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.

MATHEMATICS OF SYMMETRIC KEY CRYPTOGRAPHY
2.2. MODULAR ARITHMETIC

Contents
• **The Modulus** • **Properties of Congruences** • **Modular Arithmetic Operations** • **Properties of Modular Arithmetic**

The Modulus

- If a is an integer and n is a positive integer, we define a mod n to be the remainder when a is divided by n. The integer n is called the modulus. Thus, for any integer a, we can rewrite Equation (4.1) as follows:

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

$$11 \bmod 7 = 4; \qquad -11 \bmod 7 = 3$$

- Two integers a and b are said to be congruent modulo n, if (a mod n) = (b mod n). This is written as a K b (mod n).

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

Note that if $a \equiv 0 \pmod{n}$, then $n \mid a$.

Properties of Congruences

Congruences have the following properties:

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

To demonstrate the first point, if n | (a - b), then (a - b) = k*n* for some k.

- So when b is divided by n) = (b mod n).we can write a = b + k*n*. Therefore, (a mod n) = (remainder when b + k*n* is divided by n) = remainder

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

The remaining points are as easily proved.

Modular Arithmetic Operations

❖ Note that, by definition (Figure 4.1), the (mod n) operator maps all integers into the set of integers {0, 1, c, (n - 1)}. this technique is known as modular arithmetic.

Modular arithmetic exhibits the following properties:

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

Properties of Modular Arithmetic

Define the set Zn as the set of nonnegative integers less than n:

$$Z_n = \{0, 1, \ldots, (n - 1)\}$$

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

The residue classes (mod 4) are

$[0] = \{\ldots, -16, -12, -8, -4, 0, 4, 8, 12, 16, \ldots\}$

$[1] = \{\ldots, -15, -11, -7, -3, 1, 5, 9, 13, 17, \ldots\}$

Table 4.3 Properties of Modular Arithmetic for Integers in Z_n

Property	Expression
Commutative Laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative Laws	$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$ $[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$
Distributive Law	$[w \times (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 \times w) \bmod n = w \bmod n$
Additive Inverse $(-w)$	For each $w \in Z_n$, there exists a z such that $w + z = 0 \bmod n$

2.4. EUCLID'S ALGORITHM

Contents
• **Introduction** • **Greatest Common Divisor** • **Finding the Greatest Common Divisor**

Introduction

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

Greatest Common Divisor:

- Recall that nonzero b is defined to be a divisor of a if $a = mb$ for some m, where a, b, and m are integers.
- We will use the notation gcd(a, b) to mean the **greatest common divisor** of a and b. The greatest common divisor of a and$b$ is the largest integer that divides both a and b.
- We also define gcd(0, 0) = 0.More formally, the positive integer c is said to be the greatest common divisor of a and b if
    1. c is a divisor of a and of b.
    2. Any divisor of a and b is a divisor of c.

An equivalent definition is the following:

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

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

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

Finding the Greatest Common Divisor

- Suppose we have integers a, b such that d = gcd(a, b). Because gcdgcd(|a| , |b|) = gcd(a, b), there is no harm in assuming a ≥ b > 0. Now dividing a by b and applying the division algorithm, we can state:

$$a = q_1 b + r_1 \qquad 0 \le r_1 < b \qquad \textbf{(4.2)}$$

- Let us now return to Equation (4.2) and assume that $r_1 \neq 0$. Because $b > r_1$, we can divide b by r_1 and apply the division algorithm to obtain:

$$b = q_2 r_1 + r_2 \qquad 0 \le r_2 < r_1$$

The result is the following system of equations:

$$\left.\begin{array}{ll} a = q_1 b + r_1 & 0 < r_1 < b \\ b = q_2 r_1 + r_2 & 0 < r_2 < r_1 \\ r_1 = q_3 r_2 + r_3 & 0 < r_3 < r_2 \\ \quad\vdots & \quad\vdots \\ r_{n-2} = q_n r_{n-1} + r_n & 0 < r_n < r_{n-1} \\ r_{n-1} = q_{n+1} r_n + 0 & \\ d = \gcd(a, b) = r_n & \end{array}\right\} \qquad \textbf{(4.3)}$$

Let us now look at an example with relatively large numbers to see the power of this algorithm:

To find d = gcd (a,b) = gcd (1160718174, 316258250)		
$a = q_1 b + r_1$	1160718174 = 3 × 316258250 + 211943424	d = gcd(316258250, 211943424)
$b = q_2 r_1 + r_2$	316258250 = 1 × 211943424 + 104314826	d = gcd(211943424, 104314826)
$r_1 = q_3 r_2 + r_3$	211943424 = 2 × 104314826 + 3313772	d = gcd(104314826, 3313772)
$r_2 = q_4 r_3 + r_4$	104314826 = 31 × 3313772 + 1587894	d = gcd(3313772, 1587894)
$r_3 = q_5 r_4 + r_5$	3313772 = 2 × 1587894 + 137984	d = gcd(1587894, 137984)
$r_4 = q_6 r_5 + r_6$	1587894 = 11 × 137984 + 70070	d = gcd(137984, 70070)
$r_5 = q_7 r_6 + r_7$	137984 = 1 × 70070 + 67914	d = gcd(70070, 67914)
$r_6 = q_8 r_7 + r_8$	70070 = 1 × 67914 + 2156	d = gcd(67914, 2156)
$r_7 = q_9 r_8 + r_9$	67914 = 31 × 2516 + 1078	d = gcd(2156, 1078)
$r_8 = q_{10} r_9 + r_{10}$	2156 = 2 × 1078 + 0	d = gcd(1078, 0) = 1078
Therefore, d = gcd(1160718174, 316258250) = 1078		

In this example, we begin by dividing 1160718174 by 316258250, which gives 3 with a remainder of 211943424. Next we take 316258250 and divide it by 211943424. The process continues until we get a remainder of 0, yielding a result of 1078.

2.5. CONGRUENCE AND MATRICES

2.6. GROUPS, RINGS, FIELDS

Contents

- **Groups**
  - A1- Closure
  - A2 - Associative
  - A3 - Identity
  - A4 - Inverse
  - A5 - Commutative
- **Rings**
  - M1- Closure under multiplication
  - M2 - Associativity of multiplication
  - M3 - Distributive law
  - M4 – Commutativity of multiplication
  - M5 – Multiplicative Identity
  - M6 – No zero divisors
- **Fields**
  - M7 – Multiplicative Inverse

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

Groups

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

(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:	There is an element e in G such that $a \cdot e = e \cdot a = a$ for all a in G.
(A4) Inverse element:	For each a in G, there is an element a' in G such that $a \cdot a' = a' \cdot a = e$.

- If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**.
- A group is said to be **abelian** if it satisfies the following additional condition:

(A5) Commutative:	$a \cdot b = b \cdot a$ for all a, b in G.

Rings

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

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

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

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

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

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

Fields:

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

(A1–M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.

(M7) Multiplicative inverse: For each a in F, except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.

2.7. FINITE FIELDS

SYMMETRIC KEY CIPHERS

2.8. SDES

Contents

Introduction

- Proposed by NIST in 1977.
- It is a block cipher and encrypts 64 bits data using 56 bit key.

DES Encryption:

- 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 key is 56** in length.
- Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in **three phases.**
- First, the 64-bit plaintext passes through an **initial permutation (IP)** that rearranges the bits to produce the **permuted input.**
- This is followed by a phase consisting of **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 **pre output.**
- Finally, the preoutput is passed through a permutation [IP -1] that is the inverse of the initial permutation function, to produce the 64-bit **ciphertext.** With the exception of the initial and final permutations, DES has the exact structure of a **Feistel Cipher.**

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

DES Decryption

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

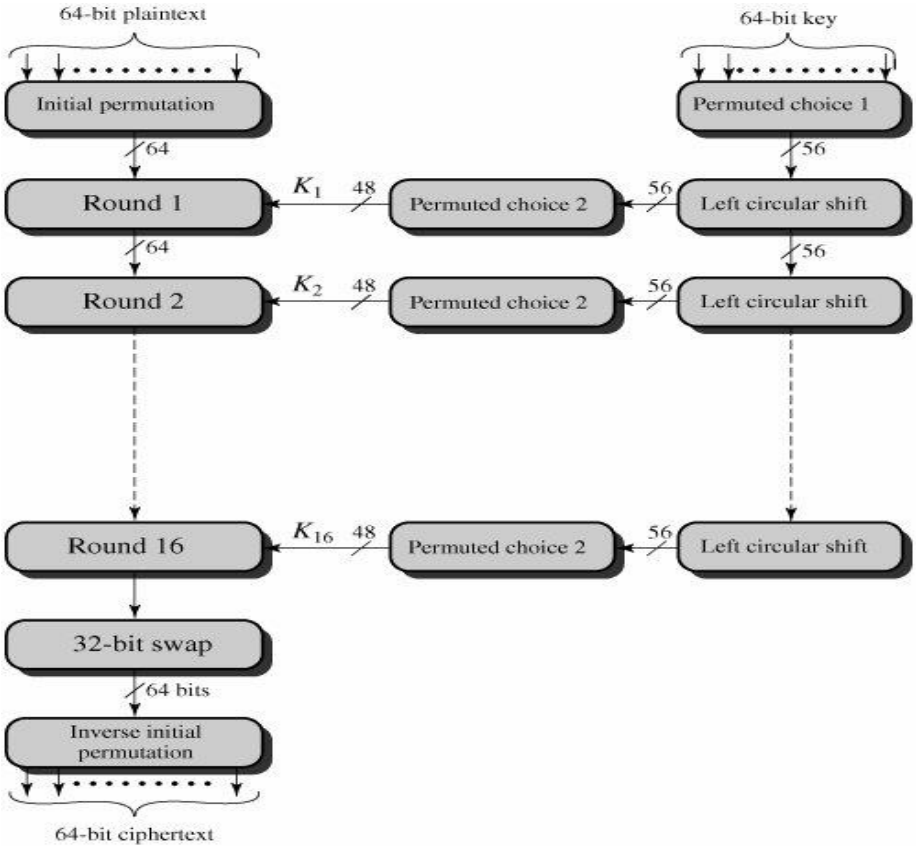


Fig .General Depiction of DES Encryption Algorithm

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 cipher text.
- 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 cipher text.
- This is referred to as the avalanche effect.

DES Round structure.

- ✓ Uses two 32 bit L & R halves.
- ✓ As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

Li = Ri-1
Ri= Li 1 { F(Ri 1, Ki)

- ✓ 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.
- ✓ The resulting 48 bits are XOR ed with. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by table.
- ✓ The role of the S-boxes in the function F is illustrated in Figure.
- ✓ The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and Produces 4 bits as output.

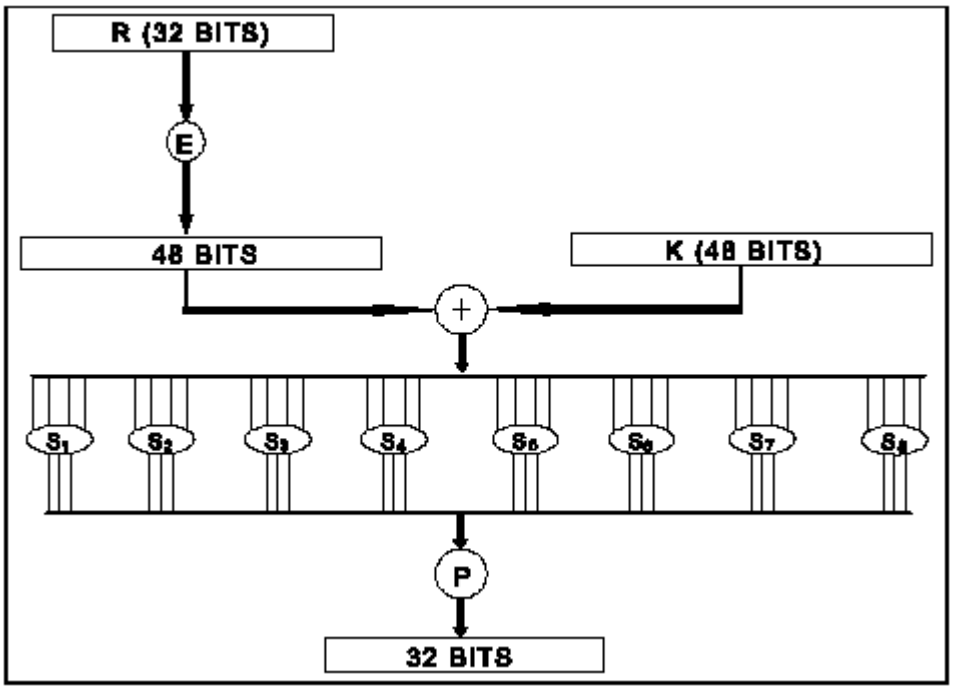


Fig: Calculation of F(R, K)

2.9. STRENGTH OF DES

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

2.9. DIFFERENTIAL AND LINEAR CRYPTANALYSIS

2.10. BLOCK CIPHER DESIGN PRINCIPLES

– Block cipher Principles of DES

Contents
• Introduction • Number of rounds • Design of function • Key Schedule Algorithm

Introduction

- There are three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling

Number of Rounds

- The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F.
- In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple **brute-force key search attack.**
- The differential cryptanalysis attack requires 255.1 operations, whereas brute force requires 255.
- If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than a brute-force key search.
- This criterion is attractive, because it makes it easy to judge the strength of an algorithm and to compare different algorithms.
- In the absence of a cryptanalytic breakthrough, the strength of any algorithm that satisfies the criterion can be judged solely on key length.

Design of Function F

- The heart of a **Feistel block cipher** is the function F, which provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "**unscramble**" the substitution performed by F.
- One obvious criterion is that F be **nonlinear**. The more nonlinear F, the more difficult any type of cryptanalysis will be.
- The more difficult it is to approximate F by a set of linear equations, the more nonlinear F is. Several other criteria should be considered in designing F.
- We would like the algorithm to have good avalanche properties.
- A more stringent version of this is the **strict avalanche criterion (SAC)**, which states that any output bit *j* of an S-box should change with probability 1/2 when any single input bit *i* is inverted for all *i*, *j*.

- Although SAC is expressed in terms of S-boxes, a similar criterion could be applied to F as a whole. This is important when considering designs that do not include S-boxes.
- Another criterion proposed is the **bit independence criterion (BIC)**, which states that output bits *j* and *k* should change independently when any single input bit *i* is inverted for all *i*, *j*, and *k*. The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

Key Schedule Algorithm

- With **any Feistel block cipher**, the key is used to generate one subkey for each round.
- In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key.
- No general principles for this have yet been promulgated.
- At minimum, the key schedule should **guarantee key/ciphertext Strict Avalanche Criterion** and **Bit Independence Criterion.**

2.11. BLOCK CIPHER MODE OF OPERATION

Contents

- Electronic Code Book
- Cipher Block Chaining Mode
- Cipher Feedback Mode
- Output Feedback Mode
- Counter Mode

Electronic Code Book

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

Table 6.1 Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 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 block of plaintext and the preceding block 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

- For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary.
- **Decryption is performed one block at a time**, always using the same key.
- We can define ECB mode as follows.

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

- The ECB method is ideal for a **short amount of data**, such as an encryption key.
- For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities.

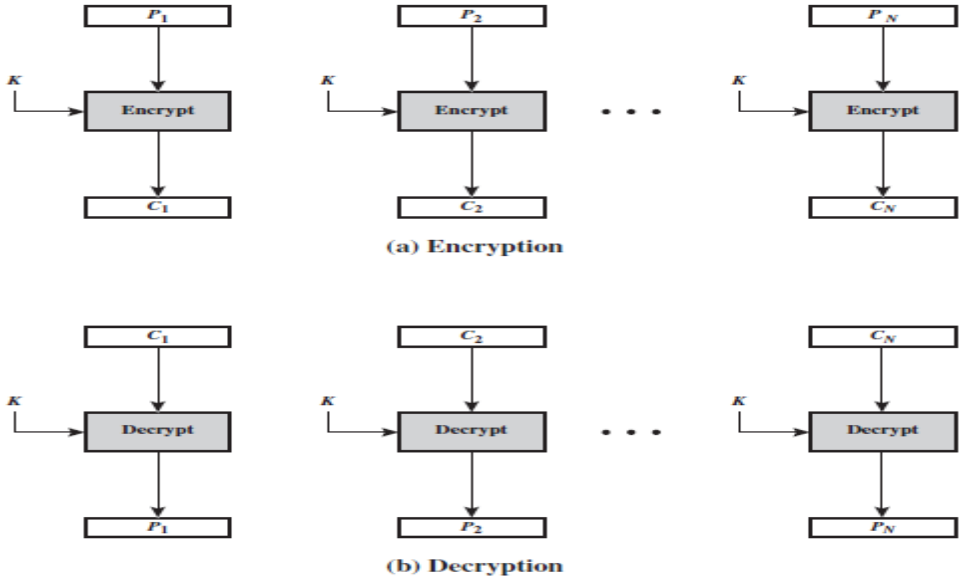


Figure 6.3 Electronic Codebook (ECB) Mode

Cipher Block Chaining Mode

- In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block.
- In effect, we have chained together the processing of the sequence of plaintext blocks.
- The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.
- The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

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

Then

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

- 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.
- We can define CBC mode as

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

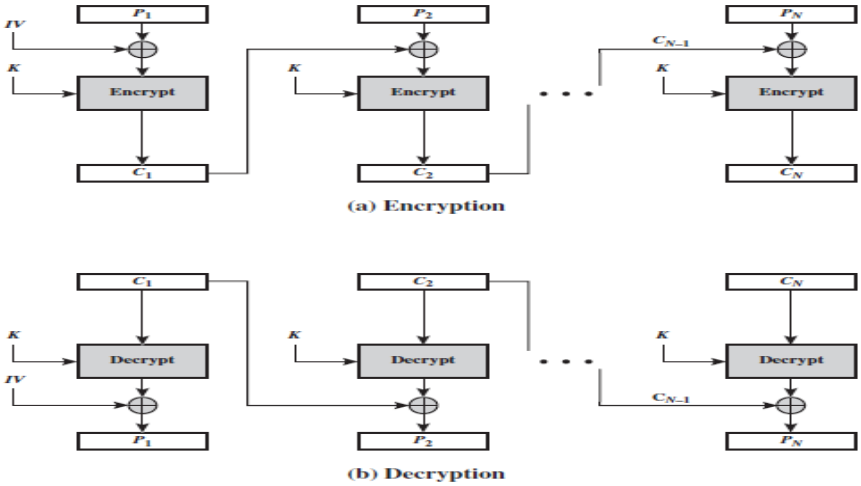


Figure 6.4 Cipher Block Chaining (CBC) Mode

Cipher Feedback Mode

- As with CBC, the units of plaintext are chained together, so that the cipher text of any plaintext unit is a function of all the preceding plaintext.
- In this case, rather than blocks of b bits, the plaintext is divided into segments of s bits.
- First, consider encryption. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV).
- **The leftmost (most significant) s** bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C1, which is then transmitted.
- In addition, the contents of the shift register are shifted left by s bits, and C1 is placed in the rightmost (least significant) s bits of the shift register.
- This process continues until all plaintext units have been encrypted. For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit
- This is easily explained. Let MSBs(X) be defined as the **most significant s bits of X.** Then

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

Therefore, by rearranging terms:

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

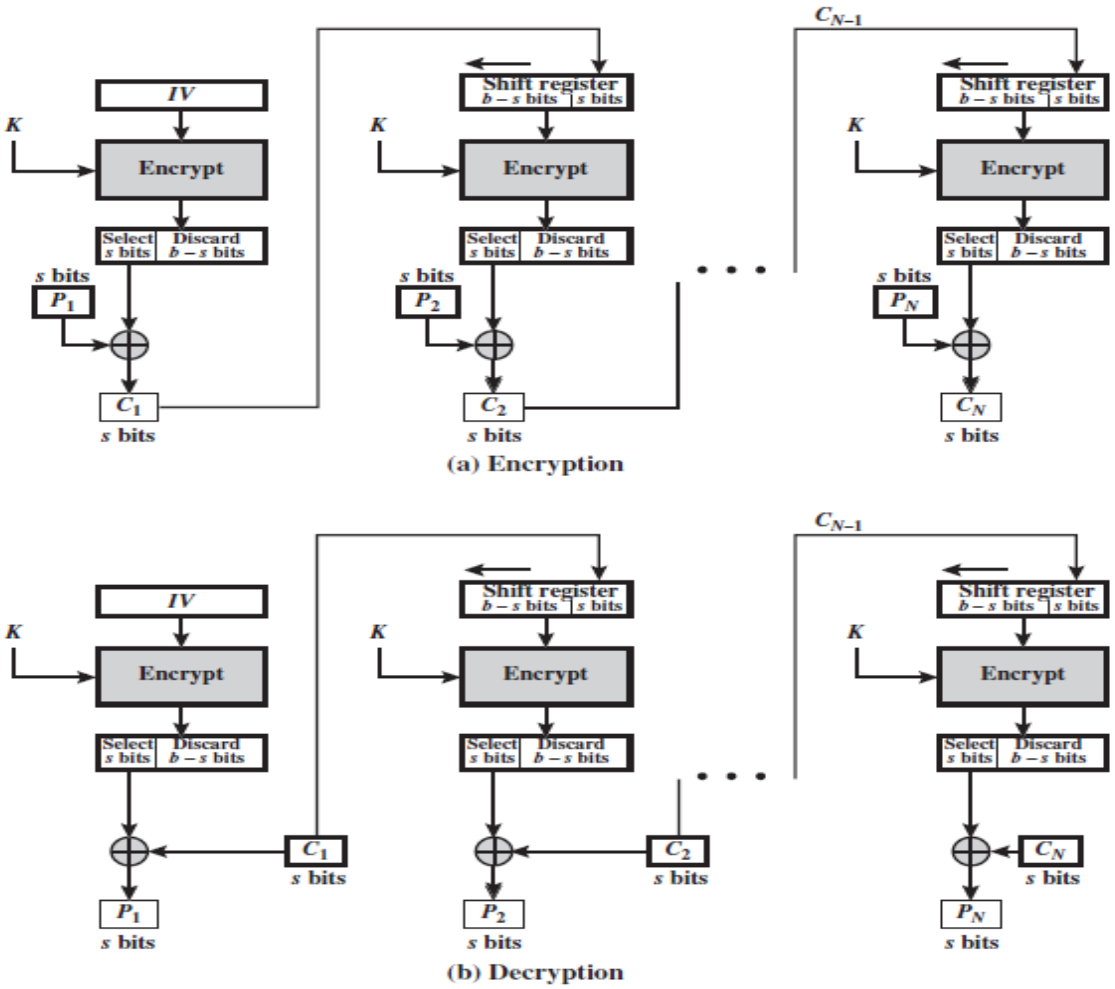


Figure 6.5 *s*-bit Cipher Feedback (CFB) Mode

We can define CFB mode as follows.

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

Output feedback (OFB) mode

- The output feedback (OFB) mode is similar in structure to that of CFB. For OFB, the output of the encryption function is feed back to become the input for encrypting the next block of plaintext (Figure 6.6).
- The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, whereas CFB operates on an s-bit subset. OFB encryption can be expressed as

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

where

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

Some thought should convince you that we can rewrite the encryption expression as:

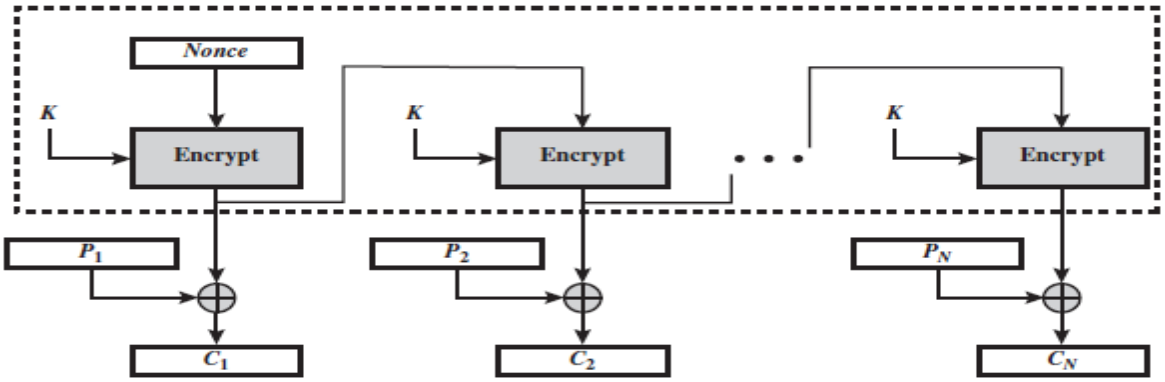
$$C_j = P_j \oplus \mathrm{E}(K, [C_{j-1} \oplus P_{j-1}])$$

By rearranging terms, we can demonstrate that decryption works.

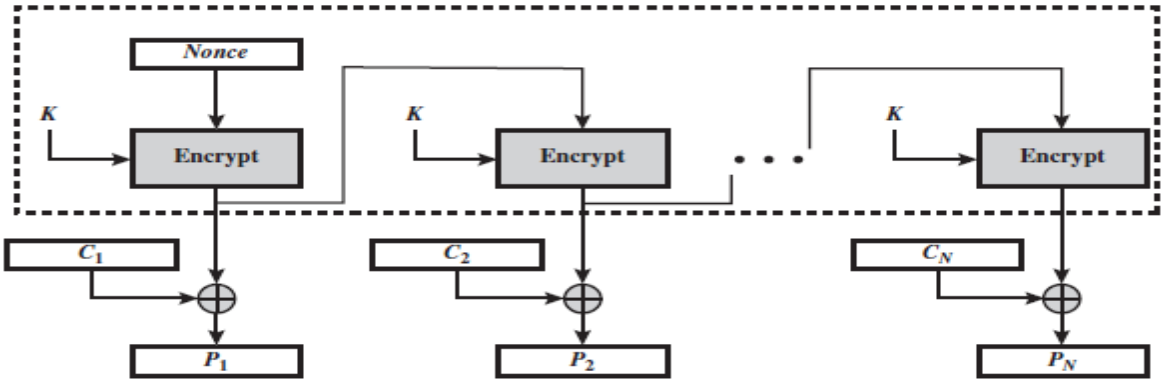
$$P_j = C_j \oplus \mathrm{E}(K, [C_{j-1} \oplus P_{j-1}])$$

We can define OFB mode as follows.

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



(a) Encryption



(b) Decryption

Figure 6.6 Output Feedback (OFB) Mode

Counter Mode

- Although interest in the **counter (CTR**) mode has increased recently with applications to ATM **(asynchronous transfer mode) network security and IP sec (IP security**), this mode was proposed early on (e.g., [DIFF79]).
- Figure 6.7 depicts the CTR mode. A counter equal to the plaintext block size is used.
- Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2b, where b is the block size).
- For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining.
- For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block. Thus, the initial counter value must be made available for decryption. Given a sequence of counters T1, T2, c, TN, we can define CTR mode as follows.

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

2.12. EVALUATION CRITERIA FOR AES

2.13. ADVANCED ENCRYPTION STANDARD

Finite Field Arithmetic

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

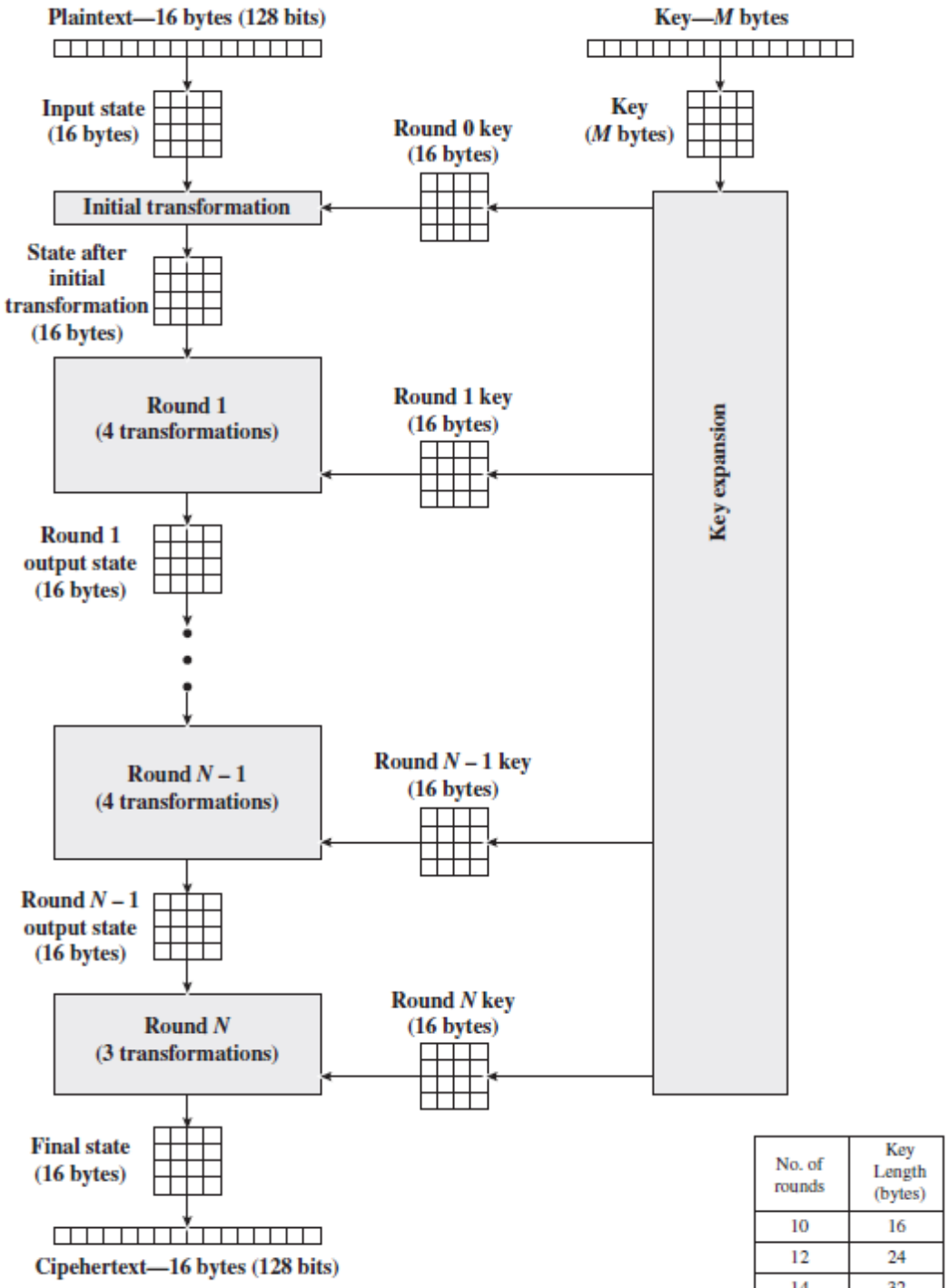
AES Structure

- General Structure
- Detailed Structure

General Structure

- Figure(5.1). shows the overall structure of the **AES encryption process**.
- The cipher takes a plaintext block size of **128 bits, or 16 bytes**.
- The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

- The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a 4 * 4 square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of **encryption or decryption**.
- After the final stage, **State** is copied to an output matrix.
- This key is then expanded into an array of key schedule words.
- Each word is four bytes, and the total key schedule is 44 words for the 128-bit key
- The **cipher** consists of *N* rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key.
- The first *N* - 1 rounds consist of four distinct transformation functions: **SubBytes, ShiftRows, MixColumns, and AddRoundKey**, which are described subsequently.
- The final round contains only three transformations, and there is a initial single transformation **(AddRoundKey)** before the first round, which can be considered Round 0. Each transformation takes one or more 4 * 4 matrices as input and produces a 4 * 4 matrix as output.



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

Fig : AES Encryption Process

Table 5.1 AES Parameters

Key Size (words/bytes/bits)
Plaintext Block Size (words/bytes/bits)
Number of Rounds
Round Key Size (words/bytes/bits)
Expanded Key Size (words/bytes)

Detailed Structure

Figure (5.1) shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function.

We can make several comments about the overall AES structure.

1. One noteworthy feature of this structure is that it **is not a Feistel structure**
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; these are indicated in Figure 5.**3**

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 GF(28)

• **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded Key.

4. The structure is **quite simple**.

5. Only the **AddRoundKey** stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage.

6. The AddRoundKey stage is, in effect, 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.

7. Each stage is easily reversible.

8. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.

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

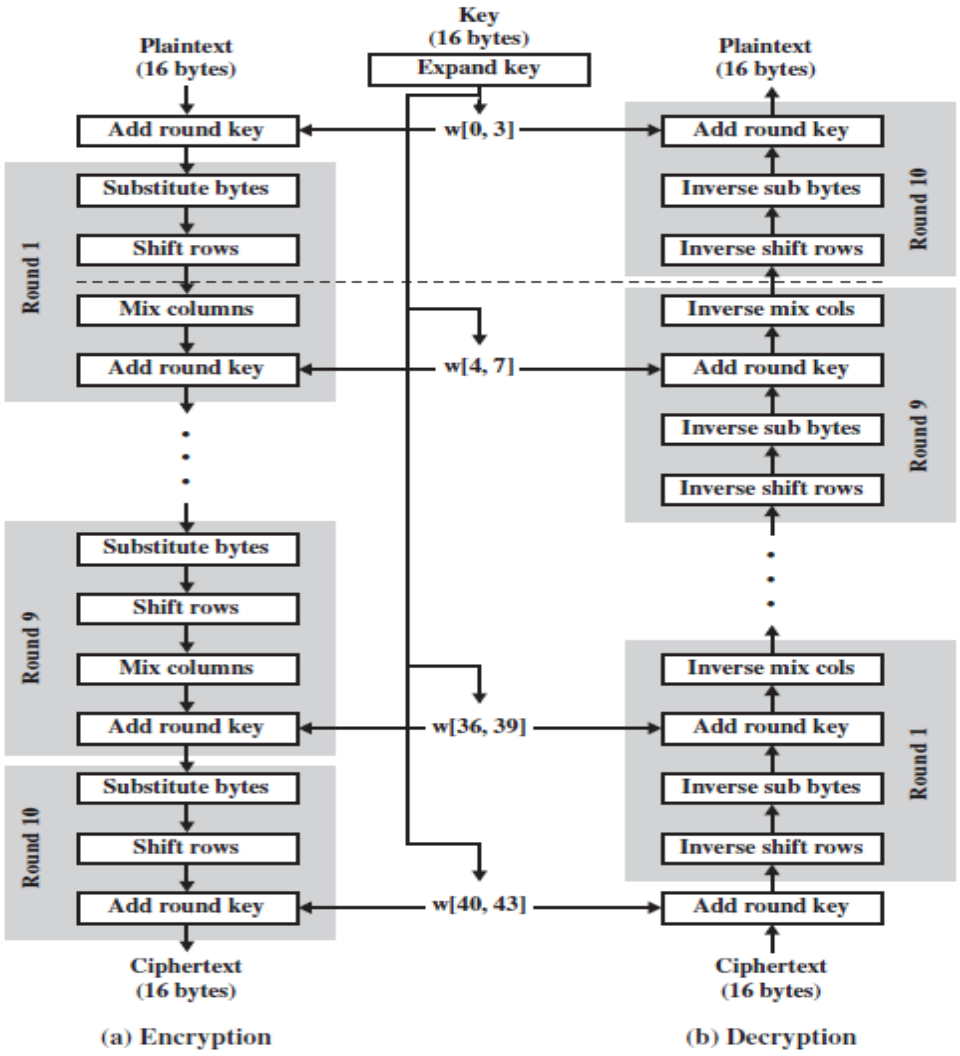


Fig (5.3): AES Encryption and Decryption

AES Transformation Functions

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

- Substitute Bytes Transformation
- Shift Rows Transformation
- Mix Columns Transformation
- AddRoundKey Transformation

Substitute Bytes Transformation

- The **forward substitute byte transformation**, called SubBytes, is a simple table lookup. AES defines a 16 * 16 matrix of byte values, called **an S-box**, that contains a permutation of all possible 256 8-bit values.
- Each individual byte of **State is mapped into a new byte in the following way:**

- The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.
- These row and column values serve as indexes into the S-box to select a unique 8-bit output value.

Shift Rows Transformation

- The first row of State is not altered.
- For the second row, a 1-byte circular left shift is performed.
- For the third row, a 2-byte circular left shift is performed.
- For the fourth row, a 3-byte circular left shift is performed.
- The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

MixColumns Transformation

- 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.

AddRoundKey Transformation

- AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key.

2.14. RC4

Contents
• **Characteristics** • **RC5 Parameters** • **Key Expansion** • **Encryption** • **Decryption** • **RC5 Modes**

RC5 is a symmetric encryption algorithm developed by Ron Rivest. RC5 was designed to have the **following characteristics:**

- **Suitable for hardware or software**
- **Fast**
- **Adaptable to processors of different word lengths**
- **Variable number of rounds:**
- **Variable-length key**

- **Simple** .
- **Low memory requirement**
- **High security**

❖ RC5 has been incorporated into RSA Data Security, Inc-'s major products, including BSAFE, JSAFE, and S/MAIL.

RC5 Parameters

RC5 is actually a family of encryption algorithms determined by three parameters, as follows:

Parameter	Definition	Allowable Values
w	Word size in bits. RC5 encrypts 2-word blocks	16, 32, 64
r	Number of rounds	0, 1, ..., 255
b	Number of 8-bit bytes (octets) in the secret key K	0, 1, ..., 255

Key Expansion

❖ RC5 performs a complex set of operations on the secret key to produce a total of *t* subkeys. Two subkeys are used in each round, and two subkeys are used on an additional operation that is not part of any round, so $t = 2r + 2$. Each subkey is one Word (w bits) in length.

❖ Figure 4-11 illustrates the technique used to generate subkeys; The subkeys are stored in a *t*-word array labeled S[0], S[1], ….,' S[t-1]. Using the parameters *r* and *w* as inputs, this array is initialized to a particular fixed pseudorandom bit pattern.

❖ Then the *b*-byte key, K[0…. *b* - 1], is converted into a *c*-word array L[0…. *c -1*]. On a little endian machine, this is accomplished by zeroing out the array *L* and copying the string *K* directly into the memory positions represented by L.

❖ If *b* is not an integer multiple of w, then a portion of L at the right end remains zero-Finally, a mixing operation is performed that applies the contents of *L* to the initialized value of S to produce a final value for the array S.

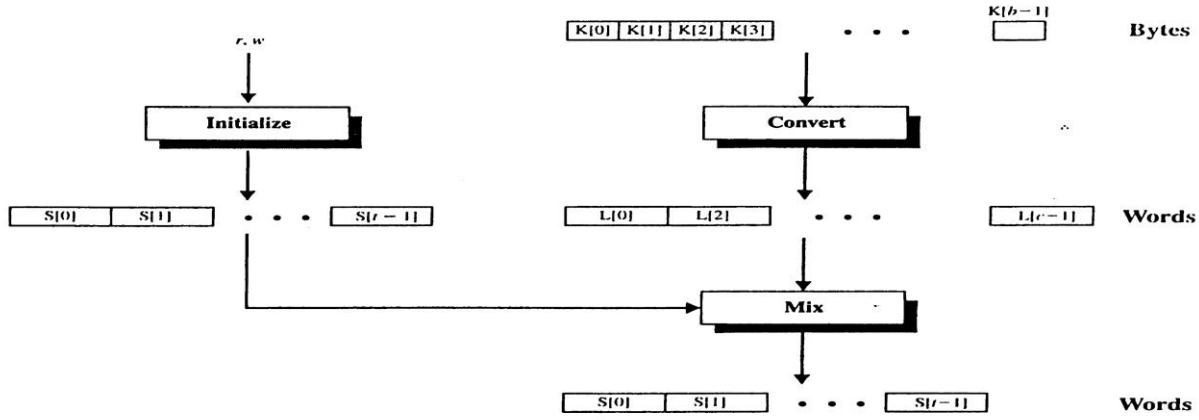


Figure 4.11 RC5 Key Expansion.

Let us look at this operations in detail. The initialize operation makes use of two word-length constants defined as follows,

$$P_w = \text{Odd}[(e-2)2^w]$$
$$Q_w = \text{Odd}[(\phi-1)2^w]$$

Where

$e = 2.718281828459 \ldots$ (base of natural logarithms)

$\phi = 1.618033988749 \ldots$ (golden ratio[3]) $= \left(\frac{1+\sqrt{5}}{2}\right)$

Encryption:

- RC5 uses three primitive operations (and their inverses):
  - **Addition:** Addition of words, denoted by +, is performed modulo 2^w. The inverse operation, denoted by -, is subtraction modulo 2^w.
  - **Bitwise exclusive-OR:** This operation is denoted by $\oplus$.
  - **Left circular rotation:** The cyclic rotation of word x left by y bits is denoted by $x <<< y$. The inverse is the right circular rotation of word x by y bits, denoted by $x >>> y$.

Figure 4-12a depicts the encryption operation. Note that this is not a classic Feistel structure. The plaintext is assumed to initially reside in the two w-bit registers A and B.

We use the variables LE_i and RE_i to refer to the left and right half of the data after round i has completed.

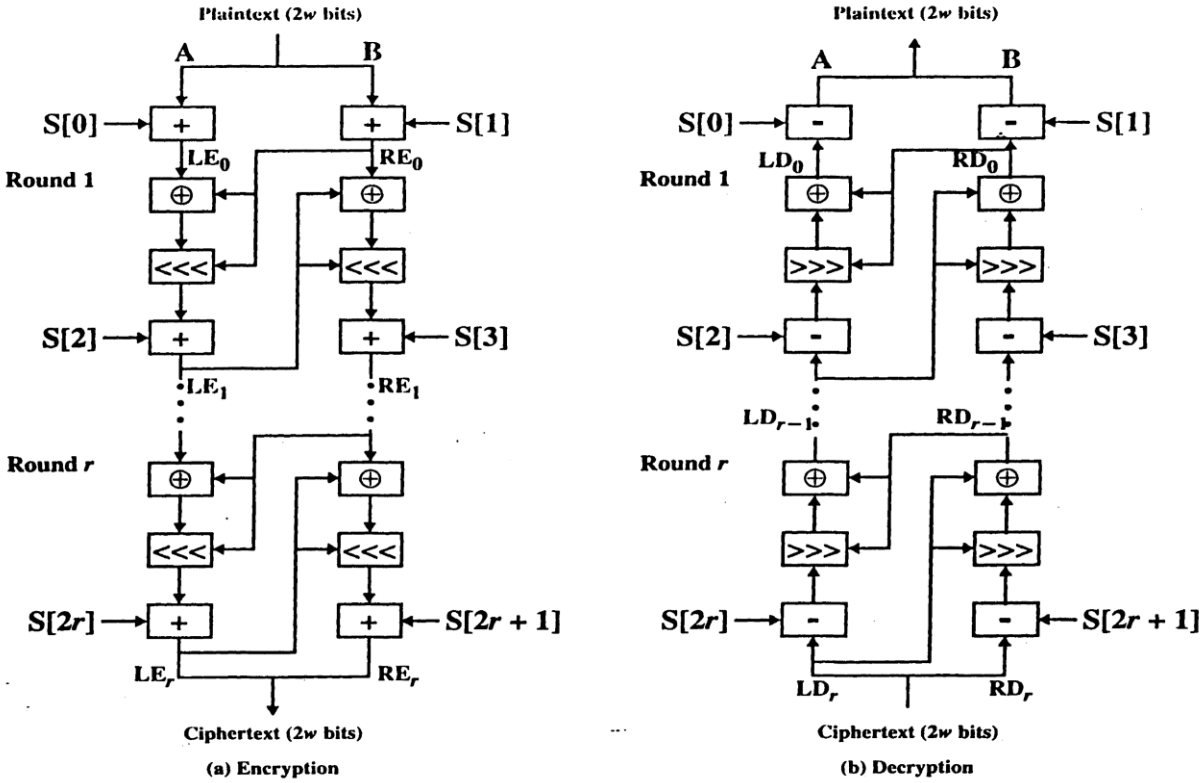


Figure 4.12 RC5 Encryption and Decryption.

Decryption

- Decryption, shown in Figure 4-12b, is easily derived from the encryption algorithm. In this case, the 2*w* bits of ciphertext are initially assigned to the two one-word variables LD_r, and RD_r.
- We use the variables LD_i and RD_i to refer to the left and right half of the data before round *i* has begun, where the rounds are numbered from *r* down to 1.

RC5 Modes:

To enhance the effectiveness of RC5 in interoperable implementations, RFC 2040 defines four different modes of operation:

- **RC5 block cipher:** This is the raw encryption algorithm that takes a fixed—size input block (2*w* bits) and produces a ciphertext block of the same length using a transformation that depends on a key.
- **RCS-CBC:** This is the cipher block chaining mode for RC5- CBC. CBC processes messages whose length is a multiple of the RC5 block size (multiples of 2w bits. CBC provides enhanced security compared to ECB because repeated blocks of plaintext produce different blocks of ciphertext.

- **RCS-CBC-Pad:** This is a CBC style of algorithm that handles plaintext of any length- The ciphertext will be longer than the plaintext by at most the size of a single RC5 block.
- **RCS-CTS:** This is the ciphertext stealing mode, which is also a CBC style of algorithm- This mode handles plaintext of any length and produces ciphertext of equal length.

The encryption sequence is as follows:

1. Encrypt the first $(N - 2)$ blocks using the traditional CBC technique.
2. Exclusive-OR P_{N-1} with the previous ciphertext block C_{N-2} to create Y_{N-1}.
3. Encrypt Y_{N-1} to create E_{N-1}.
4. Select the first L bytes of E_{N-1} to create C_N.
5. Pad P_N with zeros at the end and exclusive-OR with E_{N-1} to create Y_N.
6. Encrypt Y_N to create C_{N-1}.

2.15. KEY DISTRIBUTION