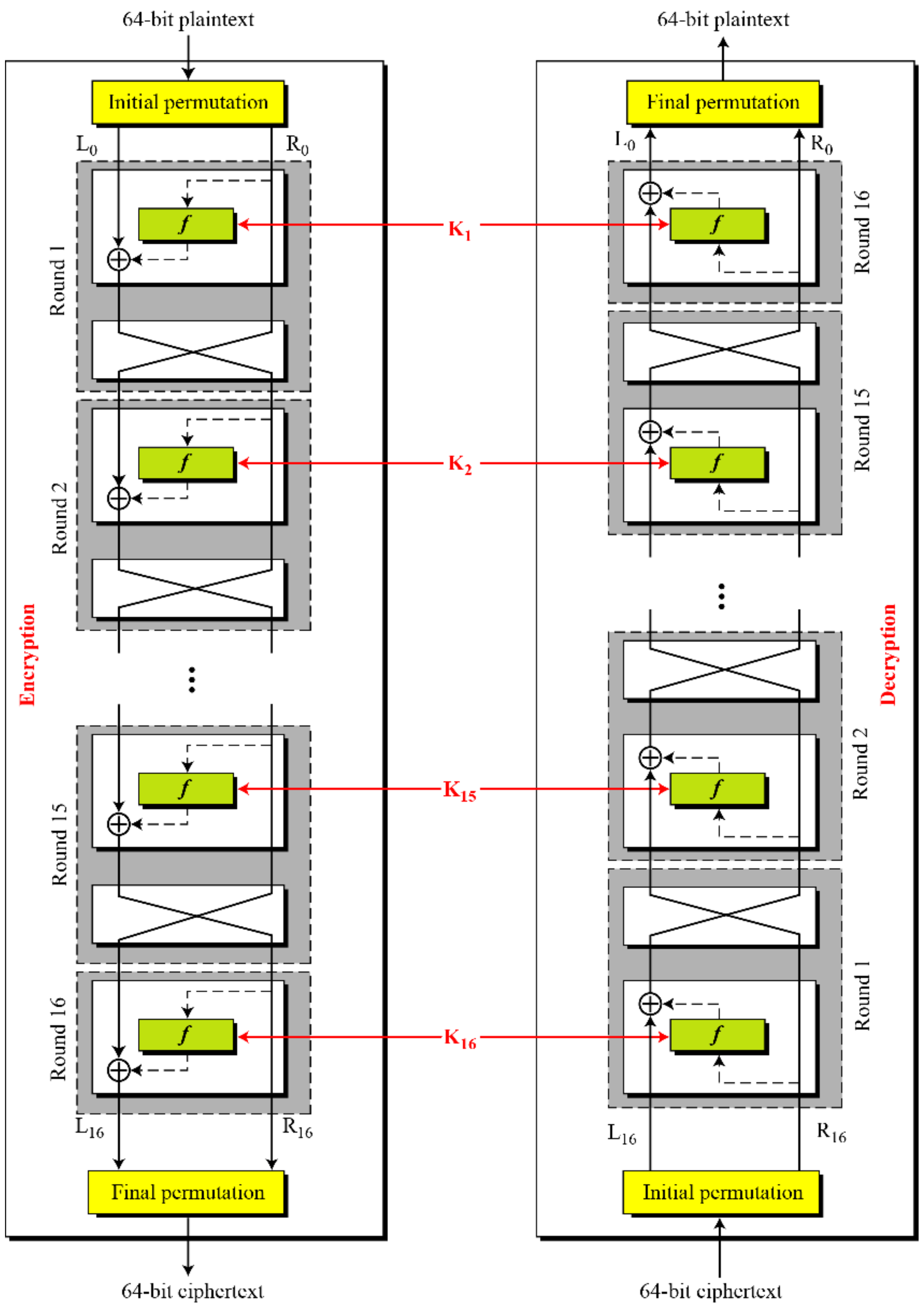
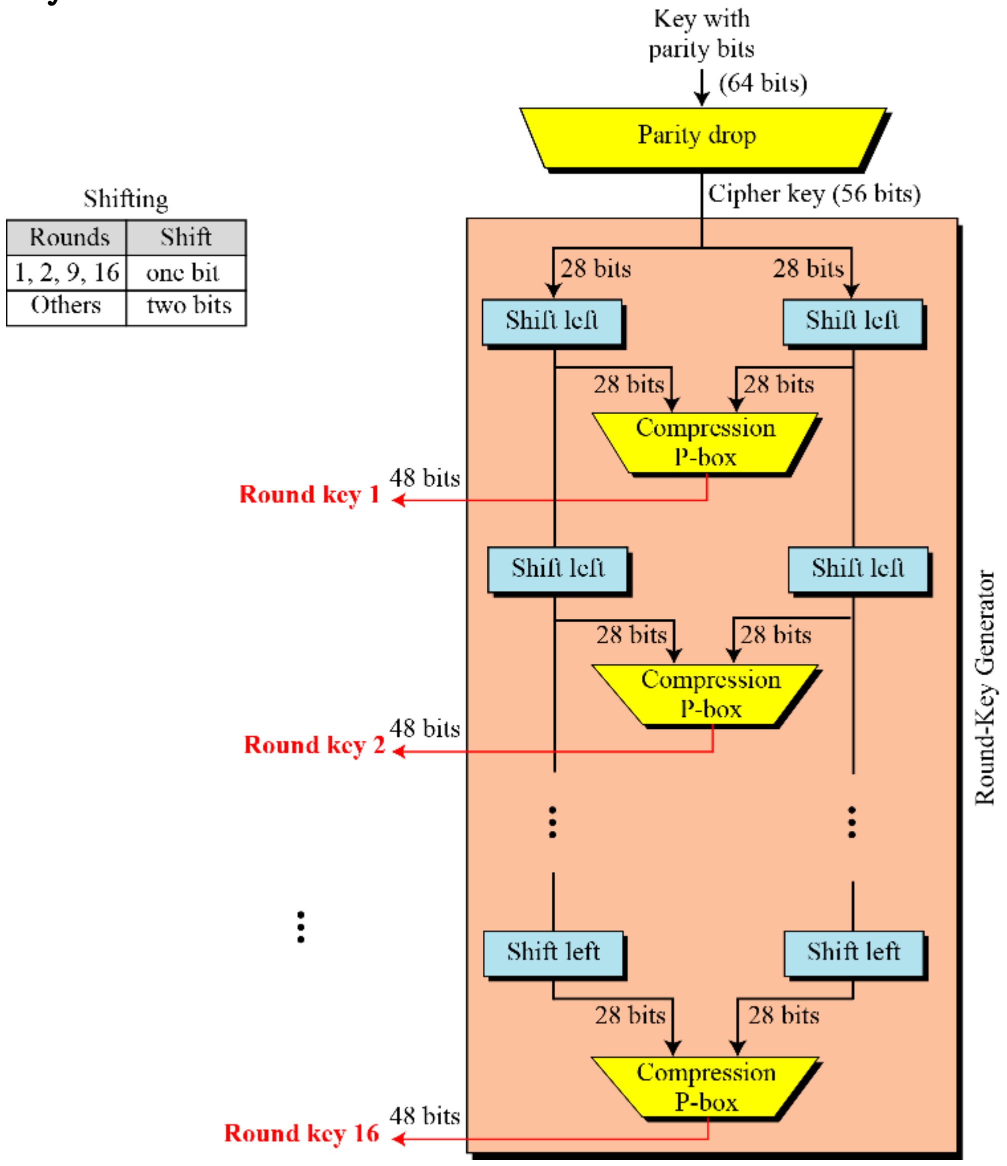


Figure 6.9 DES cipher and reverse cipher.

6.2.4 Cipher and reverse cipher

- The last round (round 16) different from the others; it has only a mixer and no swapper. This is shown in figure 6.9.
- Although the rounds are not aligned, the elements (mixer or swapper) are aligned.
- We proved in Chapter 5 that a mixer is a self-inverse; so is a swapper.
- The final and initial permutations are also inverses of each other.
- The left section of the plaintext at the encryption site, L_0, is enciphered as L_{16} at the encryption site; L_{16} at the decryption is deciphered as L_0 at the decryption site. The situation is the same with R_0 and R_{16}.
- A very important point we need to remember about the cipher is that the round keys (K_1 to K_{16}) should be applied in the reverse order. At the encryption site, round 1 uses K_1, and round 16 uses K_{16}; at the decryption site, round 1 uses K_{16} and round 16 uses K_1.

Key Generation



Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

Figure 6.10 Key generation

The **round-key generator** creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in figure 6.10

Parity Drop

- The preprocess before key expansion is a compression transposition step that we call parity bit drop.
- It drops the parity bits (bits 8, 16, 24, 32, ...64) from the 64-bit key and permutes the rest of the bits according to Table
- The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop step (a compression D-box) is shown in Table 6.

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Table 6.5 Parity-bit drop table

Circular left shift
After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. Table 6.13 shows the number of shifts for each round.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 6.6 Number of bit shifts

Compression D-Box The compression D-box changes the 58 bits to 48 bits, which are used as a key for a round. The compression step is shown in Table 6.7.

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Table 6.7 Key-compression table

6.3 DES Analysis

Critics have used a strong magnifier to analyze DES. Tests have been done to measure the strength of some desired properties in a block cipher. The elements of DES have gone through scrutinies to see if they have met the established criteria. We discuss some of these in this section.

6.3.1 Properties

Two desired properties of a block cipher are the avalanche effect and the completeness.

1. **Avalanche Effect:**

Avalanche effect means a small change in the plaintext (or key) should create a significant change in the ciphertext. DES has been proved to be strong with regard to this property.

2. **Completeness Effect:**

It means that each bit of the ciphertext needs to depend on many bits on the plaintext. The diffusion and confusion produced by D-boxes and S-boxes in DES, show a very strong completeness effect.

6.3.2 Design Criteria

The design of DES was revealed by IBM in 1994. Many tests on DES have proved that it satisfies some of the required criteria as claimed. We briefly discuss some of these design issues.

S-Boxes: We have discussed the general design criteria for S-boxes in Chapter 5; we only discuss the criteria selected for DES here. The design provides confusion and diffusion of bits from each round to the next. According to this revelation and some research, we can mention several properties of S-boxes.

1. The entries of each row are permutations of values between 0 and 15.
2. S-boxes are nonlinear. In other words, the output is not an affine transformation of the input. See Chapter 5 for discussion on the linearity of S-boxes.
3. If we change a single bit in the input, two or more bits will be changed in the output.
4. If two inputs to an S-box differ only in two middle bits (bits 3 and 4), the output must differ in at least two bits. In other words, $S(x)$ and $S(x \oplus 001100)$ must differ in at least two bits where x is the input and $S(x)$ is the output.
5. If two inputs to an S-box differ in the first two bits (bits 1 and 2) and are the same in the last two bits (5 and 6), the two outputs must be different. In other words, we need to have the following relation $S(x) \neq S(x \oplus 11bc00)$, in which b and c are arbitrary bits.
6. There are only 32 6-bit input-word pairs (x_i and x_j), in which $x_i \oplus x_j \neq (000000)_2$. These 32 input pairs create 32 4-bit output-word pairs. If we create the difference between the 32 output pairs, $d = y_i \oplus y_j$, no more than 8 of these d's should be the same.
7. A criterion similar to # 6 is applied to three S-boxes.
8. In any S-box, if a single input bit is held constant (0 or 1) and the other bits are changed randomly, the differences between the number of 0s and 1s are minimized.

D-Boxes:

Between two rows of S-boxes (in two subsequent rounds), there are one straight D-box (32 to 32) and one expansion D-box (32 to 48). These two D-boxes together provide diffusion of bits. We have discussed the general design principle of D-boxes in Chapter5. Here we discuss only the ones applied to the D-boxes used inside the DES function. The following criteria were implemented in the design of D-boxes to achieve this goal:

1. Each S-box input comes from the output of a different S-box (in the previous round).
2. No input to a given S-box comes from the output from the same box (in the previous round).
3. The four outputs from each S-box go to six different S-boxes (in the next round).
4. No two output bits from an S-box go to the same S-box (in the next round).
5. If we number the eight S-boxes, S_1, S_2, ..., S_8,
    a. An output of S_{j-2} goes to one of the first two bits of S_j (in the next round).
    b. An output bit from S_{j-1} goes to one of the last two bits of S_j (in the next round).
    c. An output of S_{j+1} goes to one of the two middle bits of S_j (in the next round).
6. For each S-box, the two output bits go to the first or last two bits of an S-box in the next round. The other two output bits go to the middle bits of an S-box in the next round.
7. If an output bit from S_j goes to one of the middle bits in S_k (in the next round), then an output bit from S_k cannot go to the middle bit of S_j. If we let $j = k$, this implies that none of the middle bits of an S-box can go to one of the middle bits of the same S-box in the next round.

Number of Rounds:

- DES uses sixteen rounds of Feistel ciphers.
- It has been proved that after eight rounds, each ciphertext is a function of every plaintext bit and every key bit; the ciphertext is thoroughly a random function of plaintext and ciphertext.
- Therefore, it looks like eight rounds should be enough. However, experiments have found that DES versions with less than sixteen rounds are even more vulnerable to known- plaintext attacks than brute-force attack, which justifies the use of sixteen rounds by the designers of DES.

6.3.3 DES Weaknesses

During the last few years critics have found some weaknesses in DES.

Weaknesses in Cipher Design

We will briefly mention some weaknesses that have been found in the design of the cipher

S-boxes:

At least three weaknesses are mentioned in the literature for S-boxes.

1. In S-box 4, the last three output bits can be derived in the same way as the first output bit by complementing some of the input bits.
2. Two specifically chosen inputs to an S-box array can create the same output.

3. It is possible to obtain the same output in a single round by changing bits in only three neighboring S-boxes.

D-boxes:

One mystery and one weakness were found in the design of D-boxes:

1. It is not clear why the designers of DES used the initial and final permutations; these have no security benefits.
2. In the expansion permutation (inside the function), the first and fourth bits of every 4- bit series are repeated.

Weakness in Key

Table 6.18 *Weak keys*

Keys before parities drop (64 bits)	*Actual key (56 bits)*
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

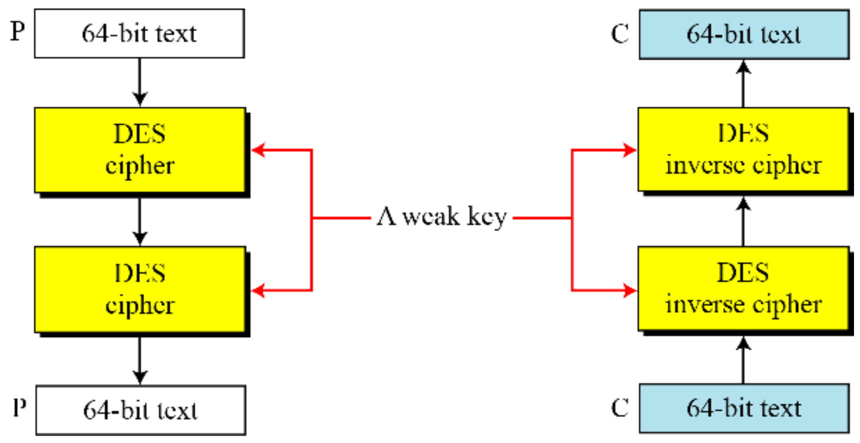


Figure: Double encryption and decryption with a weak key

First key in the pair	*Second key in the pair*
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

Figure: Semi-Weak keys

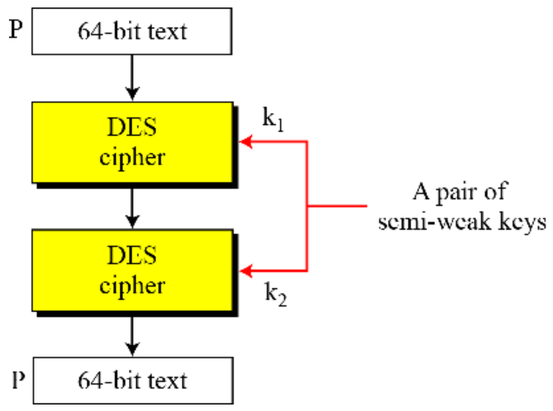


Figure: A pair of semi-weak keys in encryption and decryption

6.4 Security of DES

DES, as the first important block cipher, has gone through much scrutiny. Among the attempted attacks, three are of interest: brute-force, differential cryptanalysis, and linear cryptanalysis.

6.4.1 Brute-Force Attack

We have discussed the weakness of short cipher key in DES. Combining this weakness with the key complement weakness, it is clear that DES can be broken using 2^{55} encryptions. However, today most applications use either 3DES with two keys (key size of 112) or 3DES with three keys (key size of 168). These two multiple- DES versions make DES resistant to brute-force attacks.

6.4.2 Differential cryptanalysis:

- DES is not immune to that kind of attack. However, it has been revealed that the designers of DES already knew about this type of attack and designed S-boxes and chose 16 as the number of rounds to make DES specifically resistant to this type of attack.
- Today, it has been shown that DES can be broken using differential cryptanalysis if we have 2^{47} chosen plaintexts or 2^{55} known plaintexts.
- Although this looks more efficient than a brute-force attack, finding 2^{47} chosen plaintexts or 2^{55} know plaintexts is impractical.
- Therefore, we can say that DES is resistant to differential cryptanalysis. It has also been shown that increasing the number of rounds to 20 require more than 2^{64} chosen plaintexts for this attack, which is impossible because the possible number of plaintext blocks in DES is only 2^{64}.

6.4.3 Linear Cryptanalysis

We discussed the technique of linear cryptanalysis on modern block ciphers in Chapter 5. Linear cryptanalysis is newer than differential cryptanalysis.

DES is more vulnerable to linear cryptanalysis than to differential cryptanalysis, probably because this type of attack was not known to the designers of DES. S-boxes are not very resistant to linear cryptanalysis. It has been shown that DES can be broken using 243 pairs of known plaintexts. However, from the practical point of view, finding so many pairs is very unlikely.

6.5 Examples of block ciphers influenced by DES

6.5.1 The CAST Block Cipher

The CAST Block Cipher is an improvement of the DES block cipher, invented in Canada by Carlisle Adams and Stafford Tavares. The name of the cipher seems to be after the initials of the inventors. The CAST algorithm has 64 bit block size and has a key of size 64 bits.

CAST is based on the Feistel structure to implement the substitution permutation network. The authors state that they use the Feistel structure, as it is well studied and free of basic structural weaknesses.

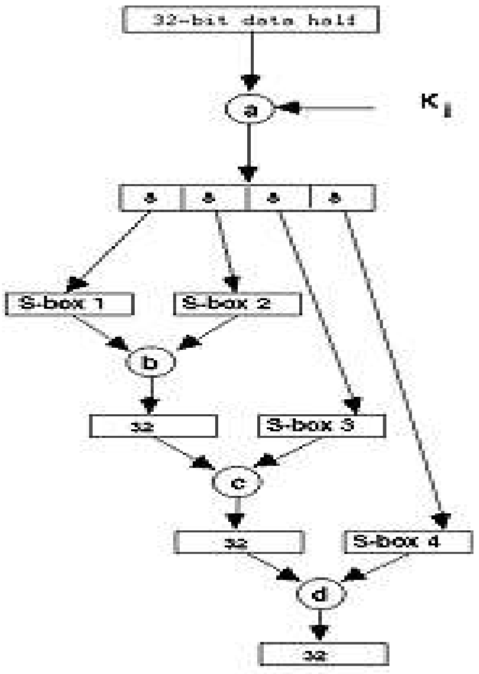


Fig. 2:
CAST Round Function

The plaintext block is divided into a left half and a right half. The algorithm has 8 rounds. Each round is essentially a Feistel structure. In each round the right half is combined with the round key using a function *f* and then XOR-ed with the left half. The new left half after the round is the same as the right half before the round. After 8 iterations of the rounds, the left and the right half are concatenated to form the ciphertext.

Key Scheduling of CAST The key scheduling in CAST has three main components:

1. A key transformation step which converts the primary key (input key) to an intermediate key.
2. A relatively simple bit-selection algorithm mapping the primary key and the intermediate key to a form, referred as partial key bits.
3. A set of key-schedule S-Boxes which are used to create subkeys from the partial key bits.

Let, the input key be denoted by KEY = $k_1k_2k_3k_4k_5k_6k_7k_8$, where k_i is the i^{th} byte of the primary key. The key transformation step generates the intermediate key, KEY' = $k'_1k'_2k'_3k'_4k'_5k'_6k'_7k'_8$ as follows:

$$k'_1k'_2k'_3k'_4 = k_1k_2k_3k_4 \approx S_1[k_5] \approx S_2[k_7]$$
$$k'_5k'_6k'_7k'_8 = k_5k_6k_7k_8 \approx S_1[k'_2] \approx S_2[k'_4]$$

Here, S_1 and S_2 are key-schedule S-Boxes of dimension 8 ¥ 32.
Subsequently, there is a bit-selection step which operates as shown below:

$$K'_1 = k_1k_2$$
$$K'_2 = k_3k_4$$
$$K'_3 = k_5k_6$$
$$K'_4 = k_7k_8$$
$$K'_5 = k'_4k'_3$$
$$K'_6 = k'_2k'_1$$
$$K'_7 = k'_8k'_7$$
$$K'_8 = k'_6k'_5$$

The partial key bits are used to obtain the subkeys, K_i. The subkeys are 32 bits, and are obtained as follows:

$$K_i = S_1(K'_{i,1}) \approx S_2(K'_{i,2})$$

Here, K_{i_j} is the j^{th} byte of K_i. Thus the 8 round subkeys are obtained.
The CAST block cipher can also be implemented with 128 bits, and is referred to as CAST-128. The essential structure of the cipher is still the same as discussed above.

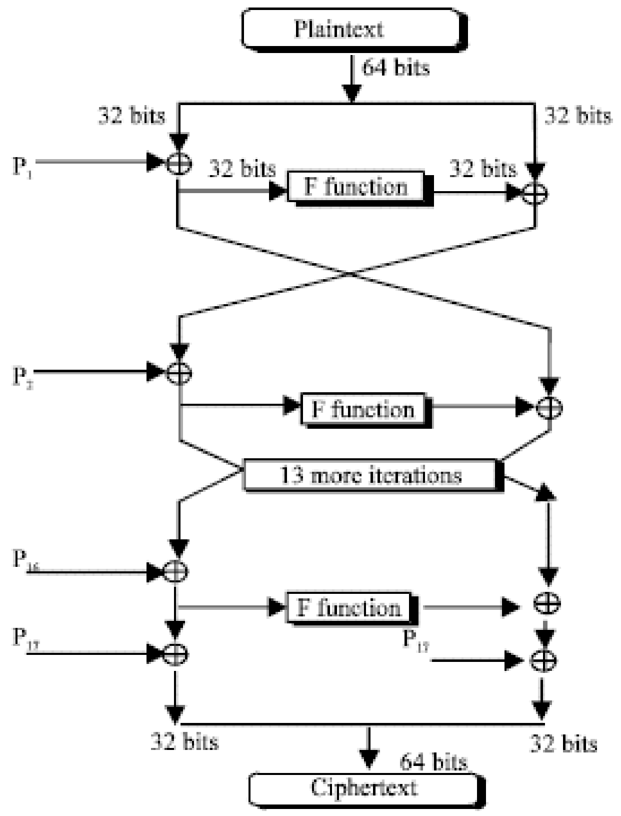
6.5.2 BLOWFISH

Blowfish is a 64-bit block cipher invented by Bruce Schneier. Blowfish was designed for fast ciphering on 32-bit microprocessors. Blowfish is also compact and has a variable key length which can be increased to 448 bits.

Blowfish is suitable for applications where the key does not change frequently like communication links or file encryptors. However for applications like packet switching or as a one-way hash function, it is unsuitable. Blowfish is not ideal for smart cards, which requires even more compact ciphers. Blowfish is faster than DES when implemented on 32-bit microprocessors.

Round Structure

The algorithm is based on the Feistel structure and has two important parts: The round structure and the key expansion function.



```
Divide x into two 32-bit halves: $x_L$, $x_R$
For $i$ = 1 to 16:
    $x_L = x_L \approx P_i$
    $x_R = F[x_L] \approx x_R$
    Swap $x_L$ and $x_R$
(undo the last swap)
$x_R = x_R \approx P_{17}$
$x_L = x_L \approx P_{18}$
Ciphertext = Concatenation of $x_L$ and $x_R$
```

The function F is central to the security of the block cipher and is defined as below:
Divide x_L into four 8-bit parts: a, b, c, d

$$F[x_L] = ((S_1[a] + S_2[b] \bmod 2^{32}) \approx S_3[c]) + S_4[d] \bmod 2^{32}$$

Key Scheduling Algorithm

The subkeys are computed using the following method:

1. The P-array and then the four *S*-Boxes are initialized with a fixed string. The string is the hexadecimal digits of π.
2. *P*1 is XOR-ed with 32 bits of the key, *P*2 is XOR-ed with the next 32 bits of the key, and so on for all the bits of the key. If needed the key bits are cycled to ensure that all the P- array elements are XOR-ed.

3. An all-zero string is encrypted with the Blowfish algorithm, with the subkeys *P*1 to *P*18 obtained so far in steps 1 and 2.
4. *P*1 and *P*2 are replaced by the 64 bit output of step 3.
5. The output of step 3 is now encrypted with the updated subkeys to replace *P*3 and *P*4 with the ciphertext of step 4.
6. This process is continued to replace all the *P*-arrays and the *S*-Boxes in order.

This complex key-scheduling implies that for faster operations the subkeys should be precomputed and stored in the cache for faster access.

Security analysis by Serge Vaudenay shows that for a Blowfish algorithm implemented with known S-Boxes (note that in the original cipher the S-Boxes are generated during the encryption process) and with r-rounds, a differential attack can recover the P-array with 2^{8r+1} chosen plaintexts.

6.5.3 IDEA

IDEA is another block cipher. It operates on 64 bit data blocks and the key is 128 bit long. It was invented by Xuejia Lai and James Massey, and named IDEA (International Data Encryption Algorithm) in 1990, after modifying and improving the initial proposal of the cipher based on the seminal work on Differential cryptanalysis by Biham and Shamir.

The design principle behind IDEA is the “mixing of arithmetical operations from different algebraic groups”. These arithmetical operations are easily implemented both in hardware and software.

The underlying operations are XOR, addition modulo 2^{16}, multiplication modulo $2^{10}+1$.

The cipher obtains the much needed non-linearity from the later two arithmetical operations and does not use an explicit S-Box.

Round Transformation of IDEA The 64-bit data is divided into four 16 bit blocks: *X*1, *X*2, *X*3, *X*4. These four blocks are processed through eight rounds and transformed by the above arithmetical operations among each other and with six 16 bit subkeys. In each round the sequence of operations is as follows:

3. Multiply *X*1 and the first subkey.
4. Add *X*2 and the second subkey.
5. Add *X*3 and the third subkey.
6. Multiply *X*4 and the fourth subkey.
7. XOR the results of step 1 and 3.
8. XOR the results of step 2 and 4.
9. Multiply the results of steps 5 with the fifth subkey.
10. Add the results of steps 6 and 7.

11. Multiply the results of steps 8 with the sixth subkey.

12. Add the results of steps 7 and 9.

13. XOR the results of steps 1 and 9.

14. XOR the results of steps 3 and 9.

15. XOR the results of steps 2 and 10.

16. XOR the results of steps 4 and 10.

The outputs of steps 11, 12, 13 and 14 are stored in four words of 16 bits each, namely *Y*1, *Y*2, *Y*3 and *Y*4. The blocks *Y*2 and *Y*3 are swapped, and the resultant four blocks are the output of a round of IDEA. It may be noted that the last round of IDEA does not have the swap step.

Instead the last round has the following additional transformations:

1. Multiply Y1 and the first subkey.
2. Add Y2 and the second subkey.
3. Add Y3 and the third subkey.
4. Multiply Y4 and the fourth subkey.

Finally, the ciphertext is the concatenation of the blocks Y1, Y2, Y3 and Y4.

Key Scheduling of IDEA

- IDEA has a very simple key scheduling. It takes the 128 bit key and divides it into eight 16 bit blocks.
- The first six blocks are used for the first round, while the remaining two are to be used for the second round.
- Then the entire 128 bit key is given a rotation for 25 steps to the left and again divided into eight blocks.
- The first four blocks are used as the remaining subkeys for the second round, while the last four blocks are to be used for the third round.
- The key is then again given a left shift by 25 bits, and the other subkeys are obtained. The process is continued till the end of the algorithm.
- For decryption, the subkeys are reversed and are either the multiplicative or additive inverse of the encryption subkeys. The all zero subkey is considered to represent $2^{16} = -1$ for the modular multiplication operation, mod $2^{16}+1$. Thus the multiplicative inverse of 0 is itself, as –1 multiplied with –1 gives 1, the multiplicative identity in the group.
- Computing these keys may have its overhead, but it is a onetime operation, at the beginning of the decryption process.
- IDEA has resisted several cryptanalytic efforts. The designers gave argument to justify that only 4 rounds of the cipher makes it immune to differential cryptanalysis.
- Joan Daemen, Rene Govaerts and Joos Vandewalle showed that the cipher had certain keys which can be easily discovered in a chosen plaintext attack.
- They used the fact that the use of multiplicative subkeys with the value of 1 or -1 gives rise to

linear factors in the round function.

- A linear factor is a linear equation in the key, input and output bits that hold for all possible input bits.
- The linear factors can be revealed by expressing the modulo 2 sum of LSBs of the output subblocks of an IDEA round in terms of inputs and key bits.

$$y_1 \approx y_2 = (X_1.Z_1)\ |_0 \approx 1 \approx x_3 \approx z_3$$

If Z_1=(-)1=0…01 (i.e if the 15 MSB bits of the Z_1 are 0), we have the following linear equation:

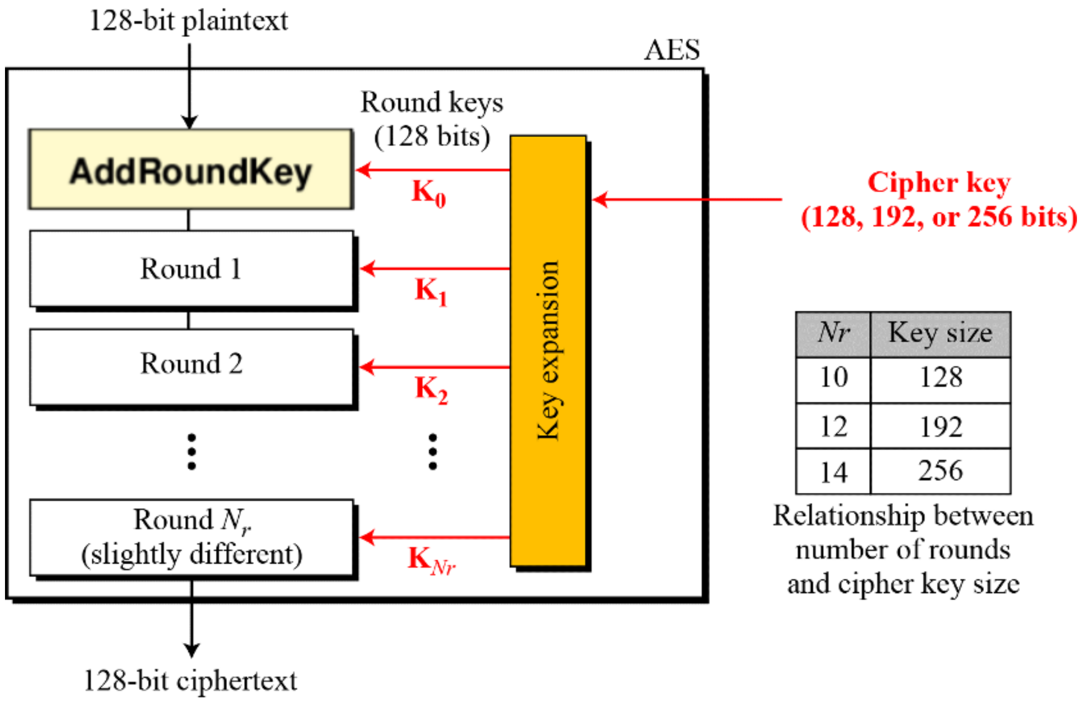
$$y_1 \approx y_2 = x_1 \approx x_3 \approx z_1 \approx z_3 \approx 1$$

Ch-7 – Advanced Encryption Standard (AES)

7.1 AES Algorithm

Video lecture references:
https://www.youtube.com/watch?v=X8whYEWoDSI&t=312s
https://www.youtube.com/watch?v=14SAn4IEAHk

- AES is cryptographic algorithm published by NIST (National Institute of Standards and Technology).
- The algorithm was proposed by Rijndael, it is also known as Rijndael algorithm.
- AES works on block cipher technique. Size of plaintext and ciphertext is same.
- The AES states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, 256 bits.
- Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES- 256 respectively.
- In the following figure, N_r defines the number of rounds. The *figure* also shows the relationship between the number of rounds and the key size.
- However, the rounds keys, which are created by the key-expansion algorithm are always 128-bits, the same size as the plaintext or ciphertext block.


128-bit plaintext
AES
Round keys (128 bits)
AddRoundKey
K_0
Cipher key (128, 192, or 256 bits)
Round 1
K_1
Round 2
K_2
Key expansion
Round N_r (slightly different)
K_{Nr}
128-bit ciphertext
Relationship between number of rounds and cipher key size

Nr	Key size
10	128
12	192
14	256

Figure: General design of AES encryption cipher.

- The number of keys generated by the key-expansion algorithm is always one more than the number of rounds. In other words, we have

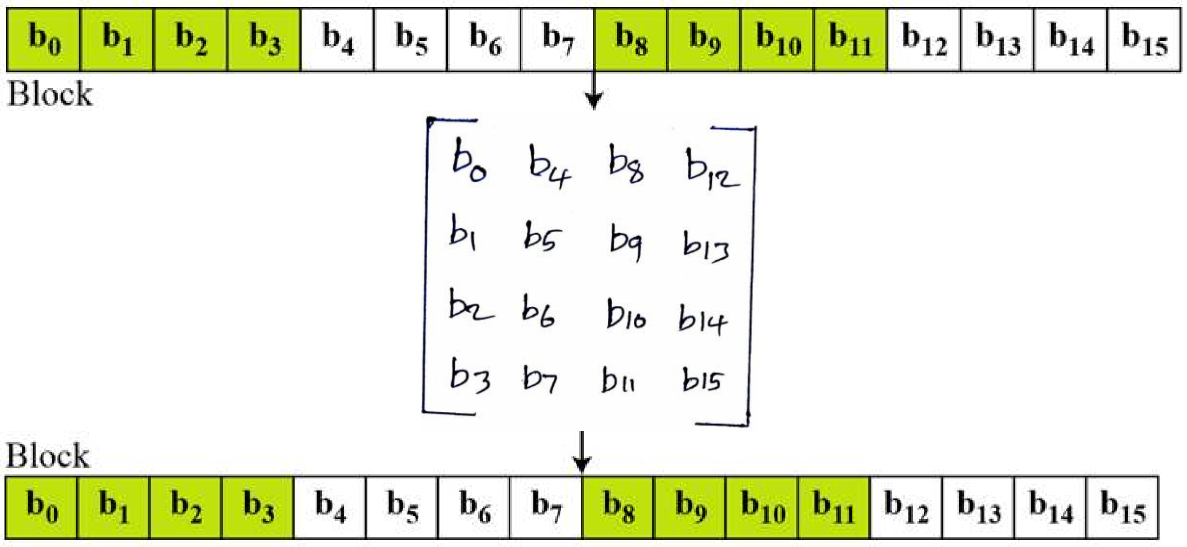
Number of round keys = number of rounds + 1

We refer round keys as K_0, K_1, K_2, ……., K_{Nr}

Plaintext transform in matrix form

- Take 128-bit block of plaintext that means 16-bytes and label them as b0 to b15 and convert into a 4 X 4 matrix of bytes; call it as **state array**.

- The first four bytes occupy first column and next four bytes occupy second column and so on.


Block
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15
b0 b4 b8 b12
b1 b5 b9 b13
b2 b6 b10 b14
b3 b7 b11 b15
Block
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15

So, AES operates 4 x 4 column-major order matrix of bytes called as state array.

EXAMPLE Let us see how a 16-character block can be shown as a 4 × 4 matrix. Assume that the text block is "AES uses a matrix". We add two bogus characters at the end to get "AESUSESAMATRIXZZ". Now we replace each character with an integer between 00 and 25. We then show each byte as an integer with two hexadecimal digits. For example, the character "S" is first changed to 18 and then written as 12_{16} in hexadecimal. The state matrix is then filled up, column by column, as shown in Fig. 7.4.

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

	DEC	HEX
A	00	00
B	01	01
:	:	:
:	:	:
Z	25	19

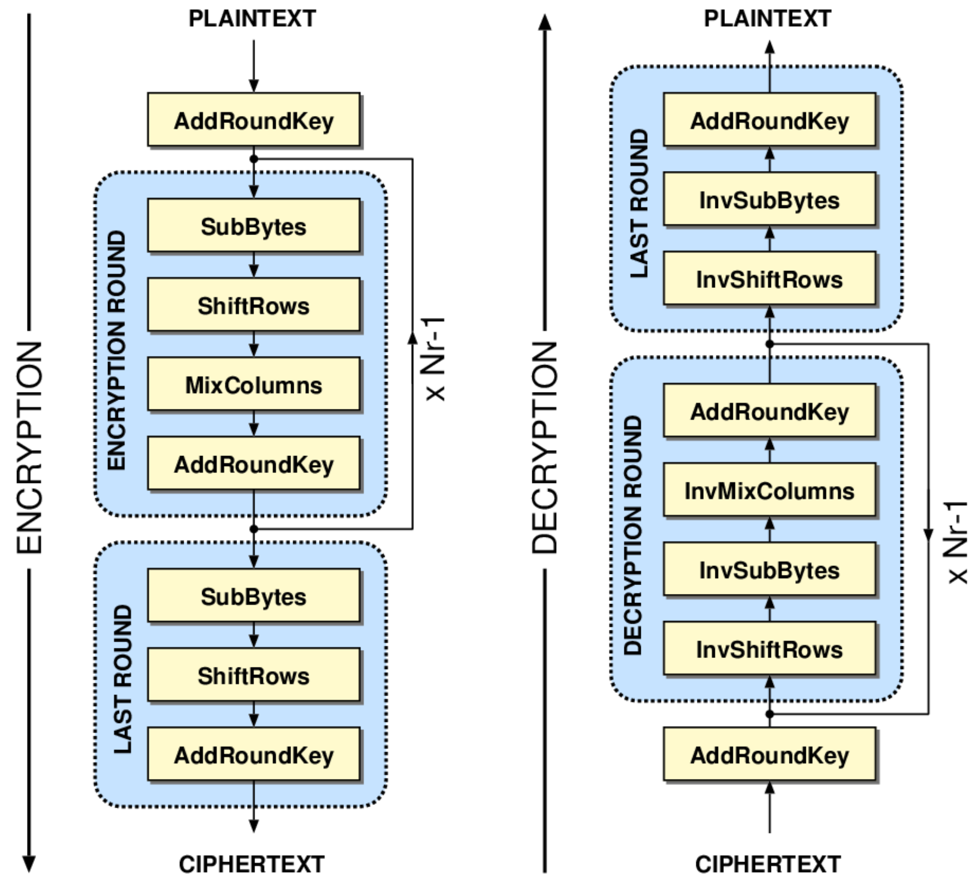
$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \text{State}$$

Fig *Changing ciphertext to state*

Overall structure of AES encryption process

- Overall structure of AES encryption process is shown in the below figure.
- For encryption, each round consists of the following steps:
  1) SubBytes
  2) ShiftRows
  3) MixColumns
  4) AddRoundKey

Above steps also called AES transformation function



1. **SubBytes / Substitution Bytes**

- AES defines a 16 X 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 left most 4 bits of the byte are used as a row value and the right most 4 bits used as a column value.
- These row and column values serve as indexes into the S-box to select a unique 8-bit output value.
- For example, the hexadecimal value 00 references row 0, column 0 of S-box which contains the value {63}, accordingly the value {12} is mapped into {C9}.

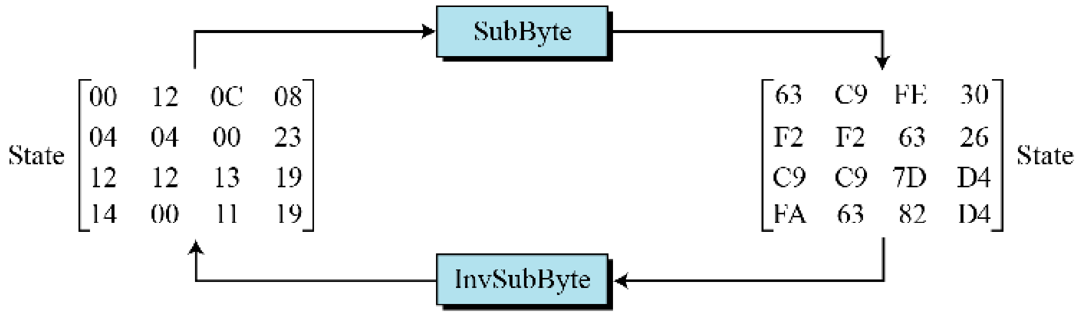


Figure: SubBytes transformation

	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

Figure: SubBytes transformation table

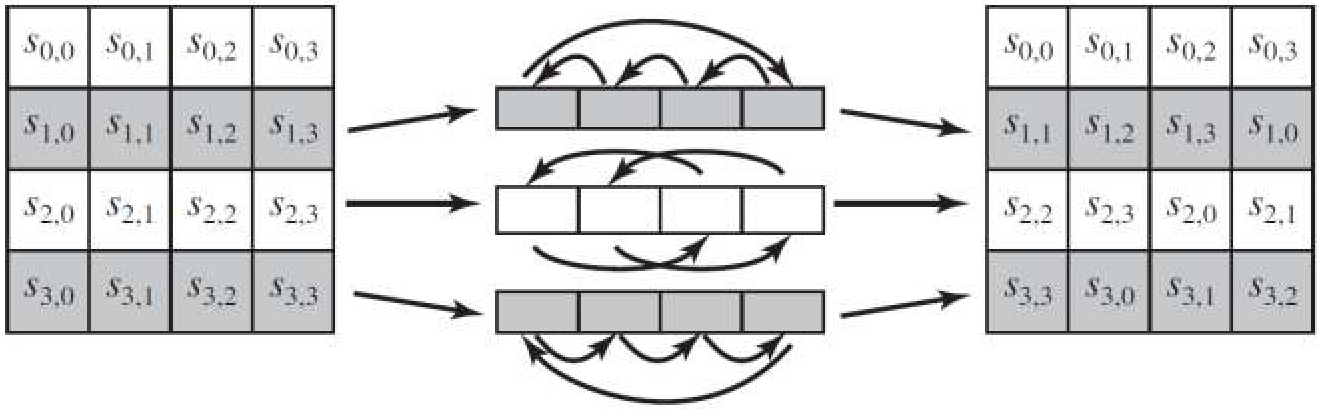
- Below table is the inverse S-Box, it will be using during decryption process.

	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

Figure: InvSubBytes transformation table.

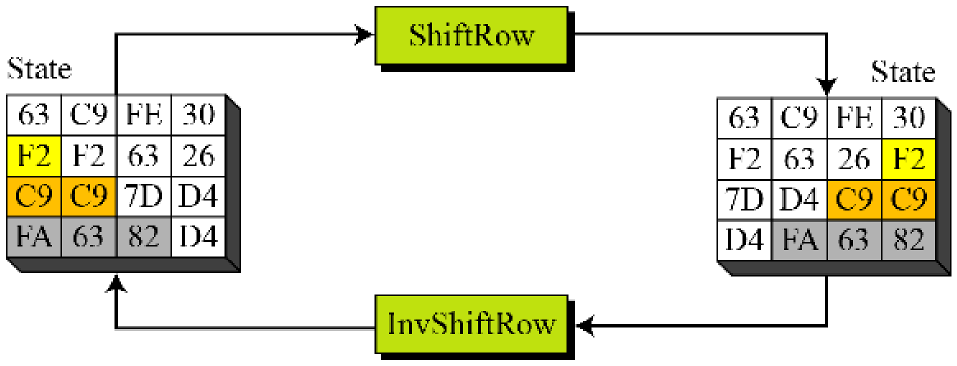
2. ShiftRows transformation

- The shift rows transformation is called ShiftRows.
- Rules of shifting rows,
  - Row 1 → now shifting
  - Row 2 → 1 byte left shift
  - Row 3 → 2 byte left shift
  - Row 4 → 3 byte left shift



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

- The inverse shift row transformation is called InverseShiftRows, performs the circular shift in the opposite direction for each of the last three rows, which a one-byte circular right shift for the second row and so on.



3. Mix Columns

- The mix column transformation is 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.

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

Predefine Matrix State Array New State Array

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

Example calculation is given below.

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

*

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

$\longrightarrow$

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

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

4. AddRoundKey

- In the forward add round key transformation, AddRoundKey, the 128-bits of State are bitwise XORed with the 128-bits of the round key.
- As shown in figure, the operation is viewed as a column wise operation between 4 bytes of a state column and one word of the round key; it can also be viewed as a **byte-level XOR operation**.

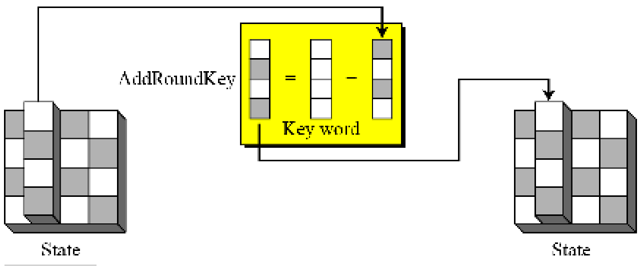


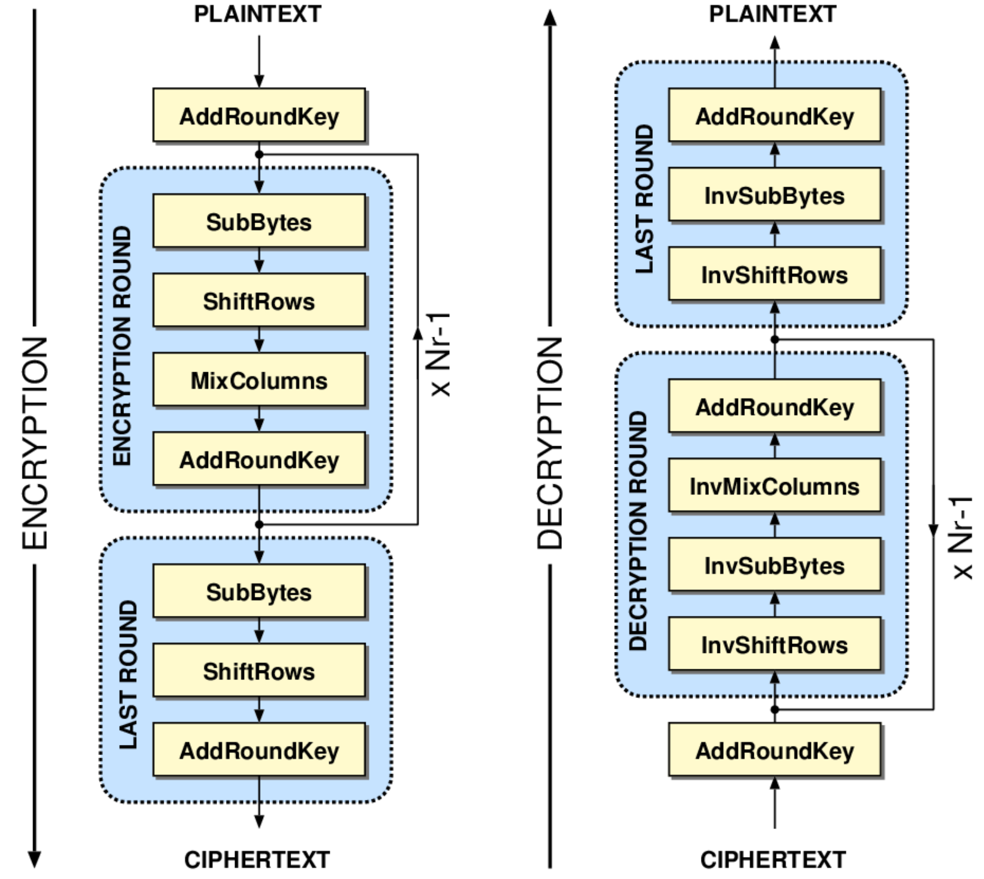
Figure: AddRoundKey transformation

Example:

$$\begin{bmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{bmatrix} \oplus \begin{bmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ F3 & 21 & 41 & 6A \end{bmatrix} = \begin{bmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D2 \end{bmatrix}$$

AES description process is shown in the following figure.

This is the inverse of encryption process. Ciphertext is input for the AddRoundKey and the output is given to the next round and in the same way it will go through N_{r-1} rounds and in the last round MixColumn part is missing same as the last round of encryption process and finally produces the plaintext.



7.2 AES Key Expansion

- 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 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.
- The below figure describes the generation of the expanded key, using the symbol *g* to represent that complex function.
- The first 4 words (W_0, W_1, W_2, W_3) are made from the cipher key. The cipher key is thought of as an array of 16 bytes (k_0 to K_{15}). The first four bytes (K_0 to K_3) become W_0; the next four bytes (k_4

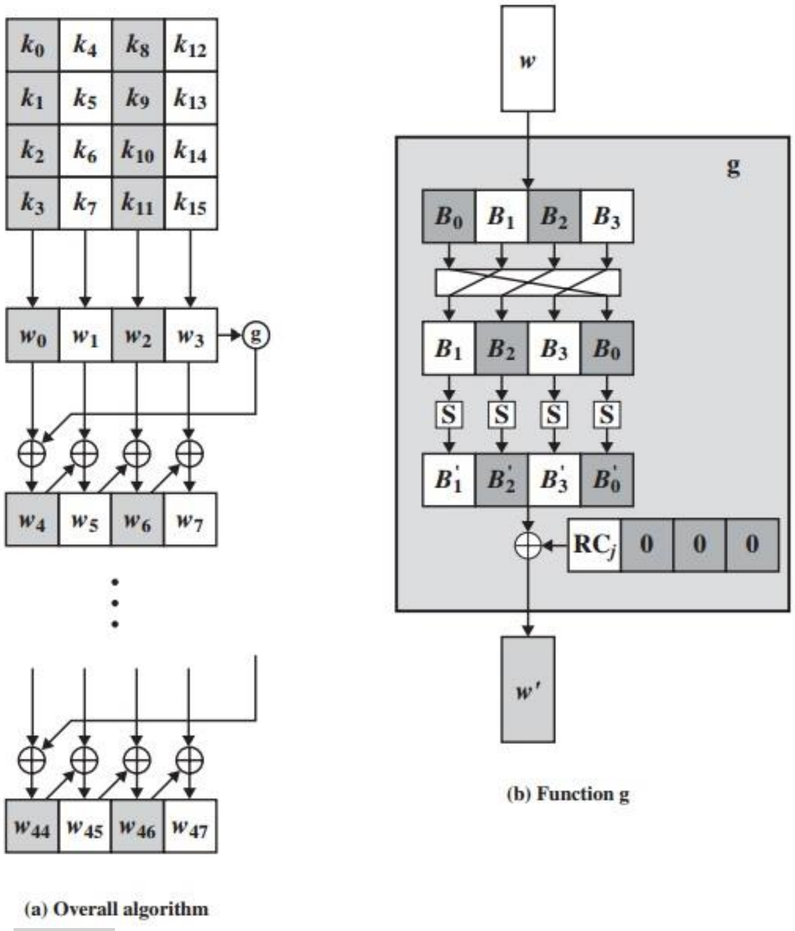
to k7) become W1; and so on. In other words, the concatenation of the words in this group replicates the cipher key.

- Words of each round are as follows

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

Figure: Words of each round

- The function *g* consists of the following sub functions.
  - This function performs the one byte circular left shift.
  - Using SubBytes transformation table (S box) each sub word performs a byte substitution.
  - Finally the result is XORed with the round constant called as Rcon[j].
  - The round constant is a word in which the three rightmost bytes are always 0.
  - The table below shows the values for AES-128 version (with 10 rounds).



k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w_{44} w_{45} w_{46} w_{47}
(a) Overall algorithm
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B'_1 B'_2 B'_3 B'_0
RC_j 0 0 0
w'
(b) Function g

Figure: AES Key Expansion

Round	Constant (RCon)	Round	Constant (RCon)
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

Table: Rcon Constants

Algorithm
Algorithm below is a simple algorithm for the key-expansion routine (version AES-128).

```
KeyExpansion ([key_0 to key_15], [w_0 to w_43])
{
    for (i = 0 to 3)
        w_i ← key_{4i} + key_{4i+1} + key_{4i+2} + key_{4i+3}

    for (i = 4 to 43)
    {
        if (i mod 4 ≠ 0)    w_i ← w_{i−1} + w_{i−4}
        else
        {
            t ← SubWord (RotWord (w_{i−1})) ⊕ RCon_{i/4}        // t is a temporary word
            w_i ← t + w_{i−4}
        }
    }
}
```

Algorithm: Pseudocode for Key expansion in AES-128

7.3 Analysis of AES (or) Three Characteristics of AES

Following is a brief review of three characteristics of AES.

Security
AES was designed after DES. Most of the known attacks on DES were already tested on AES; none of them has broken the security of AES so far.

- **Brute-Force Attack:** AES is definitely more secure than DES due to the larger-size key (128, 192, and 256 bits). Let us compare DES with 56-bitcipher key and AES with 128-bit cipher key. For DES we need 256 tests to find the key; for AES we need 2128 tests to find the key. This means that if we can break DES in t seconds, we need (272 x t) seconds to bread AES. This would be almost impossible. In addition, AES provides two other versions with longer cipher keys. The lack of weak keys is another advantage of AES over DES.

- **Statistical attacks** the strong diffusion and confusion provided by the combination of the SubBytes, ShiftRows, and MixColumns transformations removes any frequency pattern in the plaintext. Numerous tests have failed to do statistical analysis of the ciphertext.

Implementation

- AES can be implemented in software, hardware, and firmware. The implementation can use table lookup process or routines that use a well-defined algebraic structure. The transformation can be either *byte-oriented* or *word-oriented*. In the byte-oriented version, the whole algorithm can use an 8-bit processor; in the word-oriented version, it can use a 32-bit processor. In either case, the design of constants makes procession very fast.

Simplicity and Cost

- The AES algorithms are such simple that even cheap processors can implement them. They also make use of less memory.