CS8792 - CNS

UNIT- II

SYMMETRIC KEY CRYPTOGRAPHY

MATHEMATICS OF SYMMETRIC KEY CRYPTOGRAPHY : Algebraic structures - Modular arithmetic - Euclid's Algorithm - Congruence and Matrices - Groups, Rings, Fields - Finite fields - SYMMETRIC KEY CIPHERS : SDES - Block cipher Principles of DES - Strength of DES - Differential and linear cryptanalysis - Block cipher design Principles - Block cipher mode of operation - Evaluation criteria for AES - Advanced Encryption Standard - RC4 - Key distribution.

2.1 MATHEMATICS OF SYMMETRIC KEY CRYPTOGRAPHY :

ALGEBRAIC STRUCTURES

* Symmetric ciphers use symmetric algorithms to encrypt and decrypt.
  → These ciphers are used in symmetric key cryptography.
* A symmetric algorithm uses the same key to encrypt data as it does to decrypt data.

Advantage:
→ speed

Disadvantages
→ Lack in security and key management.

Algebraic Structures.

* 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 the elements of the set is called an algebraic structure.

Common algebraic structures
- Groups
- Rings
- Fields.

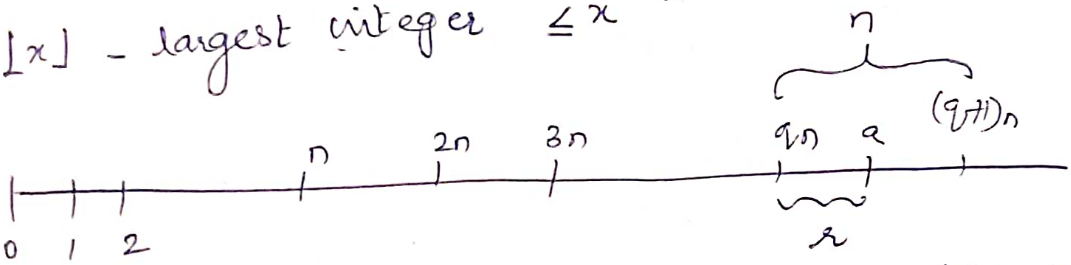
2.2 MODULAR ARITHMETIC

* Given any positive integer n and any nonnegative integer a, if we divide a by n, we get an integer quotient q and an integer r

Relationship :

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

$\lfloor x \rfloor$ - largest integer $\le x$



* Given a and positive n, to find q & r satisfy the relationship
→ distance from qn to a is r
→ Remainder r is <u>residue</u>

$a = 11 \quad n = 7 \quad 11 = 1 \times 7 + 4 \quad r = 4, q = 1$

$a = -11 \quad n = 7 \quad -11 = (-2) \times 7 + 3 \quad r = 3 \; q = -2$

* a is an integer, n is a positive integer
define $a \bmod n$ - remainder
a/n
n → modulus

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

$11 \bmod 7 = 4$, $-11 \bmod 7 = 3$

* Two integers a & b → congruent modulo n

$$(a \bmod n) = (b \bmod n)$$

written as $a \equiv b \pmod n$

$73 \equiv 4 \pmod{23}$

$21 \equiv -9 \pmod{10}$

Divisors

* A nonzero b divides a if $a = mb$

$a, b, m \rightarrow$ integers

* b divides a if there is no remainder on division.
* $b|a$ is used to mean b divides a.

$b|a \rightarrow b$ is a divisor of a

Ex: The positive divisors of 24 are 1, 2, 3, 4, 6, 8, 12, 24

relations

* If $a|1$, then $a = \pm 1$
* If $a|b$, & $b|a$ then $a = \pm b$
* Any $b \neq 0$ divides 0
* If $b|g$ & $b|h$, then $b|(mg + nh)$, integers m & n

– If $b|g$, then g is of the form $g = b \times g_1$
If $b|h$, then h is of the form $h = b \times h_1$

So
$$mg + nh = mbg_1 + nbh_1 = b \times (mg_1 + nh_1)$$

b divides $mg + nh$.

$b = 7$ $\quad g = 14$ $\quad h = 63$ $\quad m = 3$ $\quad n = 2$

$7|14$ $\quad 7|63$ $\quad$ To show: $7|(3 \times 14 + 2 \times 63)$

$$(3 \times 14 + 2 \times 63) = 7(3 \times 2 + 2 \times 9)$$
$$7|(7(3 \times 2 + 2 \times 9))$$

if $a \equiv 0 \pmod{n}$, then n/a.

2.3 THE EUCLIDEAN ALGORITHM

* To determine the greatest common divisor of two positive integers.

Greatest Common Divisor:

* Nonzero b is defined to be a divisor of a if

$a = mb.$

Notation: $\gcd(a, b)$

greatest common divisor of a & b.

* The positive integer c is said to be the greatest common divisor of a & b if

1. c is a divisor of a and of b
2. any divisor of a and b is a divisor of c.

Equivalent definition:

$$\gcd(a,b) = \max[k, \text{ such that } k|a \text{ \& } k|b]$$

$$\gcd(a,b) = \gcd(a,-b) = \gcd(-a,b) = \gcd(-a,-b)$$

$$\gcd(a,b) = \gcd(|a|, |b|)$$

$$\gcd(60,24) = \gcd(60,-24) = 12$$

* Two integers a & b are relatively prime if their only common positive integer factor is 1.

a, b are relatively prime, if $\gcd(a,b) = 1$

positive divisors of 8 = 1, 2, 4, 8
positive divisors of 15 = 1, 3, 5, 15

1 is the only integer on both lists.

Finding the Greatest Common Divisor:

Theorem

For any nonnegative integer a and any positive integer b.

$$\boxed{\gcd(a,b) = \gcd(b, a \bmod b)}$$

Ex:

$\gcd(55,22) = \gcd(22, 55 \bmod 22)$

$= \gcd(22, 11)$

$= 11$

* By the definition of gcd,

$d | a, \; d | b$

* For any positive integer b, a can be expressed in the form

$a = kb + r$

$= r (\bmod b)$

$a \bmod b = r$

* To determine the greatest common divisor.

$\gcd(18,12) = \gcd(12,6) = \gcd(6,0) = 6$

$\gcd(11,10) = \gcd(10,1) = \gcd(1,0) = 1$

Algorithm:

EUCLID (a, b)

1. $A \leftarrow a; \; B \leftarrow b$
2. if $B = 0$ return $A = \gcd(a,b)$
3. $R = A \bmod B$
4. $A \leftarrow B$
5. $B \leftarrow R$
6. goto 2

Ex: To find $\gcd(1970, 1066)$

Ans: 2

2.5 GROUPS, RINGS, AND FIELDS.

* Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra.
* Two elements of the set can be combined to obtain a third element of the set.

→ By convention, the notation for the two principal classes of operations on set elements is usually the same as the notation for addition and multiplication on ordinary numbers.

Groups:

A group G, sometimes denoted by $\{G, \cdot\}$

A group G, sometimes denoted by a binary operation, denoted by $\cdot$

- Set of elements with a binary operation, denoted by $\cdot$
- associates to each ordered pair (a, b) of elements in G an element $(a \cdot b)$ in G

Axioms

(A1) **Closure:** If a and b belong to G, then $a \cdot b$ is also in G.

(A2) **Associative:** $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b, c in G

(A3) **Identity element:** an element e in G

$a \cdot e = e \cdot a = a$ for all a in G

(A4) **Inverse element:** each a in G, an element a' in G

$a \cdot a' = a' \cdot a = e$

Finite group:

* A group has a finite number of elements

Order of the group:

* The number of elements in the group.

infinite group:

* Not finite group.

Abelian: A group is said to be abelian if it satisfies the additional condition.

(A5) Commutative:

$$a \cdot b = b \cdot a \text{ for all } a, b \text{ in } G$$

* When the group operation is addition, the identity element is 0.

* inverse element of a is $-a$

* subtraction:

$$a - b = a + (-b)$$

* Cyclic Group:

* A group G is cyclic if every element of G is a power a^k of a fixed element $a \in G$.

K - integer.

generate:

- The element a is 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

* The additive group of integers is an infinite cyclic group generated by the element 1.

- powers are interpreted additively, so that n is the nth power of 1

Rings:

* A ring R, denoted by $\{R, +, \times\}$, is a set of elements with two binary operations, called addition and multiplication.

Axioms:

(A1 - A5) : R is an abelian group with respect to addition

(M1) Closure under multiplication : $a, b \in R$
$ab \in R$

(M2) Associativity of multiplication : $a(bc) = (ab)c$

(M3) Distributive laws : $a(b+c) = ab + ac$
$(a+b)c = ac + bc$

(M4) Commutativity of multiplication : $ab = ba$

integral domain

(M5) Multiplicative identity : element 1 in R
$a1 = 1a = a$

(M6) No zero divisors : $ab = 0$, $a = 0$ or $b = 0$.

Fields:

* A field F, denoted by $\{F, +, \times\}$, is a set of elements with two binary operations, called addition & multiplication

Axioms:

* (A1 — M6)
* F is an integral domain.
* F satisfies axioms A1 through A5
  ie) and M1 through M6

(M7) Multiplicative inverse:

* For each a in F, except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$

* A field is a set in which we can do addition, subtraction, multiplication and division without leaving the set.

Division — rule:

$$a/b = a(b^{-1})$$

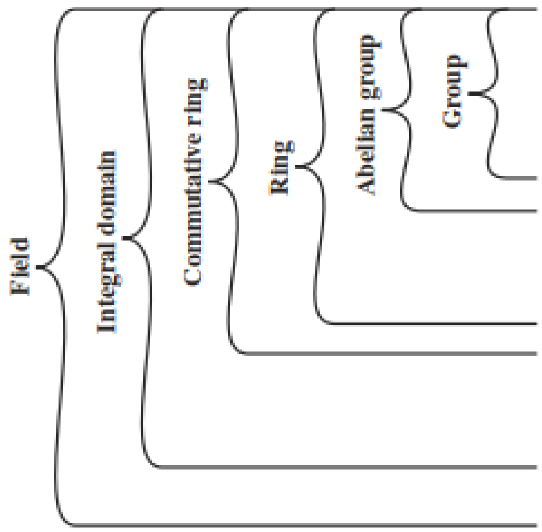
Ex.

- rational numbers
- real numbers
- complex numbers.

Not a field → set of all integers

not every element of the set has a multiplicative inverses.

Only -1 & 1 → have multiplicative inverse.



(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 4.2 Groups, Ring, and Field

2.6 FINITE FIELDS

* The finite field of order p^n is written $GF(p^n)$

GF → Galois Field.

$n=1$, finite field $GF(p)$

$n>1$, finite field has a different structure than that for finite fields

Finite Fields of Order p:

* For a given prime p, the finite field of order p, $GF(p)$ is defined as the set Z_p of integers $\{0,1,\dots p-1\}$, together with the arithmetic operations modulo p.

* The set Z_n of integers $\{0,1,\dots,n-1\}$ together with the arithmetic operations modulo n, is a commutative ring.

Properties:

Commutative laws

Associative laws

Distributive laws

Identities

Additive Inverse $(-w)$

Multiplicative Inverse (w^{-1}) — For each $w \in Z_p$, $w \neq 0$, there exists a $z \in Z_p$ such that $w \times z \equiv 1 \bmod p$.

if $(a \times b) \equiv (a \times c) \bmod p$ then $b \equiv c \bmod p$.

Simplest finite field is $GF(2)$

Arithmetic operations:

Addition

+	0	1
0	0	1
1	1	0

Multiplication

×	0	1
0	0	0
1	0	1

Inverses

w	$-w$	w^{-1}
0	0	—
1	1	1

Table 4.5 Arithmetic in GF(7)

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

(a) Addition modulo 7

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

(b) Multiplication modulo 7

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

(c) Additive and multiplicative inverses modulo 7

- If a and b are relatively prime, then b has a multiplicative inverse modulo a. That is, if gcd(a, b) = 1, then b has a multiplicative inverse modulo a.
- That is, for positive integer $b < a$, there exists a $b^{-1} < a$ such that $bb^{-1} = 1 \bmod a$. If a is a prime number and $b < a$, then clearly a and b are relatively prime and have a greatest common divisor of 1.
- We now show that we can easily compute b^{-1} using the extended Euclidean algorithm.

$$ax + by = d = \gcd(a, b)$$

- Now, if gcd(a, b) = 1, then we have $ax + by = 1$. Using the basic equalities of modular arithmetic, we can say

$$[(ax \bmod a) + (by \bmod a)] \bmod a = 1 \bmod a$$

$$0 + (by \bmod a) = 1$$

- But if $by \bmod a = 1$, then $y = b^{-1}$. Thus, applying the extended Euclidean algorithm to above Equation, it yields the value of the multiplicative inverse of b if gcd(a, b) = 1.

- Consider the example. Here we have $a = 1759$, which is a prime number, and $b = 550$. The solution of the equation $1759x + 550y = d$ yields a value of $y = 355$. Thus, $b^{-1} = 355$.
- More generally, the extended Euclidean algorithm can be used to find a multiplicative inverse in Z_n for any n. If we apply the extended Euclidean algorithm to the equation $nx + by = d$, and the algorithm yields $d = 1$, then $y = b^{-1}$ in Z_n.

2.7 SYMMETRIC KEY CIPHERS

Simplified Model of Conventional Encryption

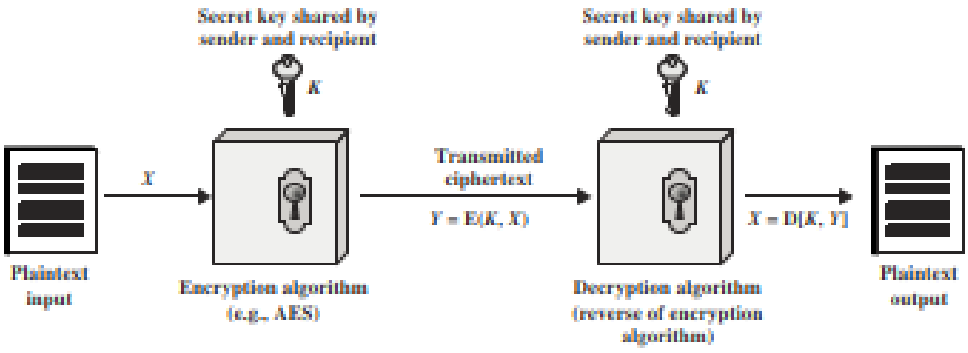


Figure 2.1 Simplified Model of Symmetric Encryption

* A symmetric encryption scheme has five ingredients.

i) PlainText:
- original intelligible message or data
- fed into the algorithm as input

ii) Encryption Algorithm:
- performs various substitutions and transformations on the plaintext.

iii) Secret Key:
- input to encryption algorithm.
- a value independent of plaintext.

iv) Ciphertext:
- scrambled message produced as output
- depends on the plaintext and the secret key.
- different keys will produce different ciphertexts

v) Decryption Algorithm:
- encryption algorithm run in reverse
- takes the ciphertext and the secret key and produce the original plaintext.

Two requirements for secure use of conventional encryption.

i) Need a strong encryption algorithm

ii) Sender and receiver must have copies of the secret key in a secure fashion and must keep the key secure.

We assume that it is impractical to decrypt a message on the basis of the ciphertext *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret, we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 2.2. A source produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K_1, K_2, \ldots, K_J]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

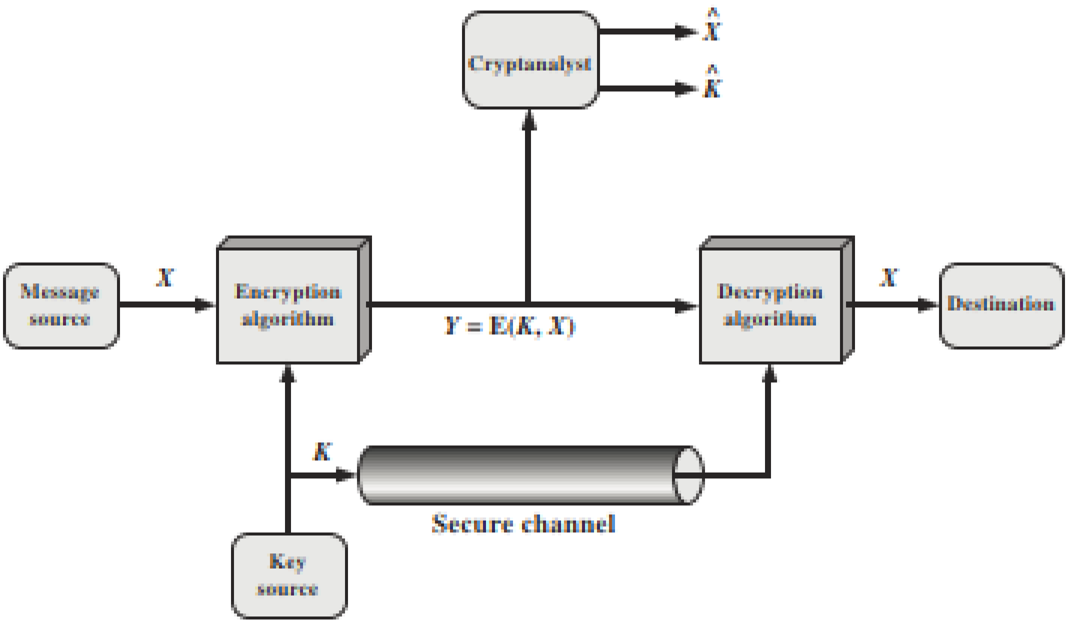


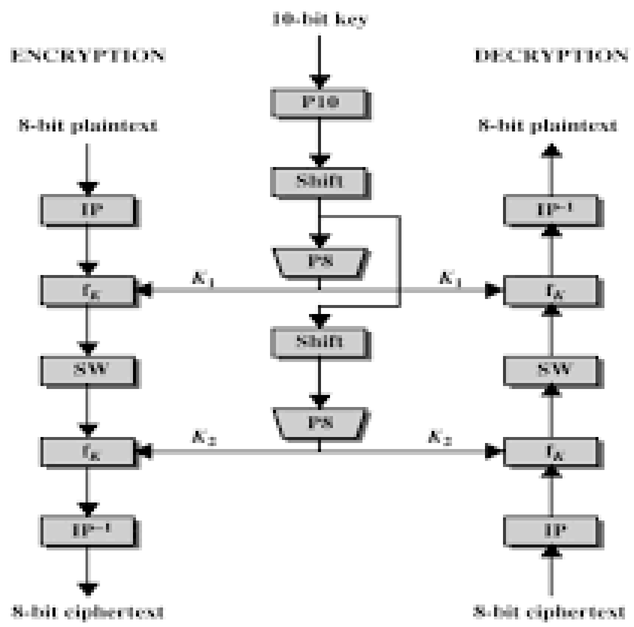
Figure 2.2 Model of Symmetric Cryptosystem

SDES

* developed by Professor Edward Schaefer

→ Overview
→ S-DES Key Generation
→ S-DES Encryption
  - Initial and Final Permutations
  - The function f_K
  - The Switch Function.
→ Analysis of Simplified DES.
→ Relationship to DES.

1. Overview



* S-DES encryption Algorithm:
  - Takes an 8-bit block of plaint text and a 10-bit key as input and produces an 8-bit block of ciphertext as output

* 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.

i) IP - an initial Permutation.

ii) f_K - a complex function.
- involves both permutation and substitution operations and depends on a key input

iii) SW - a simple permutation function
- switches the two halves of the data.

iv) f_K.

v) IP^{-1} - permutation function that is the inverse of the initial permutation.

* increases the difficulty of cryptanalysis.

Express the encryption Algorithm.

- Composition of functions.

$$\boxed{\text{ciphertext} = IP^{-1}(f_{K_2}(SW(f_{K_1}(IP(\text{plaintext})))))}$$

$$K_1 = P8(\text{shift}(P10(\text{Key})))$$
$$K_2 = P8(\text{shift}(\text{shift}(P10(\text{Key}))))$$

Decryption:

reverse the encryption

$$\boxed{\text{plaintext} = IP^{-1}(f_{K_1}(SW(f_{K_2}(IP(\text{ciphertext})))))}$$

2 S-DES Key Generation:

* S-DES depends on the use of a 10-bit key shared between sender and receiver.

* From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption Algorithm.

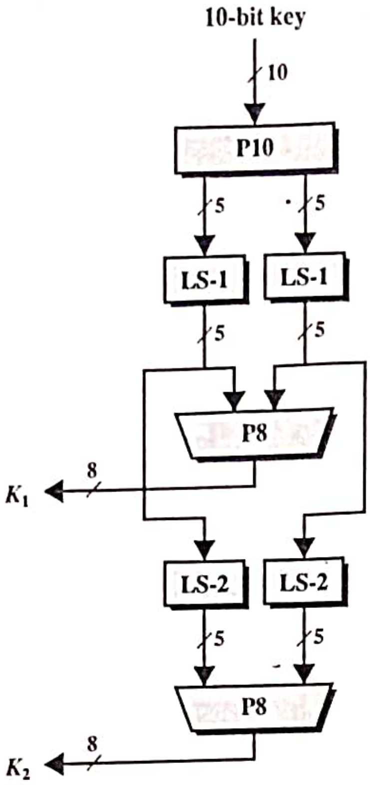


Figure 3.2 Key Generation for Simplified DES

* First, permute the key.

10-bit key.
$(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10})$

Permutation P10

$P10(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}) = (K_3, K_5, K_2, K_7, K_4, K_{10}, K_1, K_9, K_8, K_6)$

P10									
3	5	2	7	4	10	1	9	8	6

Ex: 1 0 1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 1 0 0

* Next, perform a circular left shift (LS-1) or rotation, separately on the first five bits and the second five bits.

1 0 0 0 0 0 1 1 0 0

0 0 0 0 1 1 1 0 0 0

* Next, Apply P8.
  → picks out and permutes 8 of the 10 bits.

Rule.

P8							
6	3	7	4	8	5	10	9

* Subkey 1 (K1)

$\boxed{10100100}$

* Back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string.

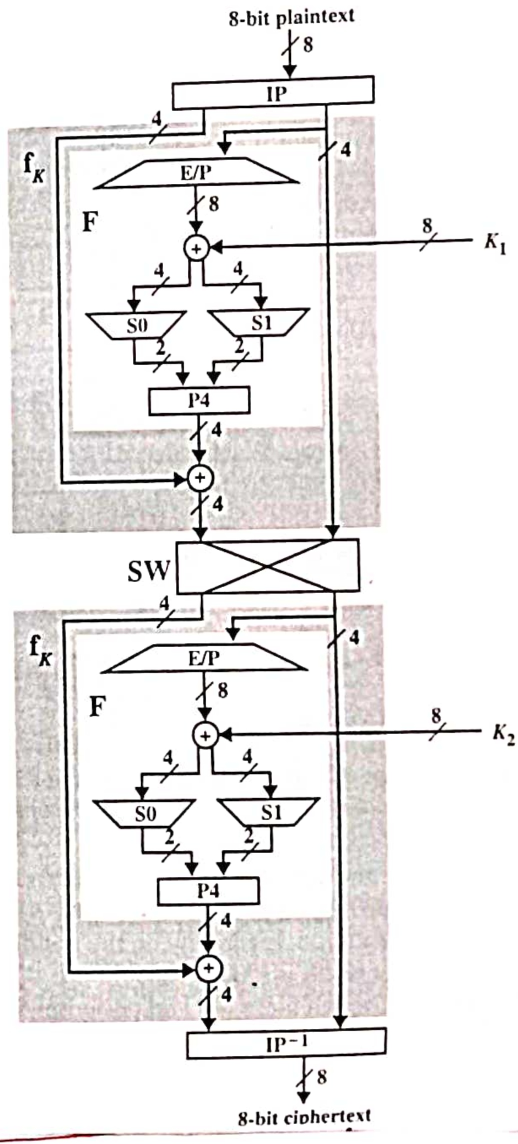
$\underbrace{00001}_{\downarrow}$ $\underbrace{11000}_{\downarrow}$

00100 00011

* P8 is applied again to produce K_2.

$\boxed{01000011}$

3 S-DES Encryption:



Initial and Final Permutation:

* The input to the algorithm is an 8-bit block of plaintext

→ 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.

* At the end of the algorithm, the inverse permutation is used.

IP^{-1}							
4	1	3	5	7	2	8	6

$$IP^{-1}(IP(x)) = x.$$

The Function f_K

* Combination of permutation and substitution functions.

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

L, R → Leftmost 4 bits and rightmost 4 bits of the 8-bit input to f_K.

F → mapping from 4-bit strings to 4-bit strings

SK → subkey.

$\oplus$ → bit-by-bit exclusive-OR function.

Ex: output of the IP stage is 10111101

F(1101, SK) = (1110) for some key SK

$L \oplus F(1101, SK) = (1011) \oplus (1110)$

$= (0101)$

$$\begin{array}{r} 1011 \\ 1110 \\ \hline 0101 \end{array}$$

$$\boxed{f_K(10111101) = (01011101)}$$

Mapping F :

* The input is a 4-bit number $(n_1 n_2 n_3 n_4)$
  → First operation is an expansion/permutation operation.

E/P							
4	1	2	3	2	3	4	1

Depict the result:

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

8-bit subkey $K_1 = (K_{11}, K_{12}, K_{13}, K_{14}, K_{15}, K_{16}, K_{17}, K_{18})$ is added using exclusive-OR

$$\Rightarrow \begin{array}{c|c} n_4 + K_{11} & n_1 + K_{12} \\ n_2 + K_{15} & n_3 + K_{16} \end{array} \quad \begin{array}{c|c} n_2 + K_{13} & n_3 + K_{14} \\ n_4 + K_{17} & n_1 + K_{18} \end{array}$$

⇓

Rename 8-bits

$$\begin{array}{c|c} P_{0,0} & P_{0,1} \\ P_{1,0} & P_{1,1} \end{array} \quad \begin{array}{c|c} P_{0,2} & P_{0,3} \\ P_{1,2} & P_{1,3} \end{array}$$

* The first 4 bits (first row) are fed into the S-box S_0 to produce a 2-bit output
- The remaining 4 bits (second row) are fed into S_1 to produce another 2-bit output.

Two boxes are defined

$$S_0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

Operation of S-boxes :

* The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box.
* the second and third bits specify a column of the S-box.

* The entry in that row and column, in base 2, is the 2-bit output.

Ex

if $(P_{0,0}\ P_{0,3}) = (00)$

$(P_{0,1}\ P_{0,2}) = (10)$

then output is from row 0, column 2 of S_0, which is 3 or (11) in binary.

* $(P_{1,0}\ P_{1,3})$ & $(P_{1,1}\ P_{1,2})$ are used to index into a row and column of S_1 to produce an additional 2 bits.

* Next, the 4 bits produced by S_0 and S_1 undergo a further permutation

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.

E/P, S0, S1, P4 functions are the same.

- The key input is K_2.

4. Analysis of Simplified DES:

Brute-force attack:

* feasible
* with 10 bit key, $2^{10} = 1024$ possibilities
* An attacker can try each possibility and analyze the result.

Cryptanalysis:

* Known plaintext attack.

P.T $(P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8)$ } are known.
C.T $(C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8)$ }

Key $(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10})$ — unknown.

permutation & addition → linear mapping.
S-boxes → nonlinearity.

Operation of S_0: 4 bit o/p (q, r, s, t)

$q = abcd + ab + ac + b + d$

$r = abcd + abd + ab + ac + ad + a + c + 1$

additions are modulo 2.

* Very complex polynomial expressions for C.T
– making cryptanalysis difficult.

5. Relationship to DES:

* DES operates on 64-bit blocks of input.
* Encryption scheme
→ $IP^{-1} \circ f_{K_{16}} \circ SW \circ f_{K_{15}} \circ SW \circ \dots \circ SW \circ f_{K_1} \circ IP$

56-bit key
→ Sixteen 48-bit subkeys are calculated.

* Initial permutation of 56 bits followed by a sequence of shifts and permutations of 48 bits.

$F = 32$ bits $(n_1, \dots n_{32})$

* After the initial expansion/permutation, the output of 48 bits can be diagrammed as

n_{32}	n_1	n_2	n_3	n_4	n_5
n_4	n_5	n_6	n_7	n_8	n_9
⋮		⋮			⋮
n_{28}	n_{29}	n_{30}	n_{31}	n_{32}	n_1

→ Matrix is added (EX-OR) to a 48-bit subkey.

→ 8 rows ⇒ 8 S-boxes.
↓
4 rows & 16 columns

1^{st} & last bit ⇒ row of S-box
middle 4 bits ⇒ column

2.8 BLOCK CIPHER PRINCIPLES OF DES

* All symmetric block encryption algorithms are based on a structure referred to as a Feistal Block Cipher.

→ Stream ciphers and Block Ciphers.
→ Motivation for the Feistal Cipher Structure
→ The Feistal Cipher
  - Diffusion and confusion
  - Feistal Cipher Structure
  - Feistal Decryption Algorithm
→ The Data Encryption Algorithm
  - DES Encryption
    - Initial Permutation
    - Details of Single Round
    - Key Generation
  - DES Decryption
  - The Avalanche Effect

Stream Ciphers and Block ciphers:

Stream ciphers:

* Encrypts a digital data stream one bit or one byte at a time.

EX: - autokeyed Vigenere cipher
- Vernam Cipher.

Block cipher:

* A block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.

- block size of 64 or 128 bits is used.

EX: Network-based symmetric cryptographic applications.

Motivation for the Feistal Cipher Structure:

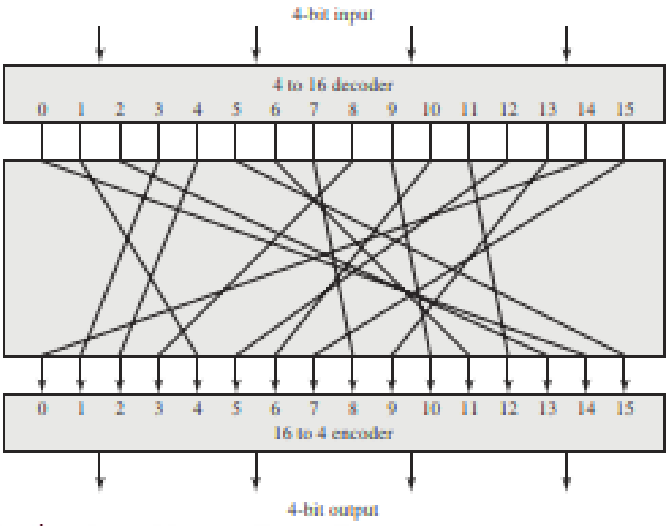
* A block cipher operates on a plaintext block of n bits to produce a ciphertext block of n bits.
  - 2^n possible different plaintext blocks
  - for the encryption to be reversible.
    - each must produce a unique ciphertext block.
  - such a transformation is called reversible or nonsingular.

Ex: Nonsingular and Singular transformation for $n=2$

Reversible Mapping		Irreversible Mapping	
Plaintext	Ciphertext	Plaintext	Ciphertext
00	11	00	11
01	10	01	10
10	00	10	01
11	01	11	01

* Number of different transformations is $\boxed{2^n!}$

General n-bit-n-bit Block Substitution $(n=4)$



* A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique one of 16 possible outputs states, each of which is represented by 4 ciphertext bits.

Table 3.1 Encryption and Decryption Tables for Substitution Cipher of Figure 3.2

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

The Feistel Cipher

- Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher.
- In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:

Substitution: Each plaintext element or group of elements is uniquely replaced by a corresponding cipher text element or group of elements.

Permutation: A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions

DIFFUSION AND CONFUSION

- The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture shannon to capture the two basic building blocks for any cryptographic system.
- Shannon's concern was to prevent cryptanalysis based on statistical analysis . Assume the attacker has some knowledge of the statistical characteristics of the plaintext.
- In **diffusion**, the statistical structure of the plaintext is degenerated into long-range statistics of the ciphertext.
- This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits.
- An example of diffusion is to encrypt a message $M = m1, m2, m3, \ldots$ of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter y_n.

- **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to prevent attempts to discover the key.
- Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so

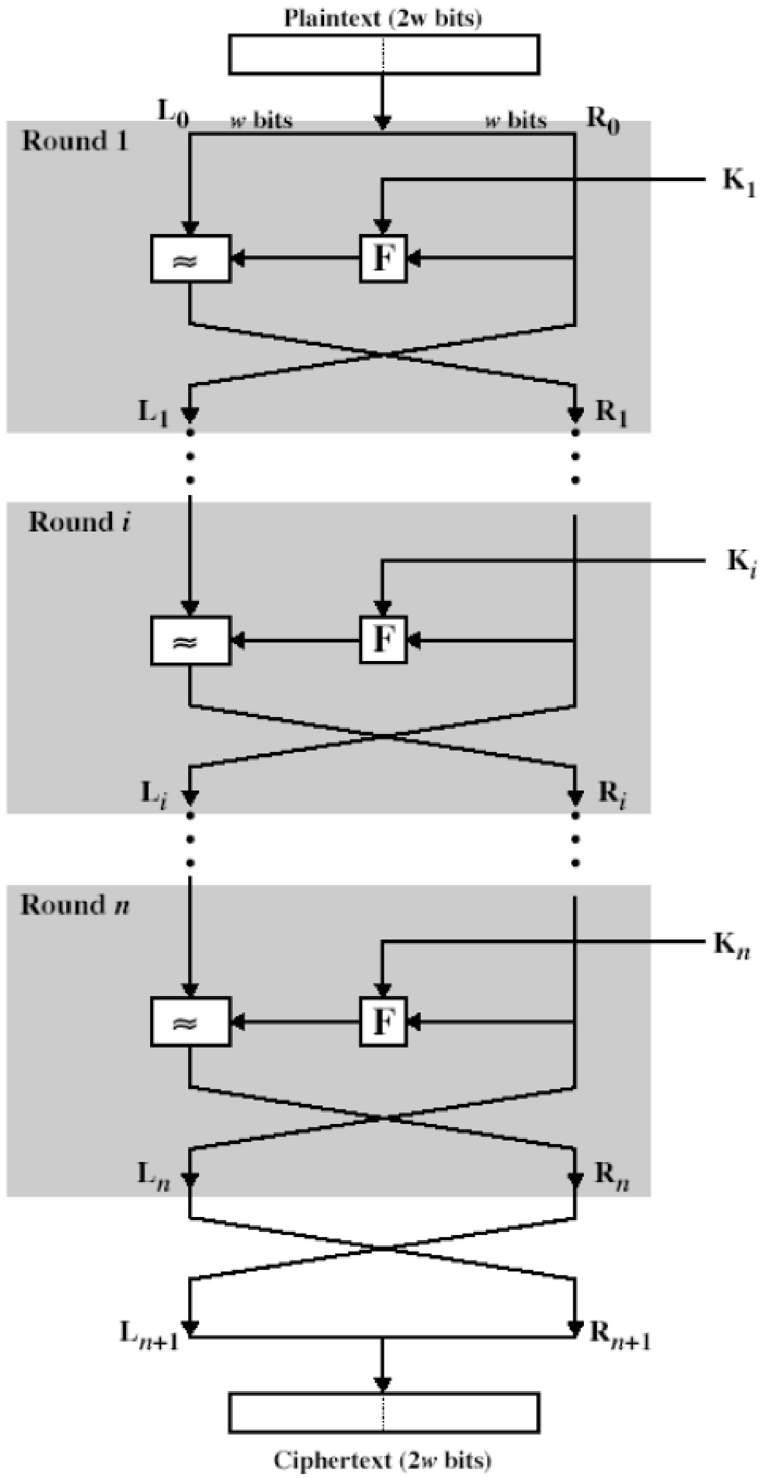
complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm.

FEISTEL CIPHER STRUCTURE

- The Figure depicts the structure proposed by Feistel . The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K. The plaintext block is divided into two halves, L_0 and R_0 .
- The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs L_{i-1} and R_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K.
- In Figure, 16 rounds are used, although any number of rounds could be implemented. All rounds have the same structure. A **substitution** is performed on the left half of the data.
- This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data.
- The round function has the same general structure for each round but is parameterized by the round subkey K_i. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.
- This structure is a particular form of the substitution-permutation network.

Feistel network depends on the choice of the following parameters and design features:

Block size: Larger block sizes mean greater security but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. A block size of 64 bits has been considered . However, the new AES uses a 128-bit block size.



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

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

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

Round function F: Again, greater complexity generally means greater resistance to cryptanalysis. There are two other considerations in the design of a Feistel cipher:

Fast software encryption/decryption: In many cases, encryption is embedded in applications or utility functions to prevent a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

Ease of analysis: Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength.

FEISTEL DECRYPTION ALGORITHM

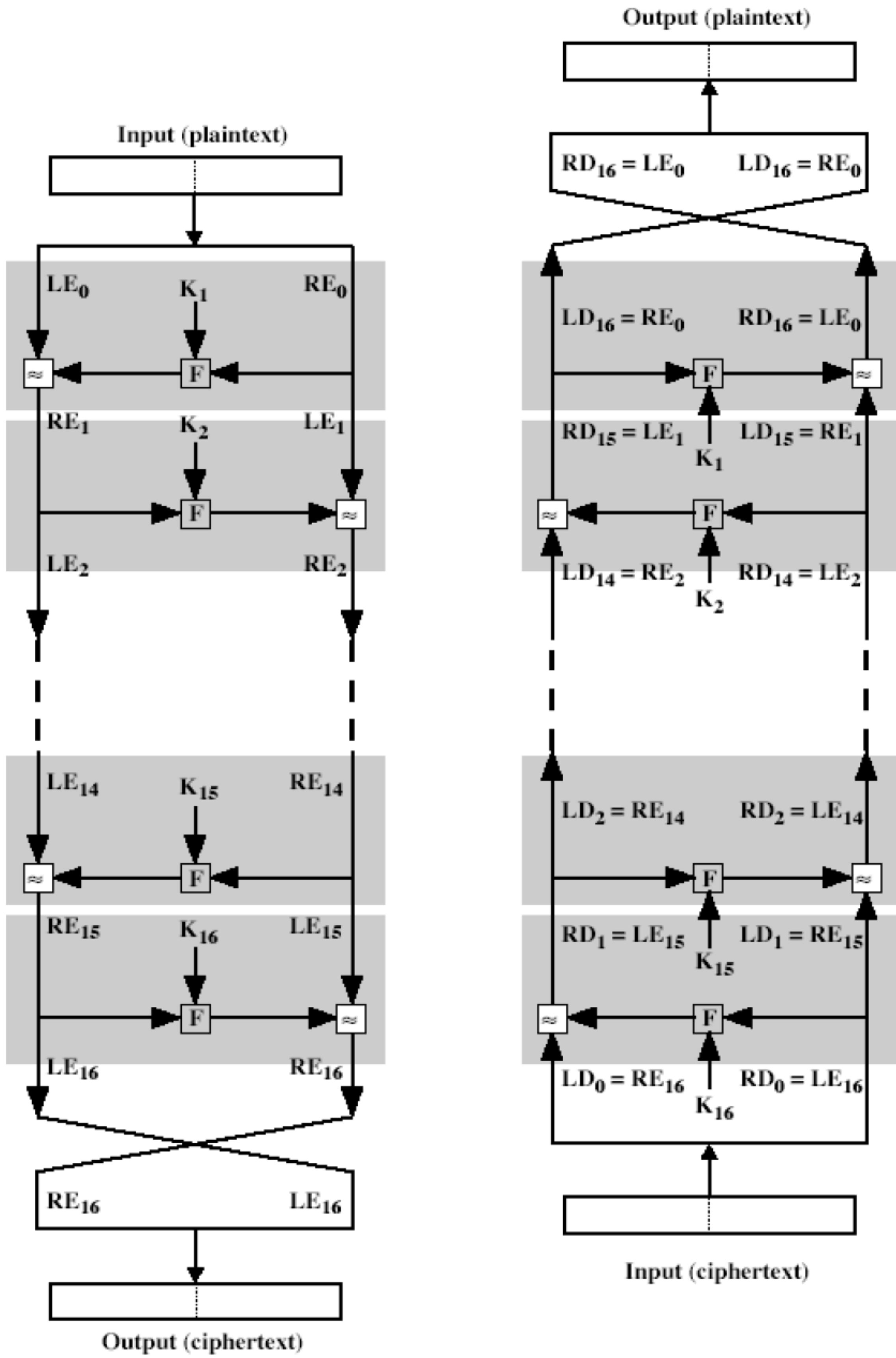
-

The process of decryption with a Feistel cipher is essentially the same as the en cryption process. The rule isas follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round,

- K_{n-1} in the second round, and so on, until K_1 is used in the last round.
- Figure 3.3 shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm.
- For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm.
-

The diagram indicates that, at every round, the intermediate value of the decrypt ion process is equal to the corresponding value of the encryption process with the two halves of the value swapped. let the output of the ith encryption round be $LE_i \| RE_i$. Then the corresponding input of the $(16-i)^{th}$ decryption round is $RE_i \| LE_i$ or, equivalently, $LD_{16-i} \| RD_{16-i}$.

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



Input (plaintext)
LE0
K1
RE0
F
RE1
K2
LE1
F
LE2
RE2
LE14
K15
RE14
F
RE15
K16
LE15
F
LE16
RE16
RE16
LE16
Output (ciphertext)
Output (plaintext)
RD16 = LE0
LD16 = RE0
LD16 = RE0
RD16 = LE0
F
RD15 = LE1
K1
LD15 = RE1
F
LD14 = RE2
K2
RD14 = LE2
LD2 = RE14
RD2 = LE14
F
RD1 = LE15
K15
LD1 = RE15
F
LD0 = RE16
K16
RD0 = LE16
Input (ciphertext)

- Now we would like to show that the output of the first round of the decryption process is equal to a 32bit swap of the input to the sixteenth round of the encryption process. Consider the encryption process.

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

On the decryption side,

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus \text{F}(RD_0, K_{16}) \\ &= RE_{16} \oplus \text{F}(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus \text{F}(RE_{15}, K_{16})] \oplus \text{F}(RE_{15}, K_{16}) \end{aligned}$$

The XOR has the following properties:

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$D \oplus D = 0$$
$$E \oplus 0 = E$$

- Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$. Therefore, the output of the first round of the decryption process is $RE_{15} \| LE_{15}$, which is the 32-bit swap of the input to the sixteenth round of the encryption. For the ith iteration of the encryption algorithm,

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

Rearranging terms:

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

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

DATA ENCRYPTION STANDARD

- The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).
- For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output.The same steps, with the same key, are used to reverse the encryption.

DES Encryption

The overall scheme for DES encryption is illustrated in Figure 3.5. 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.

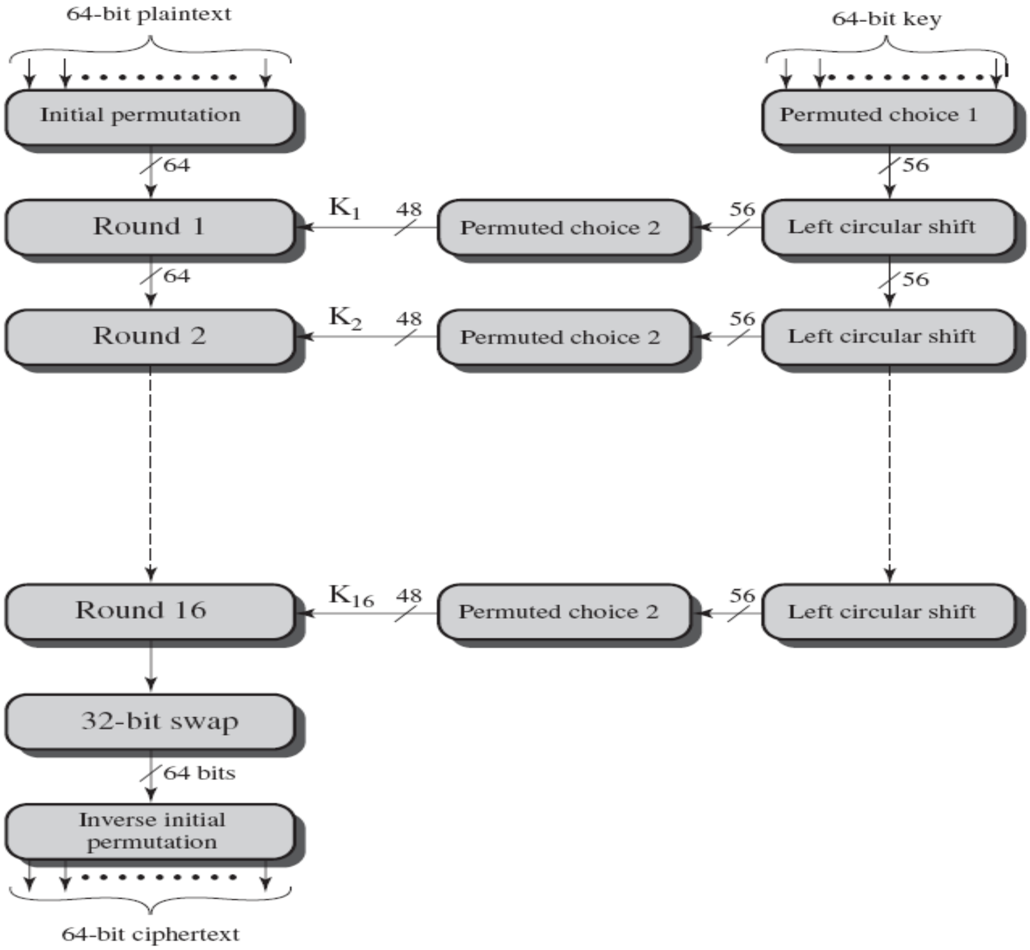


Figure 3.5 General Depiction of DES Encryption Algorithm

- From 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 sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key.
- The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

- The right-hand portion of Figure 3.5 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for eachof the sixteen rounds, a *subkey* (*Ki*) is produced by the combination of a left circular shift and a permutation.
- The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

INITIAL PERMUTATION

- The initial permutation and its inverse are defined by tables, as shown in Tables 3.2a and 3.2b, respectively. 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. consider the following 64-bit input M

Table 3.2 Permutation Tables for DES

(a) Initial Permutation (IP)

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

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

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

(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

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

where M_i is a binary digit. Then the permutation $X = (\mathrm{IP}(M)$ 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 $Y = \mathrm{IP}^{-1}(X) = \mathrm{IP}^{-1}(\mathrm{IP}(M))$, it can be seen that the original ordering of the bits is restored.

DETAILS OF SINGLE ROUND

The internal structure of a single round is shown in the figure. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right).

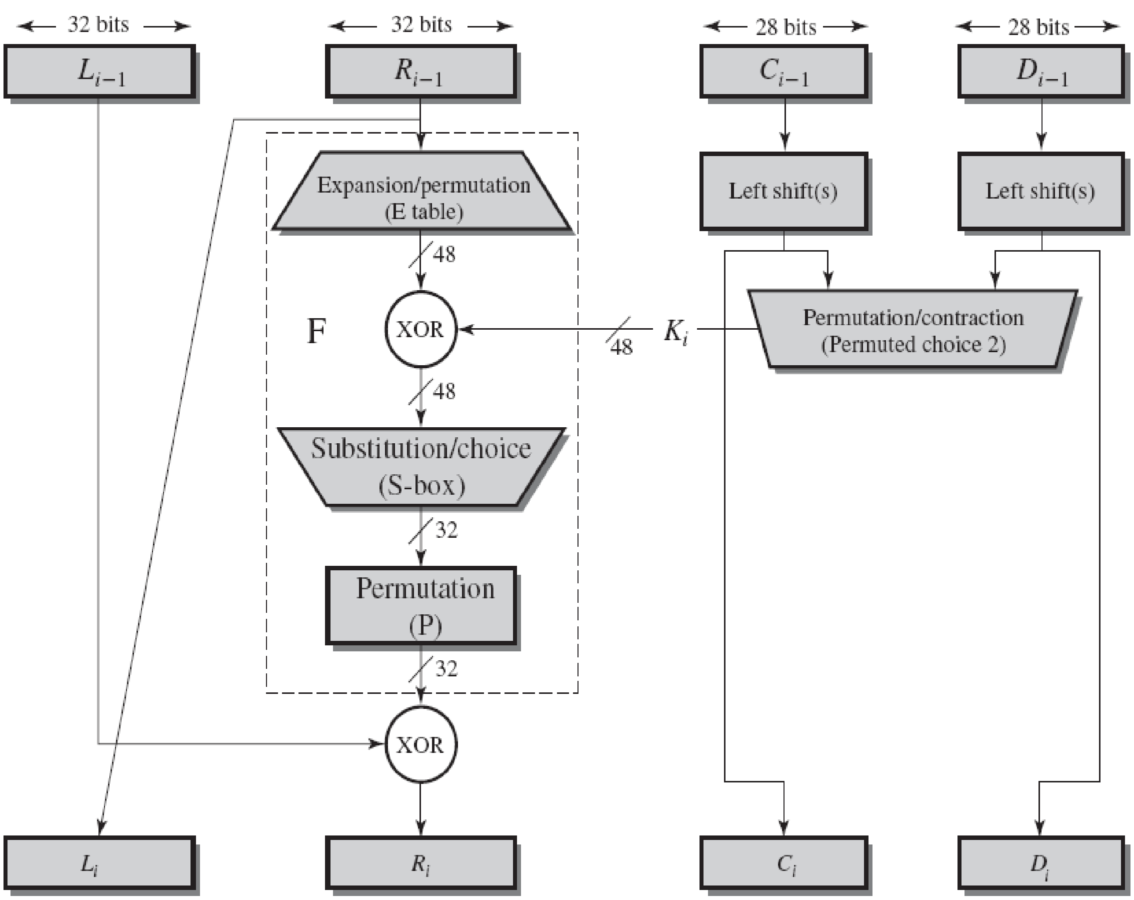
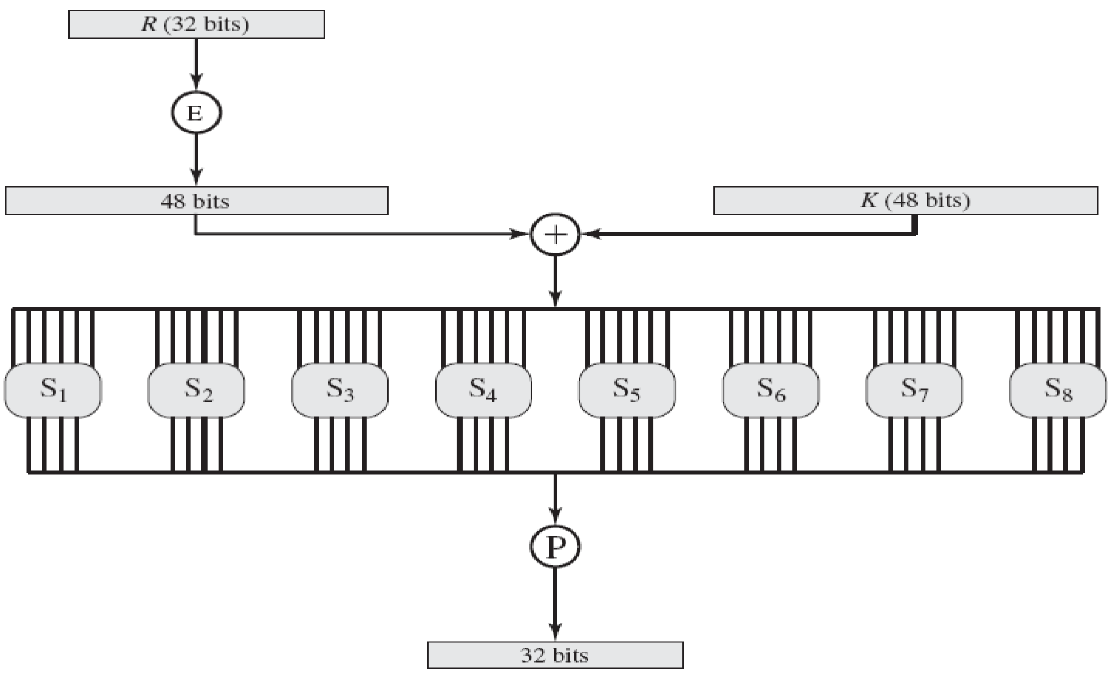


Figure 3.6 Single Round of DES Algorithm

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

- The round key is 48 bits. The input is 32 bits. This 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 bits (Table 3.2c).
- The resulting 48 bits are XORed with K_i . This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as in Table 3.2d.The role of the S-boxes in the function F is illustrated in Figure 3.7.
- The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output.



These transformations are defined in Table 3.3, which is interpreted as follows:

- The first and last bits of the input to box 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.
- Each row of an S-box defines a general reversible substitution.
- In 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 **lmnop**q ...

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

- From the Figures 3.5 and 3.6, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated in Table 3.4a.
- The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b).The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0 .

- At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift or (rotation) of 1 or 2 bits. These shifted values serve as input to the next round. 011001

Table 3.3 Definition of DES S-Boxes

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

	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
S_8	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

DES Decryption

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

DES Key Schedule Calculation

(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

(b) Permuted Choice One (PC-1)

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

(c) Permuted Choice Two (PC-2)

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

The Avalanche Effect

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

01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100

with two keys that differ in only one bit position:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100
0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

Table 3.5 Avalanche Effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

2.9 THE STRENGTH OF DES

→ Level of security provided by DES

Two areas:

i) Key size

ii) Nature of the algorithm.

→ The Use of 56-Bit Keys
→ The Nature of the DES Algorithm
→ Timing Attacks.

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

* With a key length of 56 bits, there are 2^{56} possible keys. 7.2×10^{16} keys.

* Brute-force attack appears impractical.
* DES finally and definitely proved insecure in July 1998
  - "DES cracker" machine
    - attack took < 3 days.

* Key-Search attack.
  - Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext.
    * Compressed message → difficult to automate.

* Brute-force attack
  - some degree of knowledge about the expected plaintext is needed
  - some means of automatically distinguishing plaintext from garble is also needed.

Alternatives to DES
- AES
- Triple DES.

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

* Cryptanalysis is possible by exploiting the characteristics of the DES algorithm.

Focus:

* Eight substitution tables or S-boxes, used in each iteration

- not made public, there is a suspicion.

* cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes.

- No one has so far succedded in discovering the supposed fatal weaknesses in the S-boxes.

3. Timing Attacks:

- relate to public key algorithms.

- may be relevant for symmetric ciphers.

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

- An encryption or decryption algorithm takes slightly different amounts of time on different inputs.

* fairly resistent to a successful timing attack.

2.10 DIFFERENTIAL AND LINEAR CRYPTANALYSIS

* Increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers.
* Two most powerful and promising approaches
  i) Differential cryptanalysis
  ii) Linear cryplanalysis.

1. Differential cryptanalysis:

→ History
→ Differential cryptanalysis Attack.

History:

- Not until 1990
- cryptanalysis of a block cipher called FEAL by Murphy.

* First published attack that is capable of breaking DES in < 2^{55} complexity.
  - does not do very well against DES
* The need to strengthen DES against attacks played a large part in the design of S-boxes and the permutation P.
* Differential cryptanalysis of an eight-round LUCIFER algorithm requires only 256 chosen plaintexts
  - an attack on an eight-round version of DES requires 2^{14} chosen plaintexts.

Differential Cryptanalysis Attack:

- original plaintext block m, consists of two halves m_0, m_1

* Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round.

- At each round, only one new 32-bit block is created.

Label each new block m_i $(2 \leq i \leq 17)$

$$m_{i+1} = m_{i-1} \oplus f(m_i, k_i), \quad i = 1, 2, \ldots, 16$$

Differential cryptanalysis :

- Two messages m, m'
- Known XOR difference $\Delta m = m \oplus m'$
- diff b/w the intermediate message halves:

$$\Delta m_i = m_i \oplus m_i'$$

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

* Many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used.

X may cause Y with propability p

- for a fraction p of the pairs
  the i/p XOR is X
  o/p XOR is Y

* If a number of differences are determined, it is feasible to determine the subkey used in the function f.

→ Overall strategy is based on the considerations for a single round.

Procedure:

- Two plaintext messages m, m'
- difference.
- Trace through a probable pattern of difference after each round to yield a probable difference for the ciphertext.
- 2 diff for 2 32-bit halves: $(\Delta m_{17} \| \Delta m_{16})$
- submit m, m' for encryption to determine actual difference. under unknown key.
- Compare the result to probable difference.

* If there is a match,

$$E_K(m) \oplus E_K(m') = (\Delta m_{17} \| \Delta m_{16})$$

- then suspect that all probable pattern at all the intermediate rounds are correct.
  - make deductions about the key bits
* Procedure must be repeated many times to determine all the key bits.

Differential Propagation through Three Rounds of DES

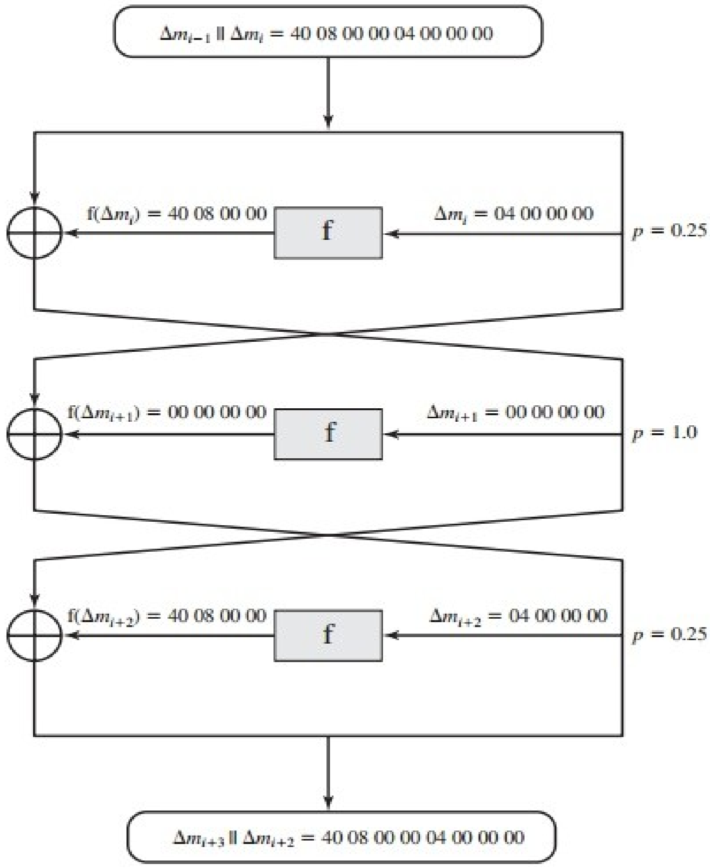


Figure 3.8 Differential Propagation through Three Rounds of DES (numbers in hexadecimal)

Linear Cryptanalysis :

- It is the more recent development
- This attack is based on finding linear approximations to describe the transformations performed in DES.
- This method can find a DES key given 2^{43} known plaintexts, as compared to2^{47} chosen plaintexts for differential cryptanalysis. I
- t may be easier to acquire known plaintext rather than chosen plaintext, leaves.
- For a cipher with n -bit plaintext and ciphertext blocks and m -bit key, let the plaintext block be labeled P[1], …P[n], the cipher text block C[1], .. C[n], and the key K[1], … , K[m] . Then define

$$A[i, j, \dots, k] = A[i] \oplus A[j] \oplus .. \oplus A[k]$$

- The objective of linear cryptanalysis is to find an effective linear equation of the form:

$$P[\alpha 1,\alpha 2\dots,\alpha a] \oplus C[\beta 1,\beta 2,\dots,\beta b] = K[\gamma 1,\gamma 2,\dots\dots\gamma c]$$

(where x = 0 or 1; $1 \leq a;b \leq n$; $c \leq m$; and where the α,β,γ terms represent fixed, unique bit locations) that holds with probability $p \neq 0.5$.

- The further p is from 0.5, the more effective the equation.
- Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext–ciphertext pairs.
  - If the result is 0 more than half the time, assume $K[\gamma 1,\gamma 2,\dots\dots\gamma c]=0$.
  - If it is 1 most of the time, assume $K[\gamma 1,\gamma 2,\dots\dots\gamma c]=1$.
- This gives us a linear equation on the key bits.

2.11 BLOCK CIPHER DESIGN PRINCIPLES

Three aspects:

i) the number of rounds
ii) design of the function F
iii) Key scheduling.

→ DES Design Criteria
→ Number of Rounds
→ Design of Function F
  * Design Criteria for F
  * S-Box Design
→ Key Schedule Algorithm

DES Design Criteria:

* The criteria used in the design of DES focused on the design of the S-boxes and on the P function that takes the output of the S-boxes.

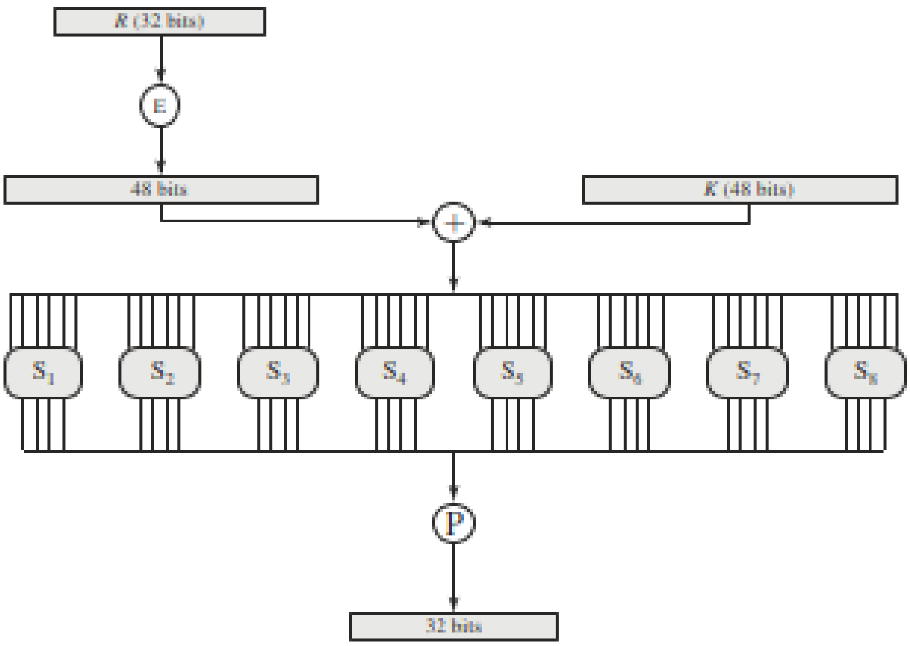


Figure 3.7 Calculation of F(R, K)

* The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits.
2. Each row of an S-box should include all 16 possible output bit combinations.

3. If two inputs to an S-box differ in exactly one bit, the o/ps must differ in atleast two bits.

4. If two i/ps to an S-box differ in the two middle bits exactly, the o/ps must differ in atleast two bits

5. If two i/ps to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.

6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of i/ps exhibiting that difference may result in the same o/p difference

7. This is a criterion similar to the previous one, but for the case of three S-boxes.

* The criteria for the permutation P are as follows:

1. The four o/p bits from each S-box at round i are distributed.
   - So that two of them affect "middle bits" of round (i+1) and the other two affect end bits.

     two middle bits → not shared

     end bits → two left-hand bits / two right-hand bits } → shared with adjacent S-boxes

2. The four output bits from each S-box affect six different S-boxes on the next round and no two affect the same S-box.

3. For two S-boxes j, k, if an o/p bit from S_j affects a middle bit of S_k on the next round, then an o/p bit from S_k cannot affect a middle bit of S_j.

   j = k, o/p bit from S_j must not affect a middle bit of S_j

   → to increase the diffusion of the algorithm.

Number of Rounds:

* The greater the number of rounds, the more difficult it is to perform cryptanalysis.

16-round DES:

differential cryptanalysis attack requires $2^{55.1}$ operations

brute force " " 2^{55} "

DCA less efficient that brute force.

* Using this criteria, it is easy to judge the strength of an algorithm and to compare different algorithms

Design of Function F:

* Heart of a Feistal block cipher
  - relies on the use of S-boxes.

Design criteria for F:

* The function F provides the element of confusion in a Feistal Cipher.
  - It must be difficult to "unscramble" the substitution performed by F.

① - F be nonlinear
* The more nonlinear F, the more difficult any type of cryptanalysis will be.

② The algorithm to have good avalanche properties.
→ a change in one bit of the input should produce a change in many bits of the output.
– strict avalanche criterion (SAC)

③ Bit Independence criterion (BIC)
→ the output bits j & k should change independently when any single input bit i is inverted, $\forall i, j, k$.

* SAC & BIC – to strengthen the effectiveness of the confusion function.

S-Box Design:

Principles:
- Any change to the input vector to an S-box to result in random-looking changes to the output.
- relationship should be nonlinear and difficult to approximate with linear functions.

Characteristics
- Size of S-box.
  * An n×m S-box has n i/p bits & m o/p bits.
- DES has 6×4 S-boxes.
  Blowfish has 8×32 S-boxes.
* Larger S-boxes, by and large, are more resistant to differential + linear cryptanalysis.

- larger the S-box, more difficult it is to design it properly.

* An $n \times m$ S-box consists of 2^n rows of m bits each.
- The n bits of i/p select one of the rows of S-box, and the m bits in that row are the output.

* All linear combinations of S-box columns should be bent.
  - Bent functions are a special class of Boolean functions that are highly nonlinear according to certain mathematical criteria.

Guaranteed Avalanche criteria (GA):

* An S-box satisfies GA of order γ iff, for a 1-bit i/p change, at least γ o/p bit change.
  - a GA in the range of order 2 to order 5 provides strong diffusion characteristics for the overall encryption algorithm.

Approaches for S-box design:

Random:

- Use some pseudorandom number generation or some table of random digits to generate the entries in the S-boxes.

small size 6×4
large size 8×32

eg: Blowfish
- Key dependent
- impossible to analyze

Random with testing:

- Choose S-box entries randomly, then test the results against various criteria, and throw away those that do not pass.

Human-made:

- manual approach with only simple mathematics to support it.
- difficult for large S-boxes.

Math-made:

- Generate S-boxes according to mathematical principles.

Key Schedule Algorithm:

* With any Feistal block cipher, the key is used to generate one subkey for each round.
- Key schedule should guarantee key/ciphertext SAC & BIC.

2.12 BLOCK CIPHER MODES OF OPERATION.

* The DES algorithm is a basic building block for providing data security.

Five modes of operation

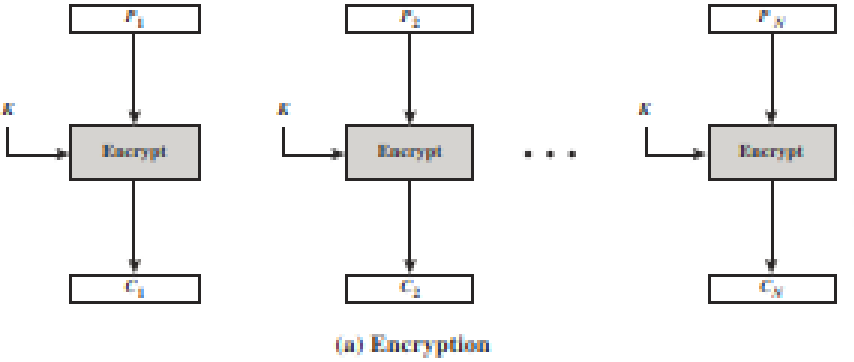
i) Electronic codebook Mode
ii) Cipher Block chaining Mode
iii) Cipher Feedback Mode.
iv) Output Feedback Mode.
v) Counter Mode.

Table 6.1 Block Cipher Modes of Operation

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

1 **Electronic codebook Mode:**

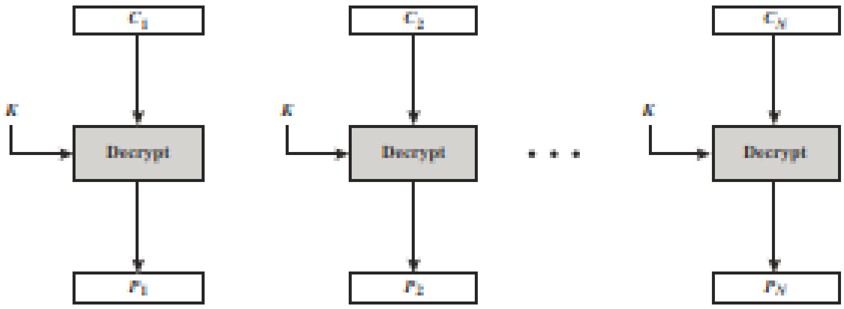
- Simplest mode
- Plaintext is handled 64 bits at a time. and each block of plaintext is encrypted using the same key.



(a) Encryption

Code book → For a given key, there is a unique ciphertext for every 64-bit block of plaintext.

* For a message longer than 64 bits, the procedure is simply to break the message into 64-bit blocks, padding the last block if necessary.



(b) Decryption

Figure 6.3 Electronic Codebook (ECB) Mode

* Decryption is performed one block at a time, always using the same key.
- The plaintext consists of a sequence of 64-bit blocks, P1, P2, ..., PN; the corresponding sequence of ciphertext blocks is C1, C2, .., CN

* ECB method is ideal for a short amount of data, such as an encryption key.
- To transmit a DES key securely, ECB is the appropriate mode to use.

Characteristic:
- The same 64-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

Disadvantages:

* For lengthy messages, the ECB mode may not be secure

2 Cipher Block Chaining Mode:

→ A technique in which the same plaintext block, if repealed, produces different ciphertext blocks.

* 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.

— No fixed relationship to the plaintext block

* repeating patterns of 64 bits are not exposed.

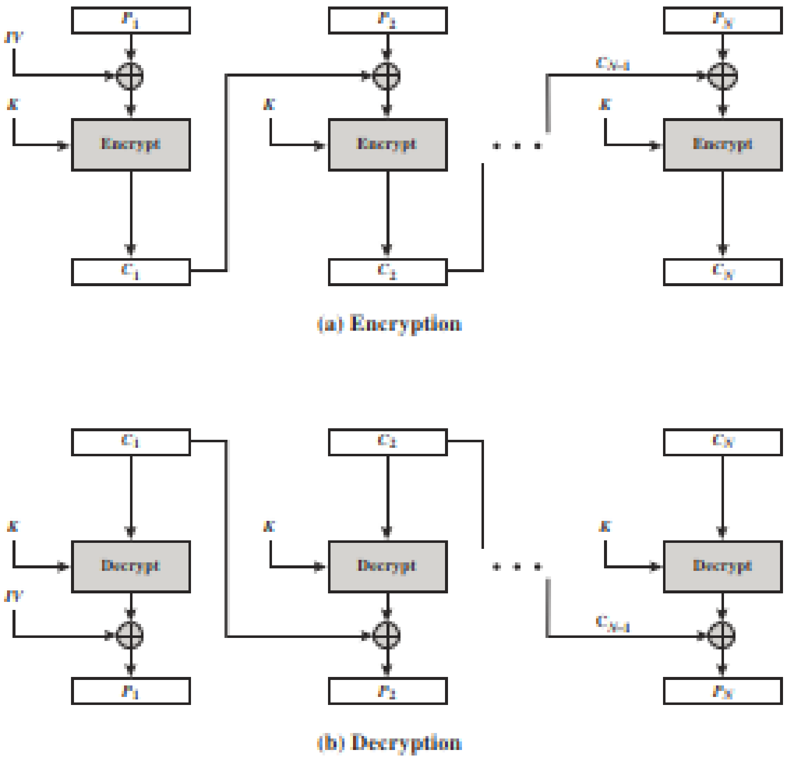


Figure 6.4 Cipher Block Chaining (CBC) Mode

Decryption:

* Each cipher block is passed through the decryption algorithm

— The result is XORed with the preceding ciphertext block to produce the plaintext block.

$$C_j = E_K[C_{j-1} \oplus P_j]$$

$$D_K[C_j] = D_K[E_K(C_{j-1} \oplus P_j)]$$

$$= C_{j-1} \oplus P_j$$

$$C_{j-1} \oplus D_K[C_j] = C_{j-1} \oplus C_{j-1} \oplus P_j$$
$$= P_j$$

* 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.

IV:

* must be known to both the sender & receiver.
* Should be protected as well as the key.

- Sending the IV using ECB encryption.

Reason for protecting the IV:

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

$$P_1 = IV \oplus D_K(C_1)$$
$$C_1 = E_K(IV \oplus P_1)$$

$x[i] \rightarrow i^{th}$ bit of the 64-bit quantity x.

$$P_1[i] = IV[i] \oplus D_K(C_1)[i]$$

using the properties of XOR, bit complementation

$$P_1[i]' = IV[i]' \oplus D_K(C_1)[i]$$

* If an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed.

Advantages

* chaining mechanism → appropriate mode for encrypting messages of length greater than 64 bits
* To achieve confidentiality
* Used for authentication.

3. Cipher Feedback(CFB):

- It is assumed that the unit of transmission is bits; a common value is . 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 blocks of bits, the plaintext is divided into segments of bits.
- The message is treated as a *stream* of bits that is added to the output of the block cipher.
- The result is feedback for the next stage.

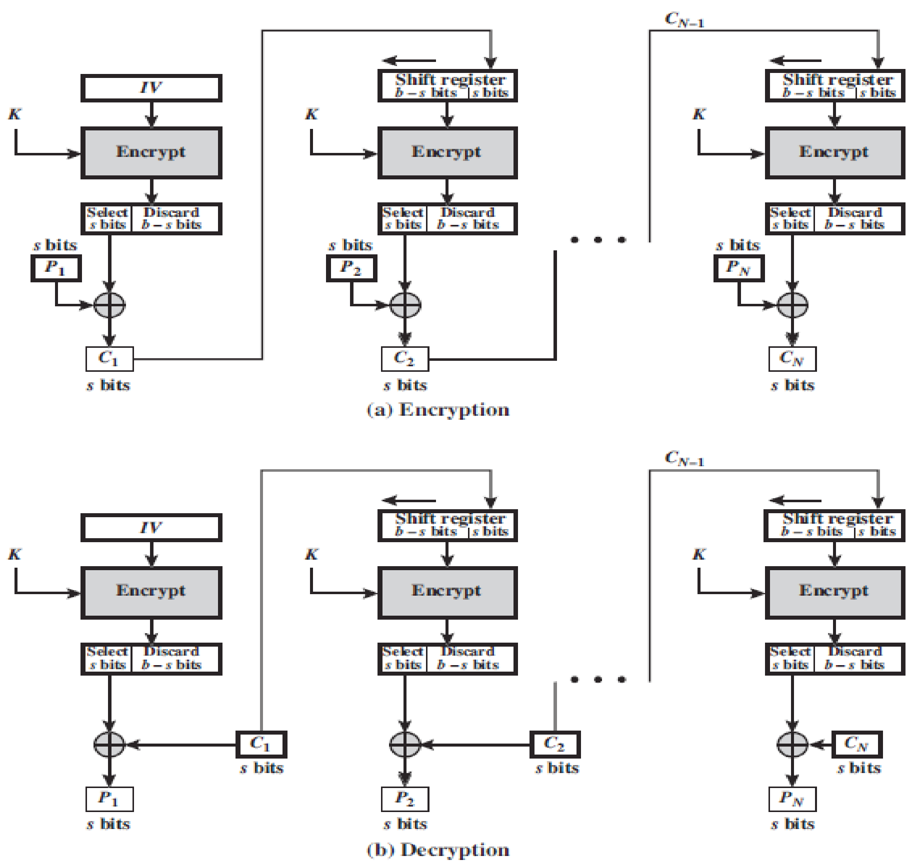


Figure: s bit Cipher Feedback mode(CFB)

Encryption:

- The input to the encryption function is a b-bit shift register initially set to some initialization vector (IV).
- The leftmost 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 C, which is then transmitted.
- The contents of the shift register are shifted left by s bits, and C1 is placed in the rightmost s bits of the shift register.
- This process continues until all plaintext units have been encrypted.

Decryption:

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

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

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

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

Define CFB:

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

Advantages:

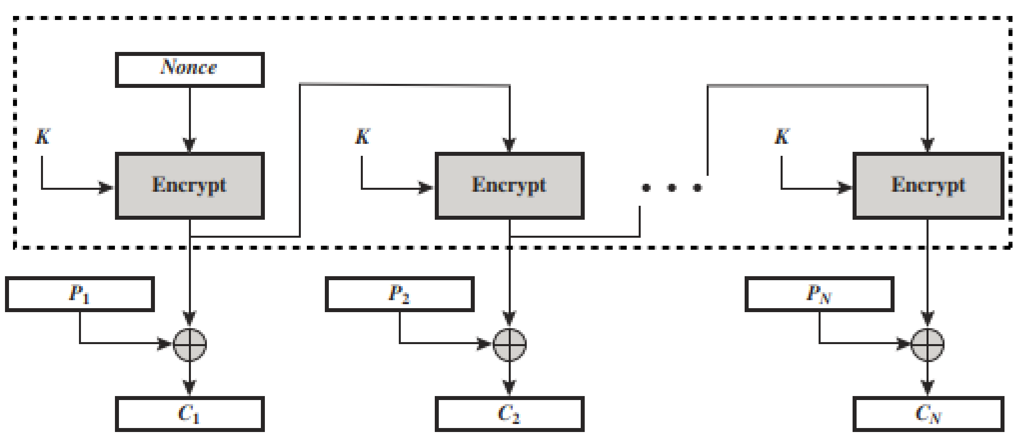
- Appropriate when data arrives in bits/bytes.
- It is the most common stream mode.

Disadvantages:

- The need to stall while you do block encryption after every n-bits.
- Note that the block cipher is used in encryption mode at both ends.
- Errors propagate for several blocks after the error.

4. Output Feedback Mode(OFB):

- It is similar in the structure of CFB.
- It is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB, the ciphertext unit is fed back to the shift register.
- The difference is that the OFB mode operates on full blocks of plaintext and ciphertext, not on an s-bit subset.



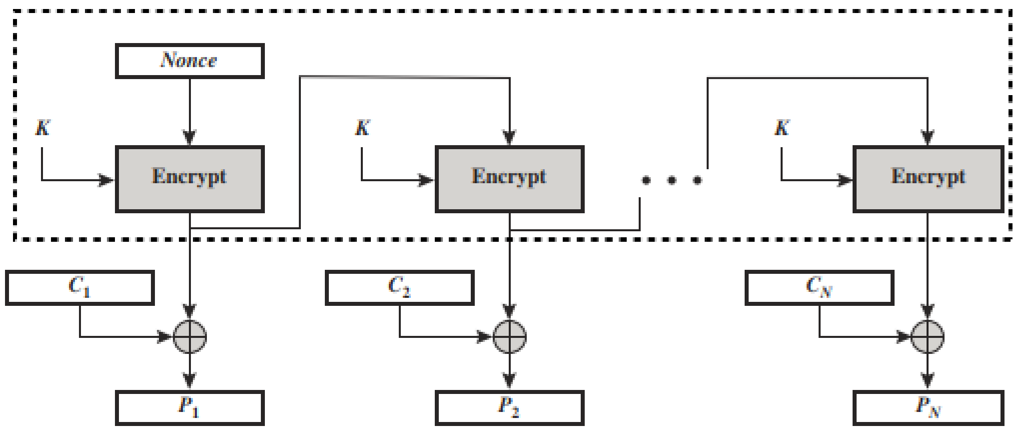
(a) Encryption

- OFB has the structure of a typical stream cipher, because the cipher generates a stream of bits as a function of an initial value and a key, and that stream of bits is XORed with the plaintext bits.
- The generated stream that is XORed with the plaintext is itself independent of the plaintext
- Encryption can be expressed as

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

- Decryption

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



(b) Decryption

Define OFB:

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

- The OFB mode requires an initialization vector.
- In the case of OFB, the IV must be a nonce;
  - that is, the IV must be unique to each execution of the encryption operation.
- The reason for this is that the sequence of encryption output blocks, depends only on the key and the IV and does not depend on the plaintext.
- Therefore, for a given key and IV, the stream of output bits used to XOR with the stream of plaintext bits is fixed.
- If two different messages had an identical block of plaintext in the identical position, then an attacker would be able to determine that portion of the stream.

Advantages:

- Bit errors in transmission do not propagated.
  - Ex:
    - If a bit error occurs in , only the recovered value of is affected; subsequent plaintext units are not corrupted.

Disadvantages:

- More vulnerable to message stream modification attack.

5. Counter Mode(CTR):

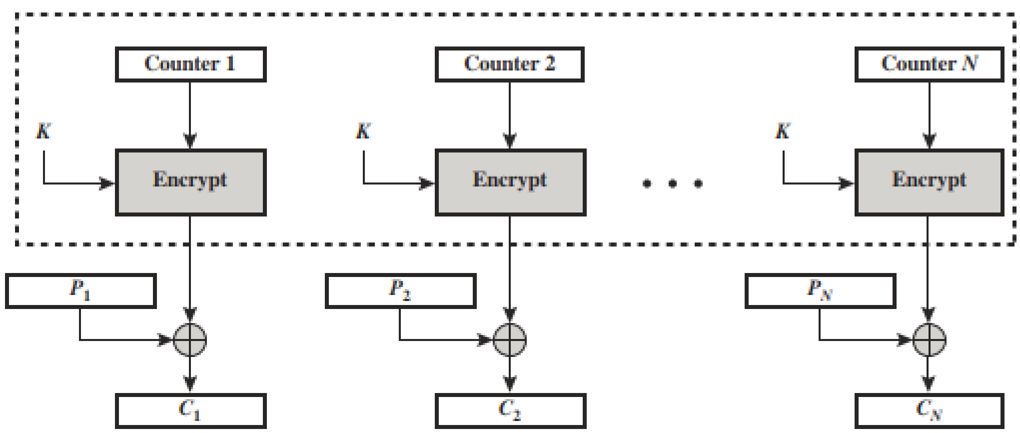
- The counter equal to the plaintext block size is used.
- The counter value must be different for each plaintext block that is encrypted.
- The counter is initialize to some values, then will be incremented by one for each subsequent block.(modulo 2^b, b is block size)

Encryption:

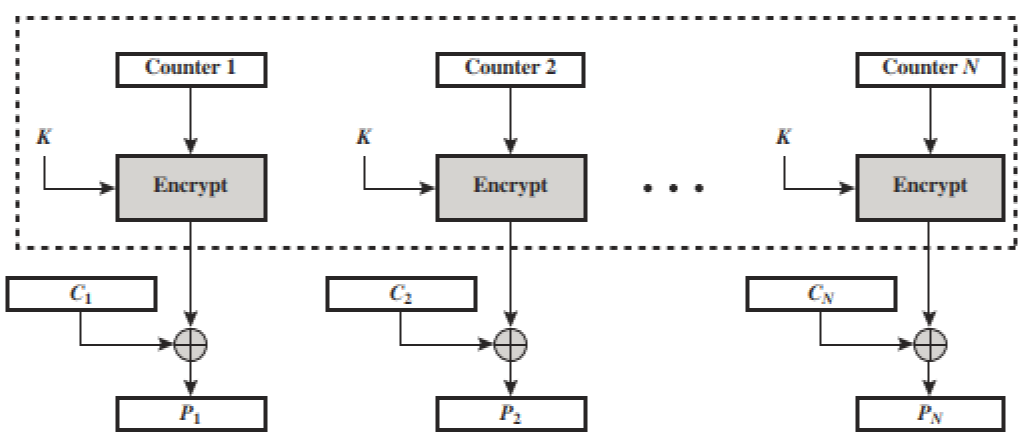
- The counter is encrypted and XORed with the plaintext block to produce the ciphertext block.
- There is no chaining.

Decryption:

- The same sequence of counter values is used, with each encrypted counter XORed with the ciphertext block to recover the corresponding plaintext block.
- the initial counter value must be made available for decryption.



(a) Encryption



(b) Decryption

Define CTR:

CTR	$C_j = P_j \oplus E(K, T_j) \quad j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus MSB_u[E(K, T_N)]$	$P_j = C_j \oplus E(K, T_j) \quad j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus MSB_u[E(K, T_N)]$

- The initial counter value must be a nonce;
  - that is, must be different for all of the messages encrypted using the same key.
- All values across all messages must be unique.
- a counter value is used multiple times, then the confidentiality of all of the plaintext blocks corresponding to that counter value may be compromised
- To ensure the uniqueness of counter values is to continue to increment the counter value by 1 across messages.
- That is, the first counter value of the each message is one more than the last counter value of the preceding message.

Advantages:

- Hardware efficiency
  - Encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext.
  - The throughput is only limited by the amount of parallelism that is achieved.
- Software efficiency
  - opportunities for parallel execution in CTR mode,
  - processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
- Preprocessing
  - preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions,
- Random access
  - The th block of plaintext or ciphertext can be processed in random-access fashion.
- Provable security
  - CTR is at least as secure as the other modes
- Simplicity
  - CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm

2.13 EVALUATION CRITERIA FOR AES

→ The origin of AES

→ AES Evaluation

- Three categories
  - * Security
  - * Cost
  - * Algorithm and implementation characteristics
- Criteria for final evaluation
  - * General Security
  - * Software implementations
  - * Restricted-space environments
  - * Hardware implementations
  - * Attacks on implementations
  - * Encryption versus decryption
  - * Key agility
  - * Other versatility & flexibility
  - * Potential for instruction-level Parallelism

The origin of AES:

Disadvantages in DES & 3DES:

3DES:

- relatively sluggish in software
- Not reasonable candidate for long-term use.
- Three times as many rounds as DES
- slower

DES:

- should only used for legacy systems.
- does not produce efficient s/w code.

3DES & DES → use a 64-bit block size.

* For efficiency & security, large block size is desirable.

* AES (Advanced Encryption Standard) was published by NIST (National Institute of Standards and Technologies) in 2001.

→ NIST specified that AES must be symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192 and 256 bits.

AES Evaluation:

Three categories of criteria:

i) Security:
- effort required to cryptanalyze an algorithm.
- min. key size for AES is 128 bits.

ii) Cost:
- practical in a wide range of applications
- AES must have high computational efficiency.
- usable in high-speed applications - broadband links.

iii) Algorithm and implementation characteristics:

Considerations:
- flexibility
- suitability for h/w & s/w impl^ns
- simplicity

Table 5.1 NIST Evaluation Criteria for AES (September 12, 1997) (page 1 of 2)

SECURITY

- **Actual security:** compared to other submitted algorithms (at the same key and block size).
- **Randomness:** The extent to which the algorithm output is indistinguishable from a random permutation on the input block.
- **Soundness:** of the mathematical basis for the algorithm's security.
- **Other security factors:** raised by the public during the evaluation process, including any attacks which demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.

COST

- **Licensing requirements:** NIST intends that when the AES is issued, the algorithm(s) specified in the AES shall be available on a worldwide, non-exclusive, royalty-free basis.
- **Computational efficiency:** The evaluation of computational efficiency will be applicable to both hardware and software implementations. Round 1 analysis by NIST will focus primarily on software implementations and specifically on one key-block size combination (128-128); more attention will be paid to hardware implementations and other supported key-block size combinations during Round 2 analysis. Computational efficiency essentially refers to the speed of the algorithm. Public comments on each algorithm's efficiency (particularly for various platforms and applications) will also be taken into consideration by NIST.
- **Memory requirements:** The memory required to implement a candidate algorithm--for both hardware and software implementations of the algorithm--will also be considered during the evaluation process. Round 1 analysis by NIST will focus primarily on software implementations; more attention will be paid to hardware implementations during Round 2. Memory requirements will include such factors as gate counts for hardware implementations, and code size and RAM requirements for software implementations.

ALGORITHM AND IMPLEMENTATION CHARACTERISTICS

- **Flexibility:** Candidate algorithms with greater flexibility will meet the needs of more users than less flexible ones, and therefore, inter alia, are preferable. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preference will not be given. Some examples of flexibility may include (but are not limited to) the following:
  - **a.** The algorithm can accommodate additional key- and block-sizes (e.g., 64-bit block sizes, key sizes other than those specified in the Minimum Acceptability Requirements section, [e.g., keys between 128 and 256 that are multiples of 32 bits, etc.])
  - **b.** The algorithm can be implemented securely and efficiently in a wide variety of platforms and applications (e.g., 8-bit processors, ATM networks, voice & satellite communications, HDTV, B-ISDN, etc.).
  - **c.** The algorithm can be implemented as a stream cipher, message authentication code (MAC) generator, pseudorandom number generator, hashing algorithm, etc.
- **Hardware and software suitability:** A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility.
- **Simplicity:** A candidate algorithm shall be judged according to relative simplicity of design.

Table 5.2 Final NIST Evaluation of Rijndael (October 2, 2000) (page 1 of 2)

General Security

Rijndael has no known security attacks. Rijndael uses S-boxes as nonlinear components. Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have facilitated its security analysis during the timeframe of the AES development process.

Software Implementations

Rijndael performs encryption and decryption very well across a variety of platforms, including 8-bit and 64-bit platforms, and DSPs. However, there is a decrease in performance with the higher key sizes because of the increased number of rounds that are performed. Rijndael's high inherent parallelism facilitates the efficient use of processor resources, resulting in very good software performance even when implemented in a mode not capable of interleaving. Rijndael's key setup time is fast.

Restricted-Space Environments

In general, Rijndael is very well suited for restricted-space environments where either encryption or decryption is implemented (but not both). It has very low RAM and ROM requirements. A drawback is that ROM requirements will increase if both encryption and decryption are implemented simultaneously, although it appears to remain suitable for these environments. The key schedule for decryption is separate from encryption.

Hardware Implementations

Rijndael has the highest throughput of any of the finalists for feedback modes and second highest for non-feedback modes. For the 192 and 256-bit key sizes, throughput falls in standard and unrolled implementations because of the additional number of rounds. For fully pipelined implementations, the area requirement increases, but the throughput is unaffected.

Attacks on Implementations

The operations used by Rijndael are among the easiest to defend against power and timing attacks. The use of masking techniques to provide Rijndael with some defense against these attacks does not cause significant performance degradation relative to the other finalists, and its RAM requirement remains reasonable. Rijndael appears to gain a major speed advantage over its competitors when such protections are considered.

Encryption vs. Decryption

The encryption and decryption functions in Rijndael differ. One FPGA study reports that the implementation of both encryption and decryption takes about 60% more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than for encryption.

Key Agility

Rijndael supports on-the-fly subkey computation for encryption. Rijndael requires a one-time execution of the key schedule to generate all subkeys prior to the first decryption with a specific key. This places a slight resource burden on the key agility of Rijndael.

Other Versatility and Flexibility

Rijndael fully supports block sizes and key sizes of 128 bits, 192 bits and 256 bits, in any combination. In principle, the Rijndael structure can accommodate any block sizes and key sizes that are multiples of 32, as well as changes in the number of rounds that are specified.

Potential for Instruction-Level Parallelism

Rijndael has an excellent potential for parallelism for a single block encryption.

2.14 ADVANCED ENCRYPTION STANDARD

- → AES Parameters
  - characteristics
- → AES Structure
- → Substitute Bytes Transformation
  - Forward and Inverse Transformations
  - Rationale
- → Shift Row Transformation
  - Forward and Inverse Transformations
  - Rationale
- → Mix Column Transformation
  - Forward and Inverse Transformations
  - Rationale
- → Add Round Key Transformation
  - Forward and Inverse Transformations
  - Rationale
- → AES Key Expansion
  - Key Expansion Algorithm
  - Rationale
- → Equivalent Inverse Cipher
  - Interchanging InvShiftRows and InvSubBytes
  - Interchanging AddRoundKey and InvMixColumns
- → Implementation Aspects
  - 8-Bit Processor
  - 32-Bit Processor

→ Block cipher
Symmetric algo
→ Block size - 128 bits
PT
(4 words / 16 bytes)
1 word - 32 bits
→ No of rounds - 10
→ Key size - 128 bits
→ No. of subkeys - 44
→ Sub Key size
1 word / 32 bit / 4 bytes
C.T - 128 bits (4 words / 16 bytes)
→ Each round
4 subkeys
4 × 32
(128 bit / 4 words / 16 bytes)

* The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192 or 256 bits.
  → The AES specification uses the same three key size alternatives.

* A number of AES parameters depend on the Key length.

Table 5.1 AES Parameters

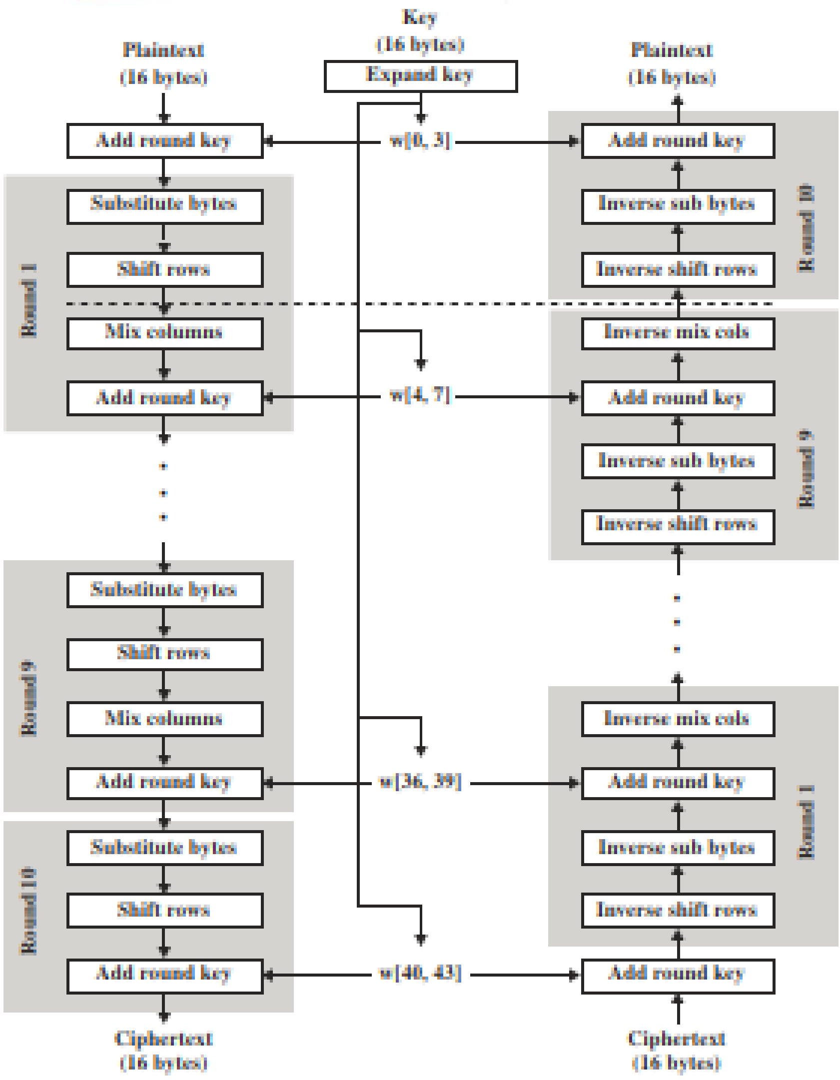
Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Assume a key length of 128 bits.

Characteristics

* Resistance against all known attacks
* Speed and code compactness on a wide range of platforms
* Design simplicity.

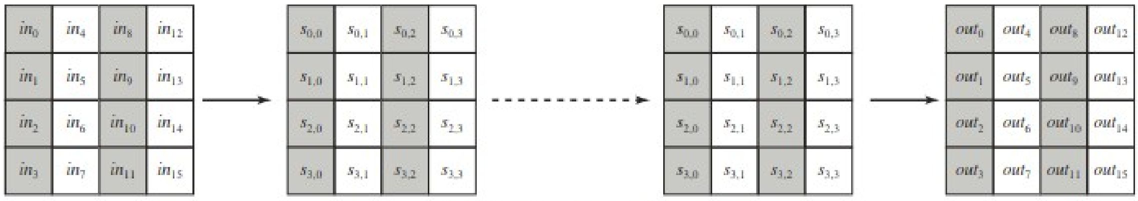
Overall Structure of AES



(a) Encryption (b) Decryption

Figure 5.3 AES Encryption and Decryption

- The input to the encryption and decryption algorithms is a single 128-bit block.
- This block is depicted as a square matrix of bytes.
- This block is copied into the State array, which is modified at each stage of encryption or decryption.
- After the final stage, State is copied to an output matrix.



(a) Input, state array, and output

- Similarly, the key is depicted as a square matrix of bytes.
- This key is then expanded into an array of key schedule words.

(b) Key and expanded key

- Each word is four bytes, and the total key schedule is 44 words for the 128-bit key.
- Note that the ordering of bytes within a matrix is by column.
    - Example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on.
    - Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the w matrix.

Comments about the overall AES structure.

1. One noteworthy feature of this structure is that it is not a Feistel structure.

AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.

2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round

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

• Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
• ShiftRows: A simple permutation
• MixColumns: A substitution that makes use of arithmetic over
• AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.

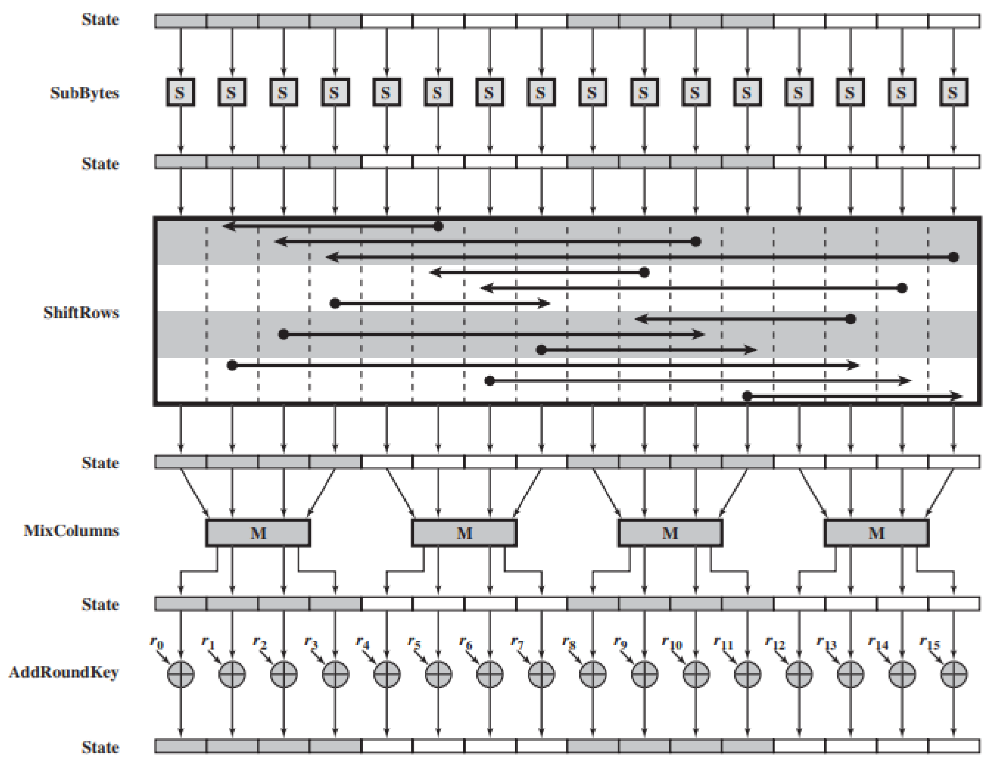


Figure 5.4 AES Encryption Round

5. Only the AddRoundKey stage makes use of the key.
   - For this reason, the cipher begins and ends with an AddRoundKey stage.
   - Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.

6. The AddRoundKey stage is a form of Vernam cipher and by itself would not be formidable.
   - The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key.We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on.This scheme is both efficient and highly secure.

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

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

9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.
   - Encryption and decryption going in opposite vertical directions.

- At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.

10. The final round of both encryption and decryption consists of only three stages.
    - Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

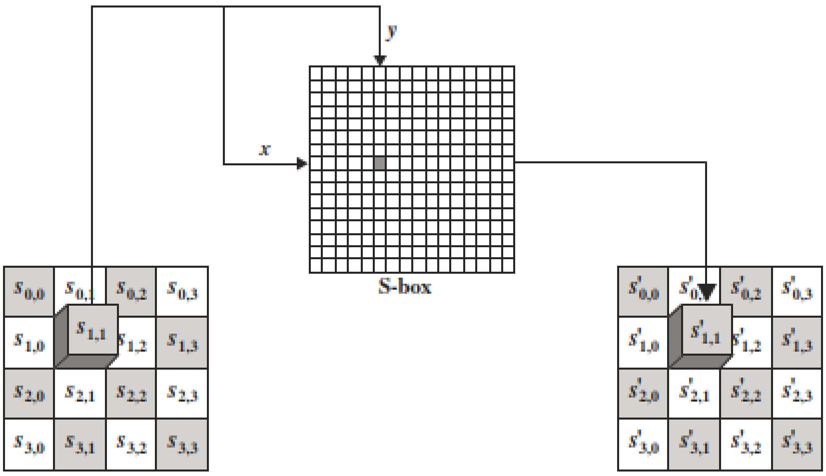
AES TRANSFORMATION FUNCTIONS

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

1. Substitute Bytes Transformation

FORWARD AND INVERSE TRANSFORMATIONS

a) The forward **substitute byte transformation**, called SubBytes, is a simple table lookup.



(a) Substitute byte transformation

- AES defines a matrix of byte values, called an S-box, that contains a permutation of all possible 256 8-bit values.

Table 5.2 AES S-Boxes

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

(a) S-box

- Each individual byte of State is mapped into a new byte in the following way:
  - The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.
  - These row and column values serve as indexes into the S-box to select a unique 8-bit output value.
    - Ex: The hexadecimal value[3] {95} references row 9, column 5 of the S-box, which contains the value {2A} . Accordingly, the value {95} is mapped into the value {2A} .

Example of the SubBytes transformation:

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

→

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

Construction of S-Box:
1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, ……, {0F} ; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row y , column x is {yx} .
2. Map each byte in the S-box to its multiplicative inverse in the finite field $GF(2)^8$; the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labelled (b7, b6, b5, b4, b3, b2, b1, b0) . Apply the following transformation to each bit of each byte in the S-box:

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \qquad \textbf{(5.1)}$$

where is the ith bit of byte c with the value {63} ; that is, (c7c6c5c4c3c2c1c0) =(01100011) .

- The prime(‘) indicates that the variable is to be updated by the value on the right.
- The AES standard depicts this transformation in matrix form as follows.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad \textbf{(5.2)}$$

- Each element in the product matrix is the bitwise XOR of products of elements of one row and one column.
- Furthermore, the final addition is a bitwise XOR.
  - the bitwise XOR is addition in $GF(2^8)$.

Example, consider the input value{95}
- The multiplicative inverse in $GF(2^8)$is $\{95\}^{-1}$ = {8A} , which is 10001010 in binary.

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

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

b) The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box

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

(b) Inverse S-box

Example, that the input {2A} produces the output {95} , and the input {95} to the S-box produces {2A} .
The inverse S-box is constructed by applying the inverse of the transformation in Equation followed by taking the multiplicative inverse in GF(2^8) .
The inverse transformation is

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

where byte d = {05} , or 00000101.

Transformation as follows.

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

- InvSubBytes is the inverse of SubBytes, label the matrices inSubBytes and InvSubBytes as X and B, respectively, and the vector versions of constants c and d as C and D, respectively.

- For some 8-bit vector B Equation (5.2) becomes B’ = XB ⊕ C .
- Need to show that Y(XB ⊕ C) ⊕ D=B.
- To multiply out, we must show YXB ⊕ YC ⊕ D=B. This becomes

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

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

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

YX equals the identity matrix, and the YC=D ,so that YC ⊕ D equals the null vector.

RATIONALE

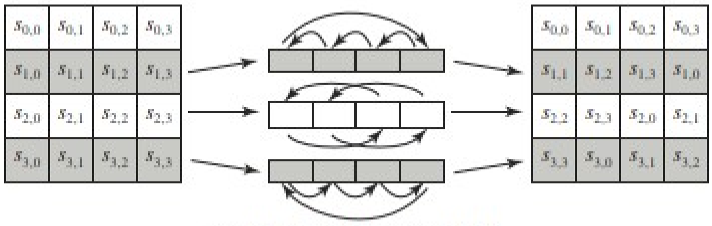
- The S-box is designed to be resistant to known cryptanalytic attacks.
- The nonlinearity is due to the use of the multiplicative inverse.
- In addition, the constant in Equation was chosen so that the S-box has no fixed points [S-box(a) = a] and no “opposite fixed points” [S-box(a) = a] , where a^- is the bitwise complement of a .

2. ShiftRows Transformation

FORWARD AND INVERSE TRANSFORMATIONS

a) The **forward shift row transformation**, called ShiftRows,

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



(a) Shift row transformation

- The following is an example of ShiftRows.

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

→

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

b) The **inverse shift row transformation**, called InvShiftRows,

- performs the circular hifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

RATIONALE

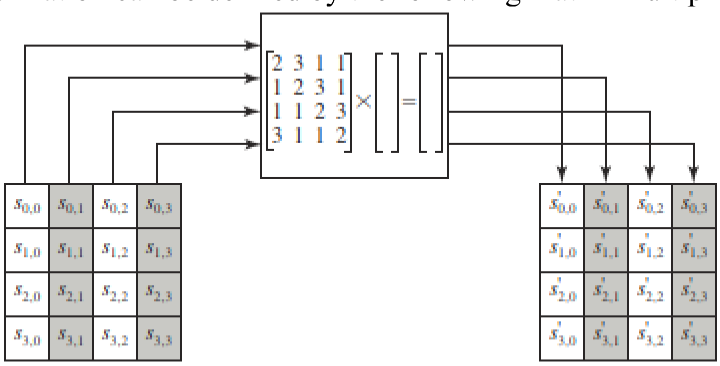
- The shift row transformation is more substantial than it may first appear.
- This is because the **State**, as well as the cipher input and output, is treated as an array of four 4-byte columns.
- Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on.
- Furthermore, as will be seen, the round key is applied to **State** column by column.
- Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes.
- The transformation ensures that the 4 bytes of one column are spread out to four different columns.

3. MixColumns Transformation

FORWARD AND INVERSE TRANSFORMATIONS

a) The **forward mix column transformation**, called MixColumns,

- operates on each column individually.
- Each byte of a column is mapped into a new value that is a function of all four bytes in that column.
- The transformation can be defined by the following matrix multiplication on **State**



(b) Mix column transformation

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

- Each element in the product matrix is the sum of products of elements of one row and one column.
- In this case, the individual additions and multiplications are performed in GF(2^8).
- The MixColumns transformation on a single column of **State** can be expressed as

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

Example of MixColumns:

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

→

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

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

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(5.5)}$$

It is not immediately clear that Equation (5.5) is the **inverse** of Equation (5.3).
Need to show

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

which is equivalent to showing

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5.6)}$$

- That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix.
- To verify the first column of Equation (5.6), need to show

$(\{0E\}.\{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\}.\{03\}) = \{01\}$
$(\{09\}.\{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\}.\{03\}) = \{00\}$
$(\{0D\}.\{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\}.\{03\}) = \{00\}$

({0B}.{02})⊕ {0D}⊕{09}⊕ ({0E}.{03}) = {00}

For the first equation,
{0E} .{02} = 00011100 and {09} .{03} = {09}⊕ ({09} .{02}) = 00001001 ⊕00010010 = 0001101

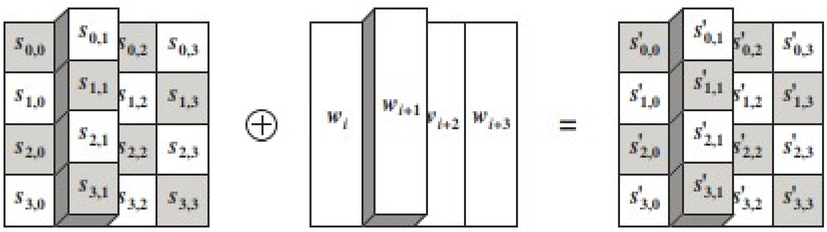
$$
\begin{array}{lcl}
\{0E\} \bullet \{02\} & = & 00011100 \\
\{0B\} & = & 00001011 \\
\{0D\} & = & 00001101 \\
\{09\} \bullet \{03\} & = & \underline{00011011} \\
 & & 00000001
\end{array}
$$

- The other equations can be similarly verified.
- The AES document describes another way of characterizing the MixColumns transformation, which is in terms of polynomial arithmetic.
- In the standard, MixColumns is defined by considering each column of State to be a four-term polynomial with coefficients in GF(2^8).
- Each column is multiplied modulo (x^4+1) by the fixed polynomial a(x) , given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad \textbf{(5.7)}$$

RATIONALE
- The coefficients of the matrix in Equation (5.3) are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column.
- The mix column transformation combined with the shift row transformation ensures that after a few rounds all output bits depend on all input bits.
- In addition, the choice of coefficients in MixColumns, which are all {01},{02} or {03}, was influenced by implementation considerations.
- Multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement.

4. Addroundkey Transformation:
FORWARD AND INVERSE TRANSFORMATIONS
a) In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key.



(b) Add round key transformation

- The operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation.
- Example of AddRoundKey:

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

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

b) The inverse add round key transformation:
- Identical to the forward add round key transformation, because the XOR operation is its own inverse.

RATIONALE :
- The add round key transformation is as simple as possible and affects every bit of **State**.
- The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

AES KEY EXPANSION

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

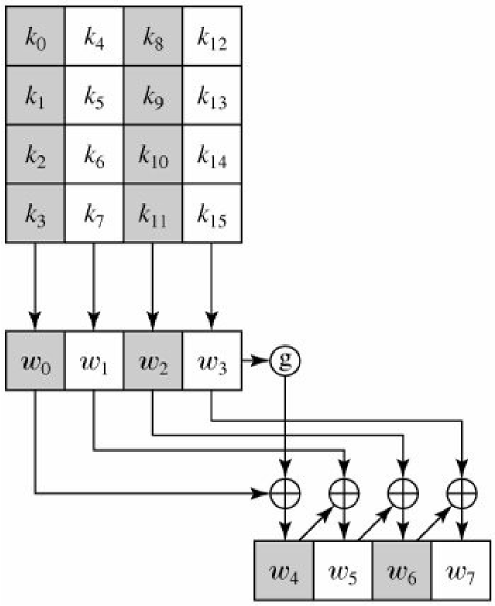
```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                      key[4*i+2],
                                      key[4*i+3]);

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

- The key is copied into the first four words of the expanded key.
- The remainder of the expanded key is filled in four words at a time.
- Each added word w[i] depends on the immediately preceding word, w[i-1] , and the word four positions back , w[i-4], .
- In three out of four cases, a simple XOR is used.
- For a word whose position in the **w** array is a multiple of 4, a more complex function is used.

The generation of the expanded key, using the symbol g to represent that complex function.

AES Key Expansion



The function g consists of the following subfunctions.

1. RotWord performs a one-byte circular left shift on a word.

- This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0] .

2. SubWord performs a byte substitution on each byte of its input word, using the S-box.

3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j] .

- The round constant is a word in which the three rightmost bytes are always 0.
- Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
- The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0),with RC[1] = 1 RC[j] = 2 . RC[j-1] , and with multiplication defined over the field GF(2^8).

The values of RC[j] in hexadecimal are

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

- Example, suppose that the round key for round 8 is

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

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

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

Rationale

- The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks.
- The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds.

Criteria:

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
- An invertible transformation [i.e., knowledge of any Nk consecutive words of the expanded key enables regeneration the entire expanded key (Nk = key size in words)].
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only.
- Simplicity of description.

Equivalent Inverse Cipher

- The AES decryption cipher is not identical to the encryption cipher.
- That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same.
- This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption.
- There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm.
- The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses).
- To achieve this equivalence, a change in key schedule is needed.

- An encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey.
- The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns.

- Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.

INTERCHANGING INVSHIFTROWS AND INVSUBBYTES

- InvShiftRows affects the sequence of bytes in **State** but does not alter byte contents and does not depend on byte contents to perform its transformation.
- InvSubBytes affects the contents of bytes in **State** but does not alter byte sequence and does not depend on byte sequence to perform its transformation.
- Thus, these two operations commute and can be interchanged.For a given **State** ,

InvShiftRows [InvSubBytes (Si)] = InvSubBytes [InvShiftRows (Si)]

INTERCHANGING ADDROUNDKEY AND INVMIXCOLUMNS

- The transformations Add- RoundKey and InvMixColumns do not alter the sequence of bytes in **State**.
- If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on **State** one column at a time.
- These two operations are linear with respect to the column input.That is, for a given **State** and a given round key ,

InvMixColumns (Si $\oplus$ wj) = [InvMixColumns (Si)] $\oplus$ [InvMixColumns (wj)]

- the first column of **State** $\mathbf{S}_i$ is the sequence (y0, y1, y2, y3) and the first column of the round key w_j is (k0, k1, k2, k3) .
- Show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

Demonstrate that for the first column entry.
Show

$$\begin{aligned} &[\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)] \\ &= [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3] \oplus \\ &\quad [\{0E\} \cdot k_0] \oplus [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3] \end{aligned}$$

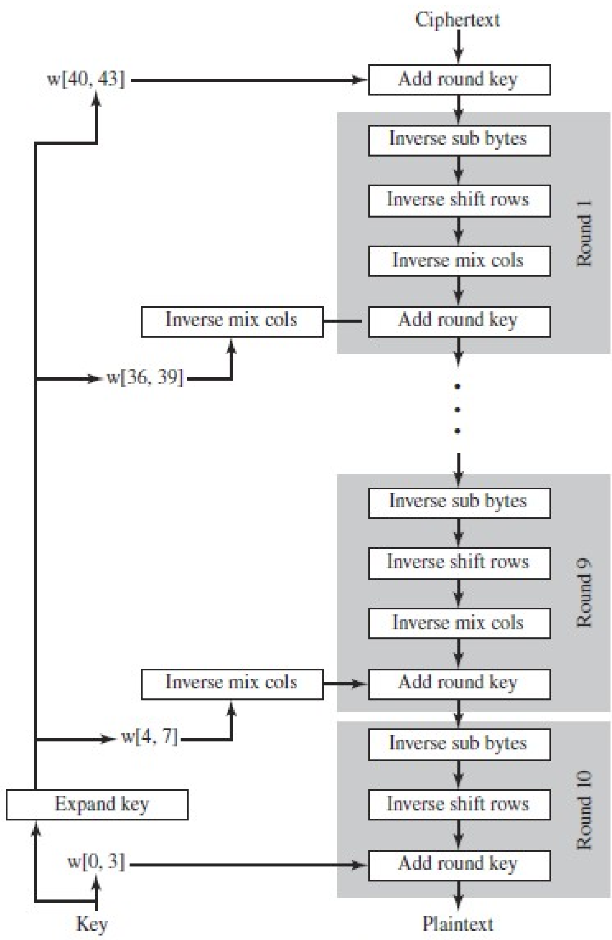


Figure 5.10 Equivalent Inverse Cipher

- can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key.
- Note that we do not need to apply InvMixColumns to the round key for the input to the first AddRoundKey transformation (preceding the first round) nor to the last AddRoundKey transformation (in round 10).
- This is because these two AddRoundKey transformations are not interchanged with InvMixColumns to produce the equivalent decryption algorithm.

Implementation Aspects

For efficient implementation on 8-bit processors, typical for current smart cards, and on 32-bit processors, typical for PCs.

8-BIT PROCESSOR

- AES can be implemented very efficiently on an 8-bit processor.
- AddRoundKey is a bytewise XOR operation.
- ShiftRows is a simple byte-shifting operation.
- SubBytes operates at the byte level and only requires a table of 256 bytes.

The transformation MixColumns requires matrix multiplication in the field $GF(2^8)$, which means that all operations are carried out on bytes. MixColumns only requires multiplication by {02} and {03}, which, as we have seen, involved simple shifts, conditional XORs, and XORs. This can be implemented in a more efficient way that eliminates the shifts and conditional XORs. Equation set (5.4) shows the equations for the MixColumns transformation on a single column. Using the identity {03}. x = ({02}. x) $\oplus$ x, we can rewrite Equation set (5.4) as follows.

$$\begin{aligned} Tmp &= s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\ s'_{0,j} &= s_{0,j} \oplus Tmp \oplus [2 \bullet (s_{0,j} \oplus s_{1,j})] \\ s'_{1,j} &= s_{1,j} \oplus Tmp \oplus [2 \bullet (s_{1,j} \oplus s_{2,j})] \\ s'_{2,j} &= s_{2,j} \oplus Tmp \oplus [2 \bullet (s_{2,j} \oplus s_{3,j})] \\ s'_{3,j} &= s_{3,j} \oplus Tmp \oplus [2 \bullet (s_{3,j} \oplus s_{0,j})] \end{aligned} \qquad \textbf{(5.9)}$$

Equation set (5.9) is verified by expanding and eliminating terms.

The multiplication by{02} involves a shift and a conditional XOR. Such an implementation may be vulnerable to a timing attack of the sort .To counter this attack and to increase processing efficiency at the cost of some storage, the multiplication can be replaced by a table lookup. Define the 256-byte table X2, such that X2[i] = {02}.i .Then Equation set (5.9) can be rewritten as

$$\begin{aligned} Tmp &= s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\ s'_{0,j} &= s_{0,j} \oplus Tmp \oplus X2[s_{0,j} \oplus s_{1,j}] \\ s'_{1,c} &= s_{1,j} \oplus Tmp \oplus X2[s_{1,j} \oplus s_{2,j}] \\ s'_{2,c} &= s_{2,j} \oplus Tmp \oplus X2[s_{2,j} \oplus s_{3,j}] \\ s'_{3,j} &= s_{3,j} \oplus Tmp \oplus X2[s_{3,j} \oplus s_{0,j}] \end{aligned}$$

32-BIT PROCESSOR

The implementation described in the preceding subsection uses only 8-bit operations. For a 32-bit processor, a more efficient implementation can be achieved if operations are defined on 32-bit words. To show this, we first define the four transformations of a round in

algebraic form. Suppose we begin with a **State** matrix consisting of elements ai, j and a round-key matrix consisting of elements ki, j .
Then the transformations can be expressed as follows.

SubBytes	$b_{i,j} = \mathrm{S}[a_{i,j}]$
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$
AddRoundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$

In the ShiftRows equation, the column indices are taken mod 4. We can combine all of these expressions into a single equation:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \mathrm{S}[a_{0,j}] \\ \mathrm{S}[a_{1,j-1}] \\ \mathrm{S}[a_{2,j-2}] \\ \mathrm{S}[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \bullet \mathrm{S}[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \bullet \mathrm{S}[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \bullet \mathrm{S}[a_{2,j-2}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \bullet \mathrm{S}[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

In the second equation, we are expressing the matrix multiplication as a linear combination of vectors.
We define four 256-word (1024-byte) tables as follows.

$T_0[x] = \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \bullet \mathrm{S}[x] \right)$	$T_1[x] = \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \bullet \mathrm{S}[x] \right)$	$T_2[x] = \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \bullet \mathrm{S}[x] \right)$	$T_3[x] = \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \bullet \mathrm{S}[x] \right)$

Thus, each table takes as input a byte value and produces a column vector (a 32-bit word) that is a function of the S-box entry for that byte value. These tables can be calculated in advance.
We can define a round function operating on a column in the following fashion.

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

As a result, an implementation based on the preceding equation requires only four table lookups and four XORs per column per round, plus 4 Kbytes to store the table.The developers of Rijndael believe that this compact, efficient implementation was probably one of the most important factors in the selection of Rijndael for AES.

2.15 RC4

→ Stream cipher structure
→ The RC4 Algorithm
  – Initialization of S
  – Stream Generation
  – Strength of RC4

Stream Cipher Structure:

* Stream cipher encrypts plaintext one byte at a time.

Stream Cipher Diagram

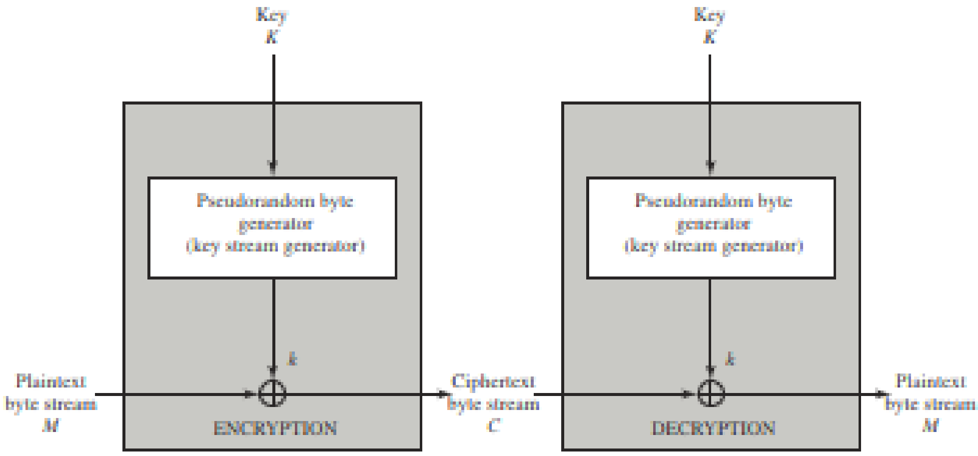


Figure 7.5 Stream Cipher Diagram

* A key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random.

→ pseudorandom stream – unpredictable without knowledge of the input key.

Key stream:
* The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

Ex: Next byte generated by the generator is 01101100
Next plaintext byte is 11001100

The resulting ciphertext byte is

	1 1 0 0 1 1 0 0	Plaintext
⊕	0 1 1 0 1 1 0 0	Key Stream
	1 0 1 0 0 0 0 0	Ciphertext

Decryption:

- same pseudorandom sequence.

	1 0 1 0 0 0 0 0	Ciphertext
⊕	0 1 1 0 1 1 0 0	Key stream
	1 1 0 0 1 1 0 0	plaintext

* stream cipher is similar to the one-time pad.

stream cipher → Pseudo random number stream.

one-time pad → uses a genuine random number stream

Design Considerations:

1. The encryption sequence should have a large period.
   → A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. (vigenere cipher)
   * The longer the keyword, the more difficult the cryptanalysis.
2. The keystream should approximate the properties of a true random number stream as close as possible.
   - equal number of 1s and 0s.
3. The output of the pseudo random generator is conditioned on the value of the input key.
   - The key needs to be sufficiently long.
   - a key length of at least 128 bits is desirable.

The RC4 Algorithm:

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key size stream cipher with byte-oriented operations.

- The algorithm is based on the use of a random permutation. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software.
- RC4 is used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers.
- It is also used in the Wired Equivalent Privacy (WEP) protocol and the newer WiFi Protected Access (WPA) protocol. RC4 was kept as a trade secret by RSA Security.
- The RC4 algorithm is remarkably simple and quite easy to explain. A vari- able length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1], Á , S[255].
- At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* (see Figure 7.5) is generated from S by selecting one of the 255 entries in a systematic fashion.
- As each value of *k* is generated, the entries in S are once again permuted.

Initialization of S

- To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is, S[0] = 0, S[1] = 1, Á , S[255] = 255 .
- A temporary vector, T, is also created. If the length of the key K is 256 bytes, then T is transferred to T.
- Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K, and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as

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

- Next we use T to produce the initial permutation of S.
- This involves starting with S[0] and going through to S[255], and for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */
    j = 0;
    for i = 0 to 255 do
        j = (j + S[i] + T[i]) mod 256;
        Swap (S[i], S[j]);
```

- Because the only operation on S is a swap, the only effect is a permutation.
- S still contains all the numbers from 0 through 255.

Stream Generation

- Once the S vector is initialized, the input key is no longer used.

- Stream generation involves cycling through all the elements of S[i], and for each S[i], swapping S[i] with another byte in S according to the current configuration of S.
- After S[255] is reached, the process continues, starting over again at S[0]:

```
/* Stream Generation */
        i, j = 0;
        while (true)
                i = (i + 1) mod 256;
                j = (j + S[i]) mod 256;
                Swap (S[i], S[j]);
                t = (S[i] + S[j]) mod 256;
                k = S[t];
```

- To encrypt, XOR the value *k* with the next byte of plaintext.
- To decrypt, XOR the value *k* with the next byte of ciphertext.

RC4 logic

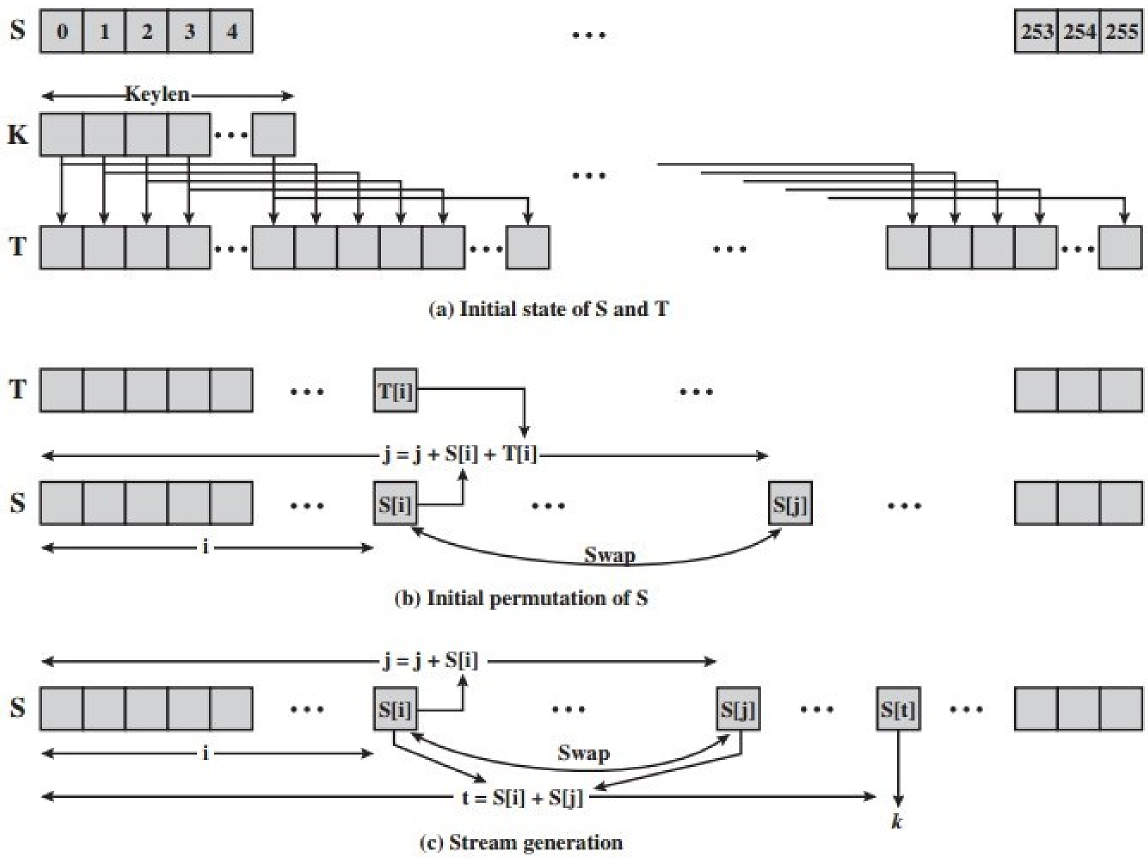


Figure 7.6 RC4

Strength of RC4

- The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach.
- In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4.
- This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated.

2.16 KEY DISTRIBUTION

Symmetric Encryption:

* The two parties to an exchange must share the same key, and that key must be protected from access by others.

→ frequent key changes are usually desirable.

Key distribution Technique:

* Delivering a key to two parties who wish to exchange data, without allowing others to see the key.

Key distribution — Number of ways:

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.

→ A Key Distribution Scenario
→ Hierarchical key control.
→ Session key lifetime
→ A Transparent key control scheme
→ Decentralized key control.
→ controlling key usage.

Link encryption → manual delivery (1, 2)

End-to-Encryption:

N/w or IP level:

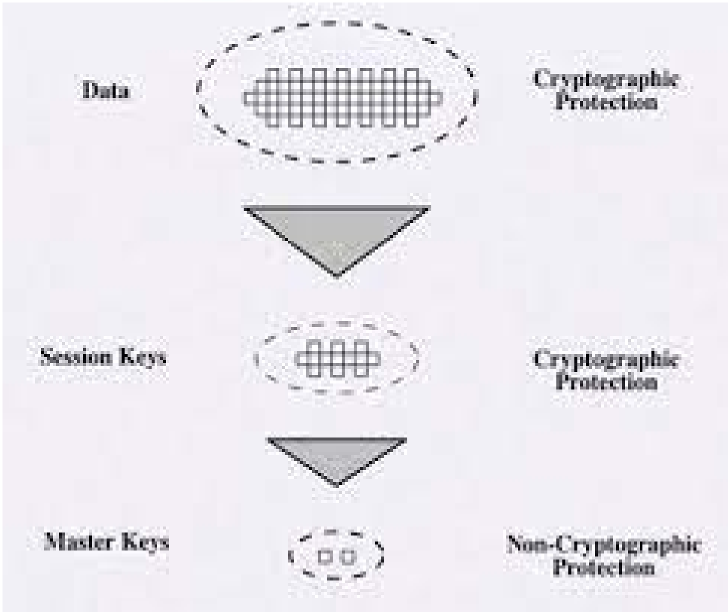
* a key is needed for each pair of hosts

N hosts,
Number of required key is $[N(N-1)]/2$.

Application level:
- Key is needed for every pair of users or processes that require communication.

The use of a key distribution center:
- based on the use of a hierarchy of keys.

Use of a Key Hierarchy



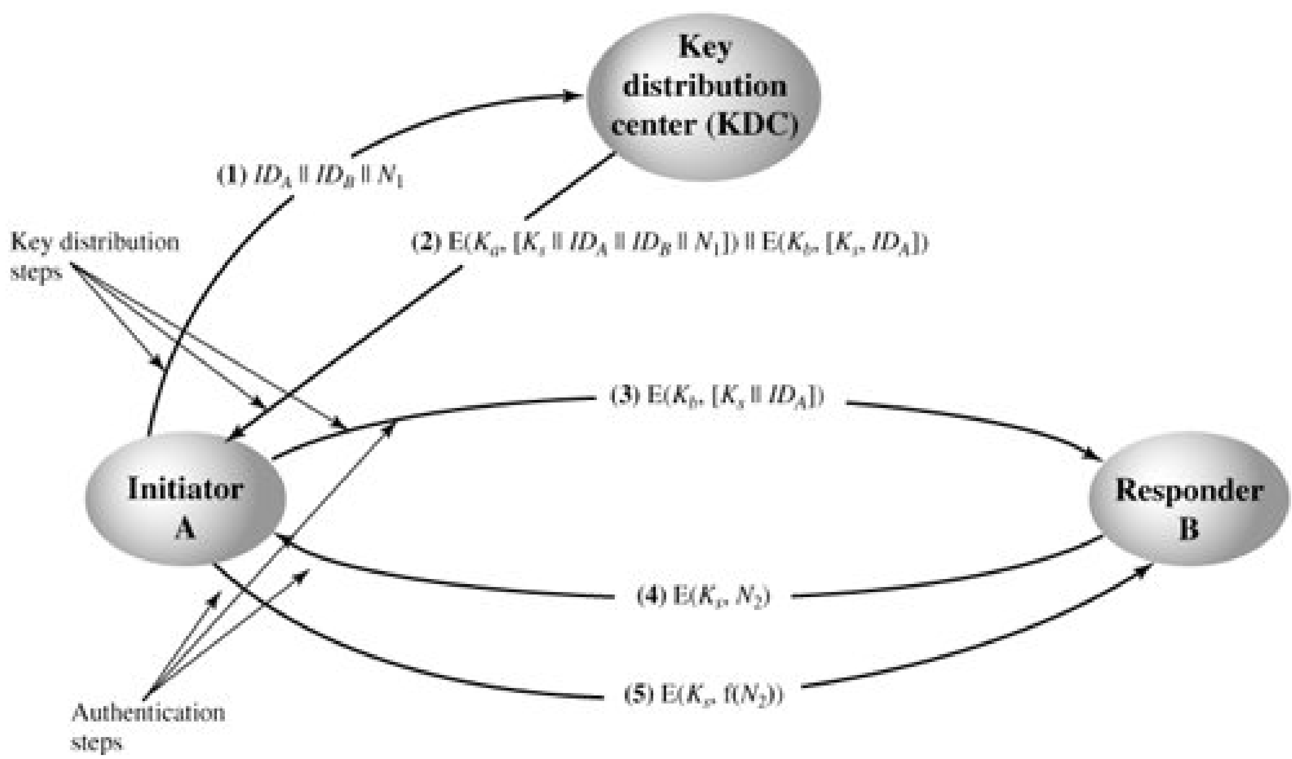
Two levels of Keys:
i) Session Key
ii) Master Key.

Session Key:
* Communication between end systems is encrypted using a temporary key.
- used for the duration of a logical connection, then discarded.
- obtained from key distribution center.

Master Key:
* Session keys are transmitted in encrypted form, using a master key that is shared by the key distribution center and an end system or user.

A Key Distribution Scenario

- The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 7.9. The scenario assumes that each user shares a unique master key with the key distribution center (KDC).



- 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. The following steps occur:

  1. A issues 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 K_a 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 stores 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.

4. Using the newly minted session key for encryption, B sends a nonce, *N2*, to A.

5. Also using K_s, A responds with f(N_2), where f is a function that performs some transformation on N_2 (e.g., adding one).

These steps assure B that the original message it received (step 3) was not a replay.

Hierarchical Key Control

- It is not necessary to limit the key distribution function to a single KDC. Indeed, for very large networks, it may not be practical to do so.
- As an alternative, a hierarchy of KDCs can be established. For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC.
- In this case, any one of the three KDCs involved can actually select the key. The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.

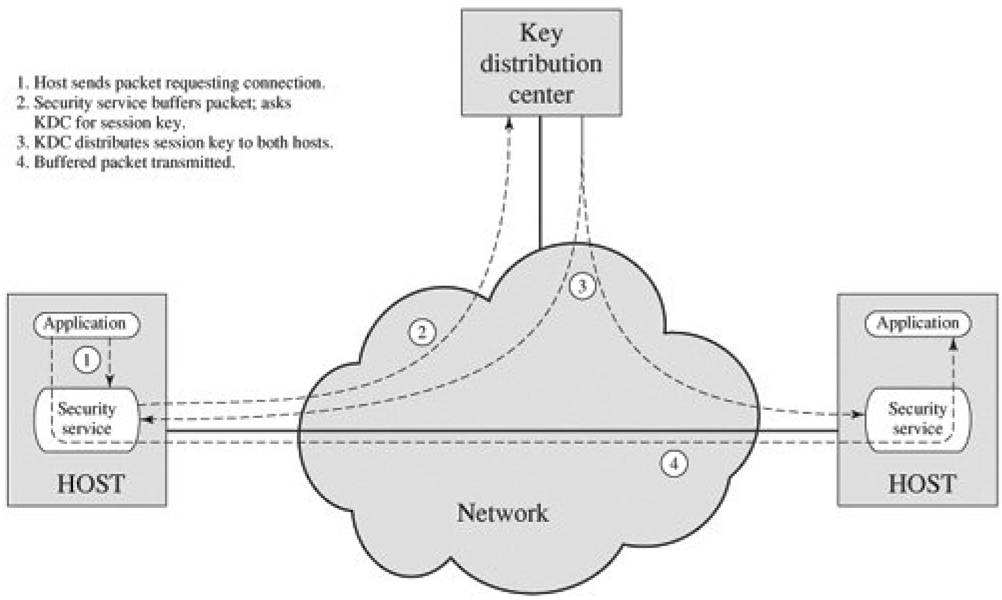
Session Key Lifetime

- The more frequently session keys are exchanged, the more secure they are, because the opponent has less ciphertext to work with for any given session key.

- 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 Transparent Key Control Scheme

- The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP. The noteworthy element of this approach is a session security module (SSM), that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.



Automatic Key Distribution for Connection-Oriented Protocol

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

Decentralized Key Control

- A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution.

- Thus, there may need to be as many as [*n*(*n* -1)]/2 master keys for a configuration with *n* end systems. A session key may be established with the following sequence of steps (Figure 7.11):

1. A issues a request to B for a session key and includes a nonce, *N1*

2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value f(*N*1), and another nonce, *N*2.

3. Using the new session key, A returns f(*N*2) to B.

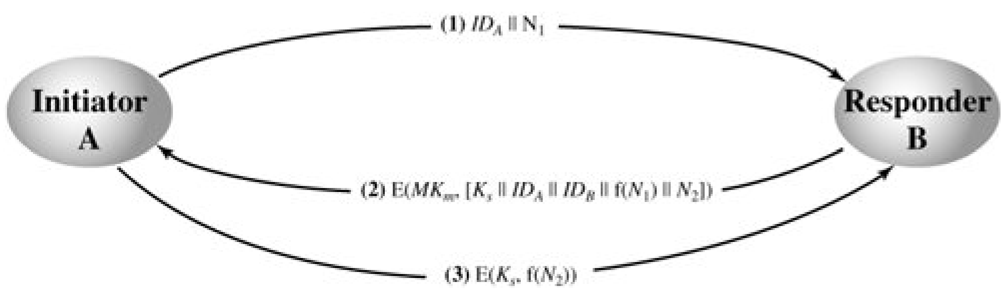


Figure 7.11. Decentralized Key Distribution

Thus, although each node must maintain at most (*n*- 1) master keys, as many session keys as required may be generated and used. Because the messages transferred using the master key are short, cryptanalysis is difficult.

Controlling Key Usage

- The concept of a key hierarchy and the use of automated key distribution techniques greatly reduce the number of keys that must be manually managed and distributed.
- different types of session keys such as
  - ✓ Data-encrypting key, for general communication across a network

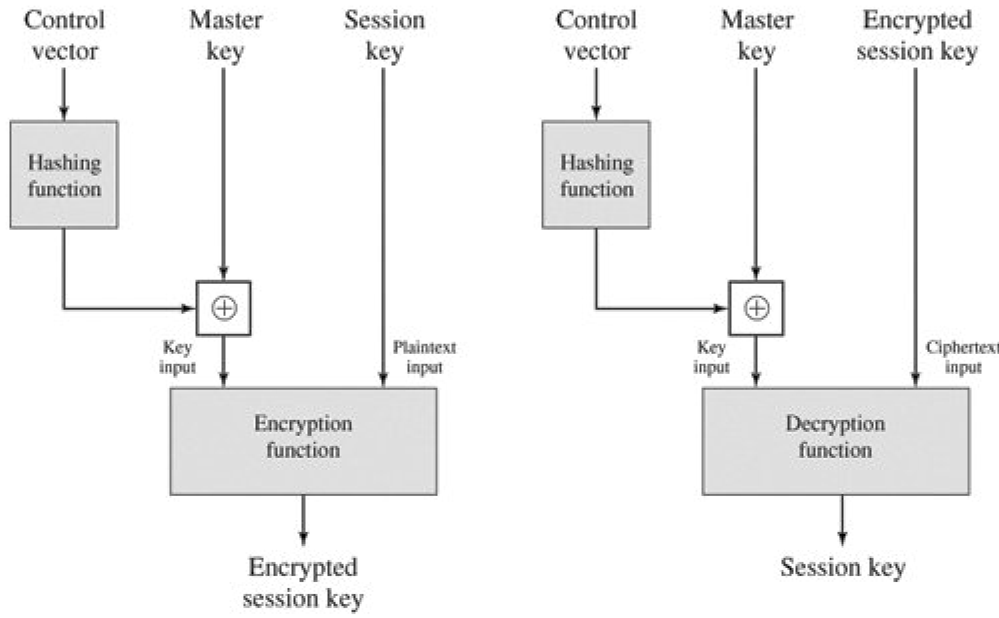
- ✓ PIN-encrypting key, for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
- ✓ File-encrypting key, for encrypting files stored in publicly accessible locations

- Normally, the master key is physically secured within the cryptographic hardware of the key distribution center and of the end systems.
- Session keys encrypted with this master key are available to application programs, as are the data encrypted with such session keys.
- The proposed technique is for use with DES and makes use of the extra 8 bits in each 64-bit DES key.
- That is, the 8 nonkey bits ordinarily reserved for parity checking form the

key tag. The bits have the following interpretation:

- ✓ One bit indicates whether the key is a session key or a master key.
- ✓ One bit indicates whether the key can be used for encryption.
- ✓ One bit indicates whether the key can be used for decryption.
- ✓ The remaining bits are spares for future use.

- In this scheme, each session key has an associated control vector consisting of a number of fields that specify the uses and restrictions for that session key.
- The length of the control vector may vary. The control vector is cryptographically coupled with the key at the time of key generation at the KDC. The coupling and decoupling processes are illustrated in Figure 7.12.
- As a first step, the control vector is passed through a hash function that produces a value whose length is equal to the encryption key length.



(a) Control Vector Encryption (b) Control Vector Decryption

Control Vector Encryption and Decryption

- The hash value is then XORed with the master key to produce an output that is used as the key input for encrypting the session key. Thus,

  Hash value = H = h(CV)

  Key input = K_m H

  Ciphertext = E([K_m H], K_s)

where K_m is the master key and K_s is the session key. The session key is recovered in plaintext by the reverse operation:

$$D([Km\ H], E([Km\ H], Ks))$$

When a session key is delivered to a user from the KDC, it is accompanied by the control vector in clear form.

- The session key can be recovered only by using both the master key that the user shares with the KDC and the control vector. Thus, the linkage between the session key and its control vector is maintained.