Symmetric Ciphers

Unit -2 [10 Hours]

Confusion and Diffusion

- "Confusion" and "diffusion" are two fundamental concepts in the design of secure cryptographic algorithms, particularly in the context of symmetric encryption.
- These concepts were introduced by ***Claude Shannon***, a pioneer in the field of cryptography, and are crucial in achieving the properties of confusion and diffusion in encryption algorithms.

Confusion

- Confusion refers to *the concept of making the relationship between the plaintext and the ciphertext as complex and unpredictable as possible*.
- In other words, confusion ensures that a change in a single bit of the plaintext should lead to changes in multiple bits of the ciphertext.
- This property makes it difficult for an attacker to deduce any meaningful information about the plaintext even if they have access to a large amount of ciphertext.
- One way to achieve confusion is by using complex mathematical operations, such as **substitution**, in the encryption process.
- Substitution involves replacing plaintext elements (e.g., bits or bytes) with corresponding elements from a predefined table or function.

Diffusion

- Diffusion is an encryption operation where the influence of one plaintext symbol is spread over many ciphertext symbols with the goal of hiding statistical properties of the plaintext.
- Diffusion is typically achieved through permutation, transposition, and mixing operations.

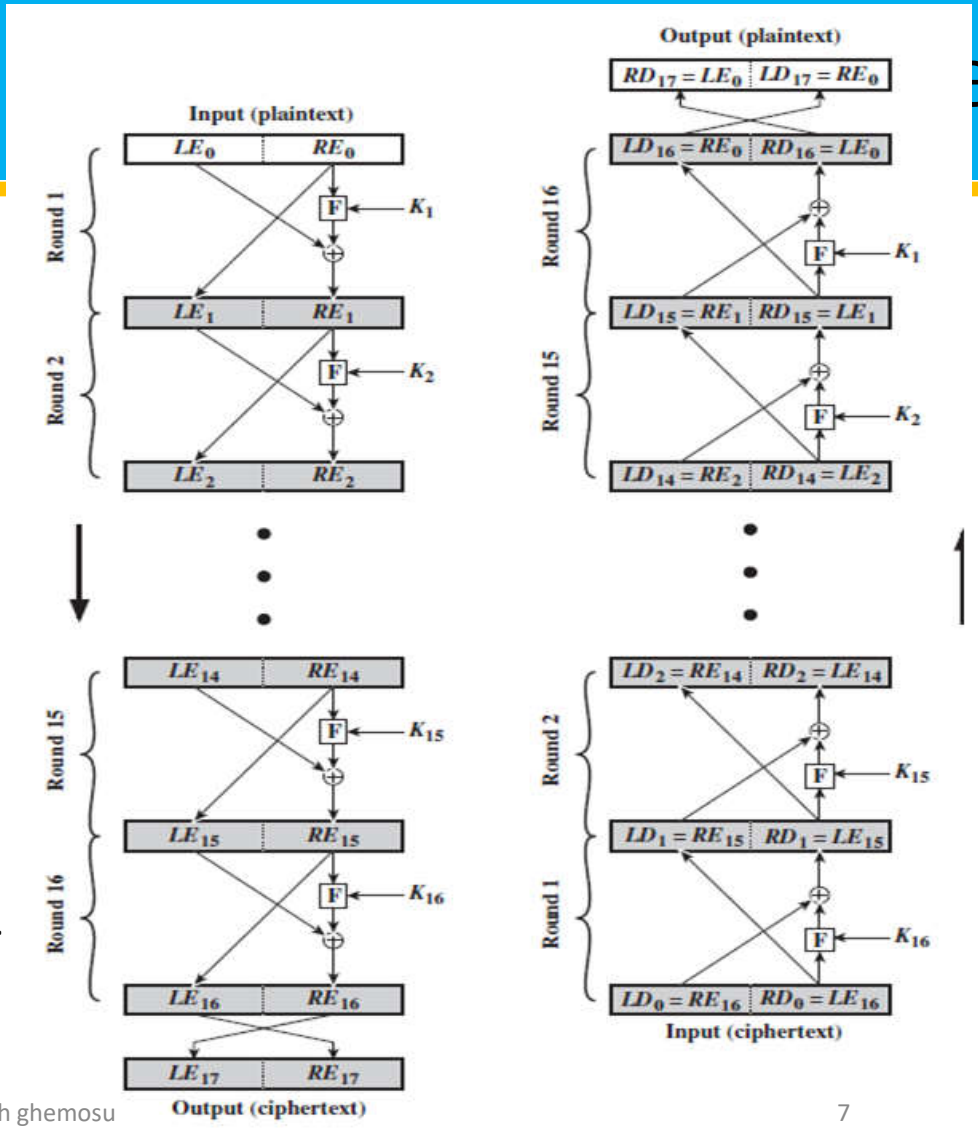
Purpose: Confusion and Diffusion

- In modern symmetric encryption algorithms, confusion and diffusion are combined to create strong encryption schemes that resist various types of attacks, including brute force attacks, statistical attacks, and differential attacks.
- These properties contribute to the overall security of the encryption process by making the relationship between the plaintext and ciphertext highly complex and difficult to predict..

Use case: Confusion and Diffusion

- An example of a symmetric encryption algorithm that incorporates both confusion and diffusion is the **Advanced Encryption Standard (AES)**.
- AES achieves these properties through a series of rounds that involve substitution (confusion) and permutation (diffusion) operations.
- The careful design of these operations and the multiple rounds contribute to AES's robust security.

- The Feistel cipher structure is a **fundamental design concept used in symmetric cryptographic algorithms for encryption and decryption.**
- **A symmetric structure used in the construction of block ciphers.**
- Feistel Cipher is not a specific scheme of block cipher. It is a design model from which many different block ciphers are derived.
- DES is just one example of a Feistel Cipher.
- A cryptographic system based on Feistel cipher structure uses the same algorithm for both encryption and decryption.
- It was introduced by Horst Feistel in the early 1970s **and has been used** as the basis for various encryption algorithms, including the **Data Encryption Standard (DES)** and its variants.



Fiestel Cipher Structure

- The Feistel cipher structure operates by dividing the plaintext into two equal-sized blocks (usually called the left half and the right half) and then processing these blocks through a series of rounds.
- Each round involves a combination of permutation, substitution, and mixing operations.
- The output of one round is used as input for the next round, and after multiple rounds, the final output is the encrypted ciphertext.
- Here's an overview of how the Feistel cipher structure works:
  1. **Key Schedule**
  2. **Initial Permutation**
  3. **Rounds**
  4. **Final Permutation**

1. Key Schedule

- Before the encryption process begins, a key schedule generates a set of round keys from the main encryption key.

2. Initial Permutation

- The plaintext is divided into two halves, L_0 and R_0.
- An initial permutation may be applied to the halves, shuffling the bits in a specific manner.

3. Rounds

- The Feistel structure consists of multiple rounds (typically 16 rounds in DES).
- In each round:
  - The right half (R_{i-1}) is subjected to a combination of substitution and permutation operations (F function) using the round key.
  - The output of the F function is XORed with the left half (L_{i-1}).
  - The left half (L_{i-1}) becomes the new right half (R_i), and the XOR result becomes the new left half (L_i).

4. Final Permutation

- After all rounds, the final left and right halves are swapped.
- A final permutation, which is the inverse of the initial permutation, is applied to the concatenated halves to produce the ciphertext.

Encryption in Fiestel Cipher Structure

- 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 an input $\mathbf{L_{i-1}}$ and $\mathbf{R_{i-1}}$ derived form the previous round, as well as a subkey $\mathbf{K_i}$ derived from the overall k.

Encryption in Fiestel Cipher Structure

- A substitution is performed on the left of the data.
  - Apply **a round Function F** to the right half of the data and then take the X-OR of the output of that function with the left half of the data.
  - F has the same general structure for each round but is parameterized by the round subkey $\mathbf{K_i}$.
- Following this substitution, **a permutation is performed** that consists of the interchange of the two halves of the data.

Decryption in Fiestel Cipher Structure

- Decryption has **same process as the encryption**; only the input keys parameterized it.
- the **subkeys are used in reverse order**, i.e. K_{16} is used in the first round of decryption.
- The **first round of decryption reverse the last round of encryption**, the second round of decryption reverse the second round of encryption, and so on.

Advantages: Fiestel cipher Structure

The Feistel cipher structure offers several advantages:

- It provides a **straightforward way to achieve both confusion and diffusion properties in encryption**.
- The same **round function** and **key schedule** can be used for both encryption and decryption, which simplifies implementation.
- The structure is amenable to parallel processing in hardware implementations.

Fiestel Cipher Structure

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block Size**: larger block size means greater security but reduced encryption/decryption speed.
- **Key Size**: larger block size means greater security but reduced encryption/decryption speed. Common size: 64 (not adequate), 128
- **Number of rounds**: 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**: greater complexity generally means greater resistance to cryptanalysis.

Fiestel Cipher Structure

- In general, for the i^{th} iteration of the encryption algorithm:

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

Rearranging terms:

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

Substitution Permutation Network

- Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper
- It forms basis of modern block ciphers such as **AES**.
- S-P nets are **based on the two primitive cryptographic operations** seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
- Provide *confusion* & *diffusion* of message & key.

Substitution Box (S-Box)

- S-boxes are **nonlinear transformations that replace a fixed number of input bits with a fixed number of output bits**.
- In particular, the length of the output should be the same as the length of the input
- They **introduce confusion by making the relationship between the input and output bits complex and non-linear**, which adds security against various attacks like linear and differential cryptanalysis.
- S-boxes are usually designed to have specific properties, such as **resistance to different types of attacks** and **avalanche effect** (small changes in input should lead to significant changes in output).

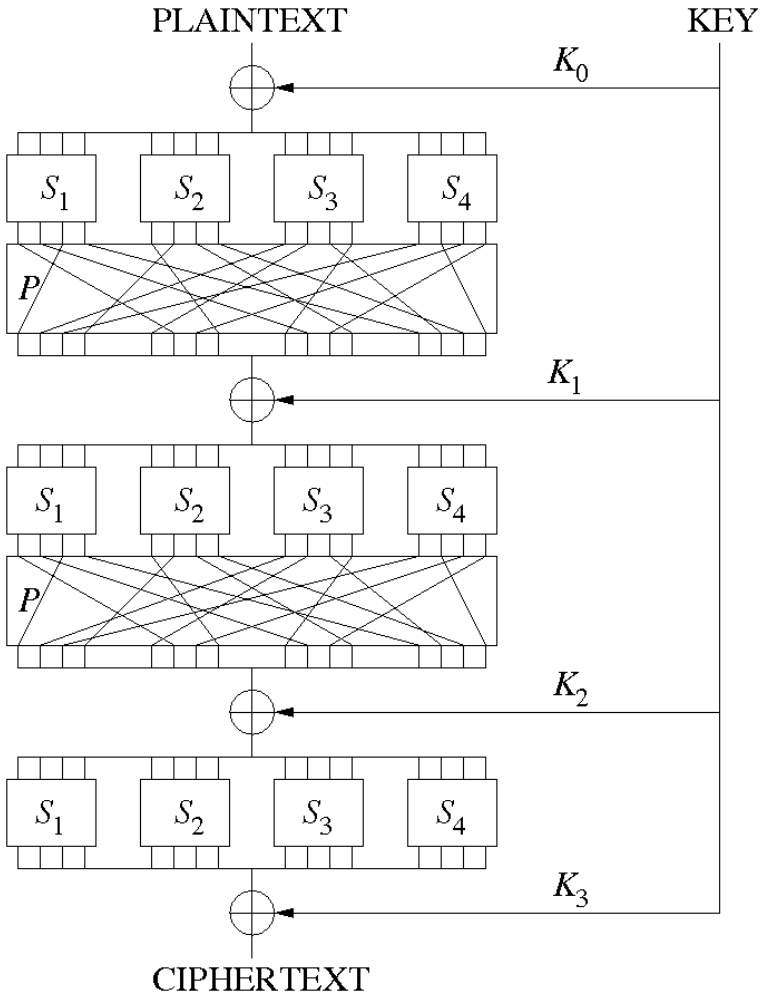


Figure: A sketch of a substitution–permutation network with 3 rounds, encrypting a plaintext block of 16 bits into a ciphertext block of 16 bits. The S-boxes are the S_i, the P-boxes are the same P, and the round keys are the K_i.

Permutation Box (P-box)

- Permutation layers, also known as P-boxes, **shuffle the bits of the data to provide diffusion**.
- It takes the outputs of all the S-boxes of one round, permutes the bits, and feeds them into the S-boxes of the next round.
- A good P-box has the property that the output bits of any S-box are distributed to as many S-box inputs as possible.
- At each round, the **round key** (obtained from the key with some simple operations, for instance, using S-boxes and P-boxes) is combined using some group operation, typically XOR.

Data Encryption Standard (DES)

- The Data Encryption Standard (DES) is **a symmetric-key block cipher** that was widely used for secure data encryption and decryption.
- It was one of the **most well-known and commonly used encryption algorithms** for several decades, until the introduction of AES in 2001.
- **First adopted in 1977** by National Bureau of Standard (NBS), now now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46).
- However, due to advances in computing power and cryptographic analysis, *DES eventually became insecure and was replaced by more secure algorithms*.

DES

- Data are encrypted in **64-bit blocks** using a **56-bit key**.
- The same steps, with the same key, are used to reverse the encryption.
- DES has the **same structure as Feistel cipher** *except* the F function and additional initial and final permutations, IP and IP^{-1}.

DES Encryption Overview

Encryption Phases

1. Initial Permutation (IP)
2. Rounds
3. Final Permutation (IP^{-1})

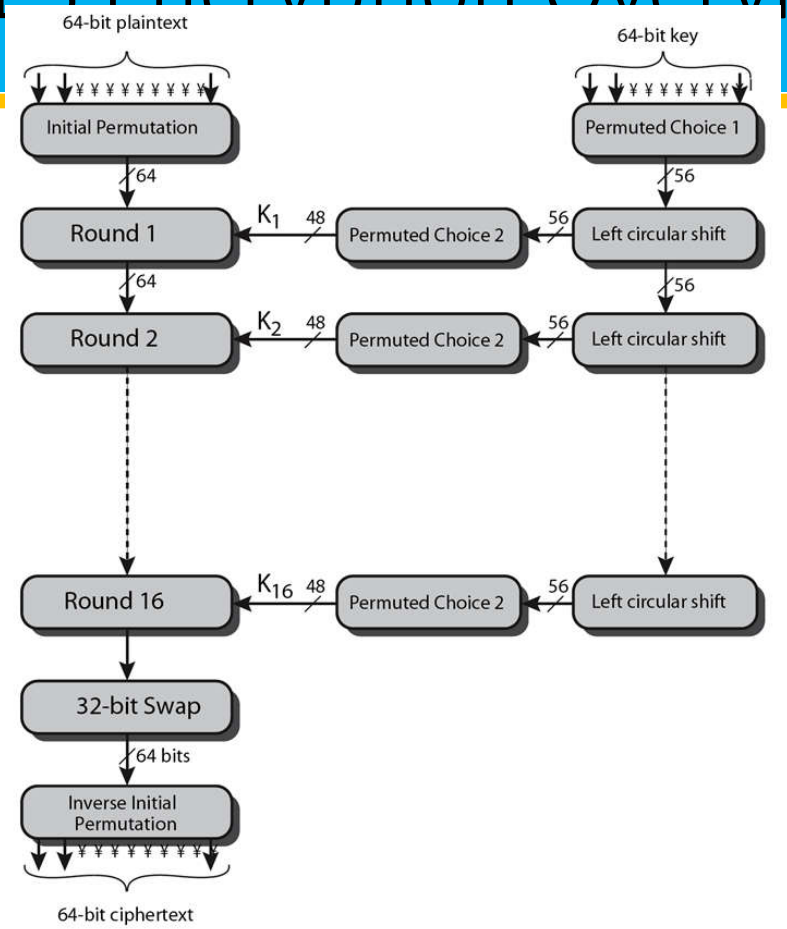


Figure: General Description of DES Encryption Algorithm

1. Initial Permutation

- The **64-bit plaintext block is permuted** according to a fixed permutation table called the **Initial Permutation (IP) table**.
- This **rearranges the bits in a predefined manner**.
- *The Initial Permutation (IP) table in the Data Encryption Standard (DES) is a fixed permutation table that reorders the bits of the input data block before starting the encryption or decryption process*.

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

Figure: IP Table

- IP table consist of 64 bits each numbered from 1 to 64.
- Each entry in the permutation table indicates the position of a numbered bit in the output.

Example: Initial Permutation

- Let's consider the following 64-bit binary input data block:

0110111001101101011100100111001101110011011100100110111101101111

- IP table

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

Step-by-Step Explanation:

1. The leftmost bit of the input data (bit position 1) is moved to the 58th bit position in the output.
2. The second bit of the input data (bit position 2) is moved to the 50th bit position in the output.
3. The third bit of the input data (bit position 3) is moved to the 42nd bit position in the output.
4. This process continues for each bit of the input data, following the mapping specified in the IP table.
5. Let's illustrate this with a few steps:

Input: 01101110 01101101 01110010 01110011 01110011 11001110 11011011 01101111
Output: 10101001 10110110 01010111 10010011 11110110 11110011 00111101 10000110

2. Rounds

DES performs 16 rounds of operations. In each round, the following steps are performed:

- **Expansion (E)**: The 32-bit right half of the data is expanded to 48 bits using the expansion permutation table.
- **Key Mixing (XOR)**: The expanded data is bitwise XORed with a round key derived from the main encryption key.
- **Substitution (S-Boxes)**: The XORed data is divided into 8 blocks of 6 bits each. Each block is substituted using a set of 8 S-boxes (substitution boxes), resulting in 8 blocks of 4 bits each.
- **Permutation (P)**: The outputs of the S-boxes are permuted using a fixed **permutation table** called the P-box, providing further diffusion.
- **XOR and Swap**: The output of the P-box is XORed with the left half of the original data and then swapped with the right half to produce **preoutput**.. The left half becomes the new right half for the next round.

2. Rounds

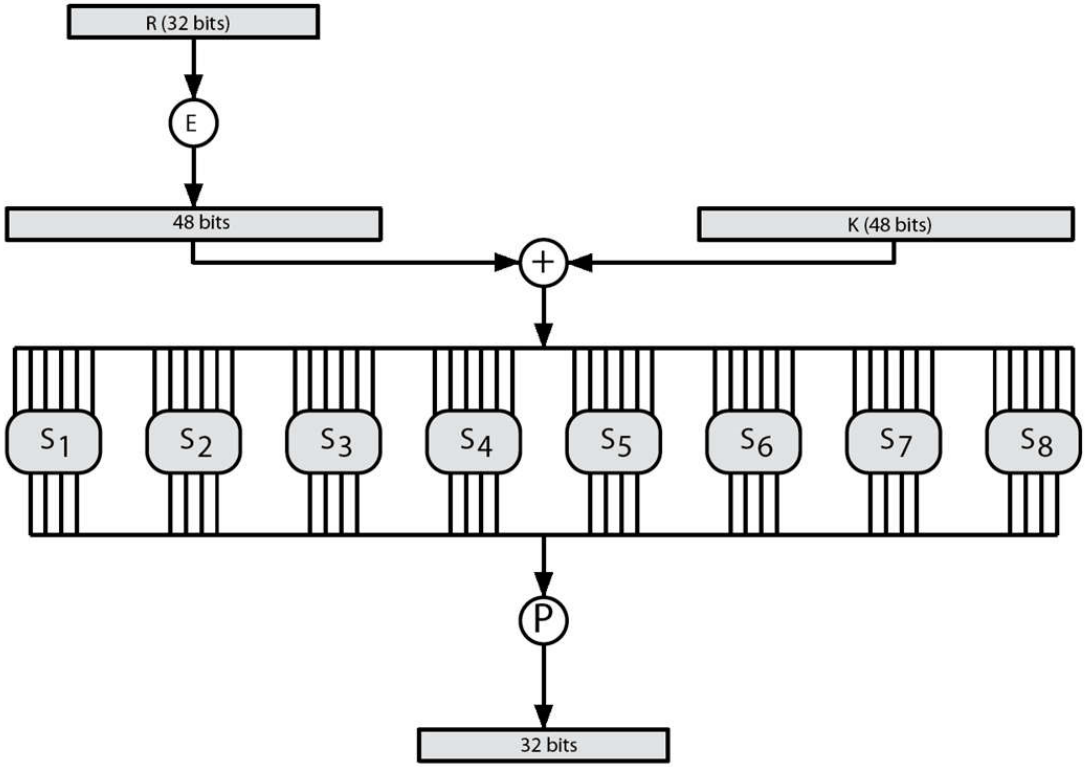


Figure: The internal structure of the DES round function.

Example: DES Expansion Permutation (E)

- The **DES expansion permutation** table consists of 48 entries that determine the positions of the bits in the expanded block. Here's the layout of the E table:

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

- Each entry in the E table indicates the position of the corresponding bit in the input in the rearranged output. For example, the first entry in the E table (32) means that the 32nd bit of the input will become the first bit in the output, the 1st bit of the input data becomes the 2nd bit of the expanded output, and so on.

Example: Let's demonstrate the application of the **E table** with a sample 32-bit input data half:

Input Data: **11001100110110101001110011011011** (32 bits)

Mapping the bits according to the E table:

Expanded Data:

?????

- The final expanded output is a 48-bit block obtained by applying the expansion permutation. This expanded data will then be used for the subsequent XOR operation with the round key in the DES algorithm.

Example: DES Rounds

- The **DES P table** consists of 32 entries that determine the permutation of the bits. Here's the layout of the P table:
- Each entry in the P table indicates the position of the corresponding bit in the input in the rearranged output. For example, the first entry in the P table (16) means that the 16th bit of the input will become the first bit in the output, and so on.

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

Figure: Permutation Table in DES

Example: Let's demonstrate the application of the P table with a sample input after S-box substitution in a round:

Input Data: 11001100110110101001110011011011 (32 bits)

After S-box substitution, let's say the output of the S-boxes is:

S-box Output: 01101111010100001000110101101011 (32 bits)

Mapping the bits according to the P table:

Output Data: 10101100011010111101011010001110 (32 bits)

DES: S-box Substitution

- DES uses a total of 8 S-boxes, each with a unique substitution table.
- maps 6 to 4 bits.
- Each 6-bit input to an S-box is split into two parts: the first and last bit determine the row, and the middle four bits determine the column within the S-box table.

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Given a 6-bit input, the 4-bit output is found by selecting the row using the outer two bits (the first and last bits), and the column using the inner four bits. For example, an input "**0**11011**1**" has outer bits "**01**" and inner bits "1101"; the corresponding output would be "1001"

3. Final Permutation (IP^{-1})

- Finally, the **preoutput** is passed through a permutation [IP^{-1}] that is the **inverse of the initial permutation function**, to produce the 64-bit ciphertext .
- The purpose of the final permutation is to provide a final diffusion of the bits and **produce the final encrypted output**.
- The Final Permutation table consists of 64 entries that determine indicates the position of the corresponding bit in the output data block**.** Here's the layout of the IP^{-1} table:

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**

Figure: Inverse Initial Permutation (IP^{-1})

DES: Key Scheduling

- Initially, the 64-bit key is passed through a permutation function.
- Every 8th bit (8, 16, …. 56, 64) is ignored.
- Then, for each of the 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.

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

Permutated choice 1 (PC-1)

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

Permutated choice 2 (PC-2)

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.

Triple DES

- Also known as TDES/3DES
- Uses DES three times for encryption and decryption with different keys
- 3DES used 3 or 2 keys
- Encrption

  $C = E(K_3, D(K_2, E(K_1, P)))$
- Decryption

  $P = D(K_1, E(K_2, D(K_3, C)))$

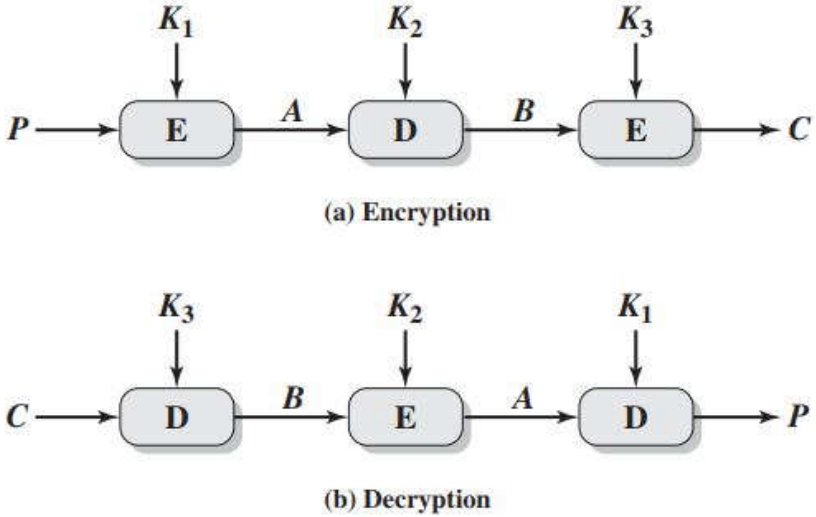


Figure: Triple DES

Finite Fields

- A number of cryptographic algorithms rely heavily on properties of finite fields, notably:
  - Advanced Encryption Standard (AES)
  - Elliptic curve cryptography
  - Message Authentication code (MAC)

Basic Concepts on Groups, Rings and Fields

- Groups, Rings and Fields are the fundamental elements of a branch of mathematics knowns as abstract algebra, or modern algebra.

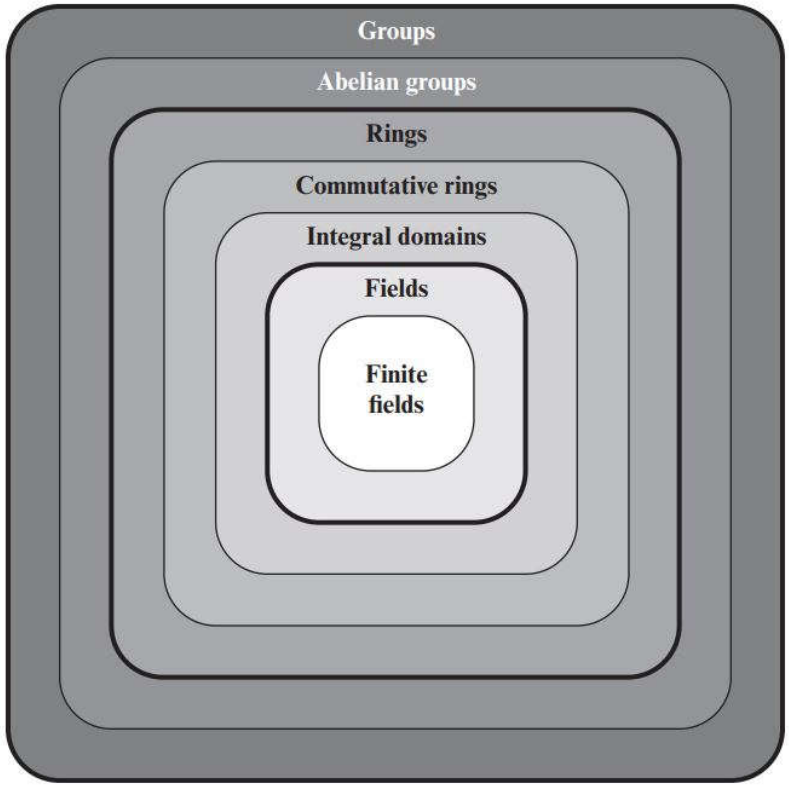


Figure: Groups, Rings and Fields

Groups

A **group** *G*, sometimes denoted by {*G*, **.**}, is a set of elements with a binary operation denoted by **.** that associates to each ordered pair (*a, b*) of elements in *G* an element (*a* **.** *b*) in *G*, such that the following axioms are obeyed . The operation **.** generic and can refer to addition, multiplication, or some other mathematical operation.

Axioms:

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

(A2) Associative: a . (b . c) = (a . b) . c for all a, b, c in G.

(A3) Identity element: There is an element e in G such that a . e = e . 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 . a′ = a′ . a = e.

Finite and Infinite Group, Abelian and Cyclic Group

- 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.
- A group G is **cyclic** if every element of G is a power a^k (k is an integer) of a fixed element $a \varepsilon G$. 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

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^{-1}.

  (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.

Rings

- 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.

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

- 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:

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

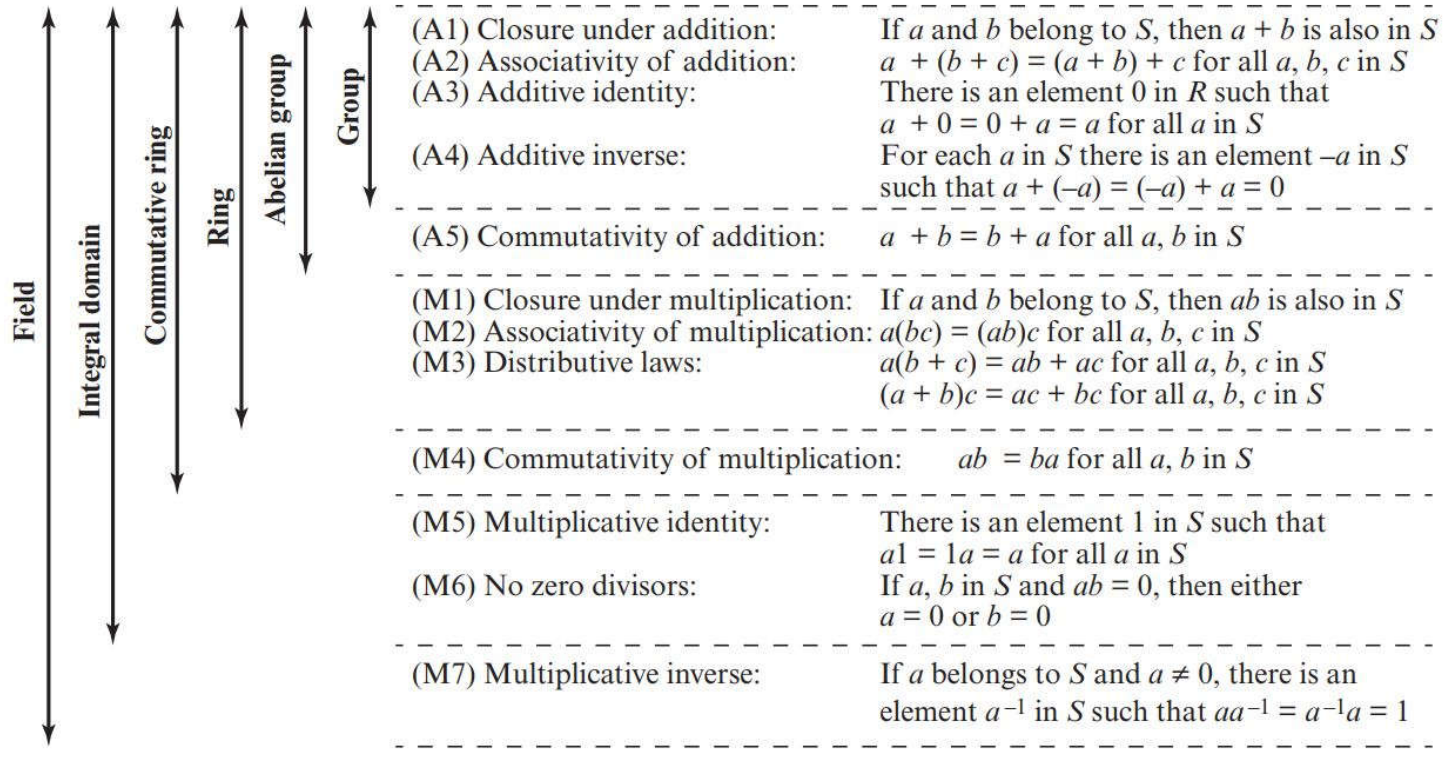


Figure: Properties of Groups, Rings and Fields

Divisibility and Division Algorithm

- We say that a non-zero number b **divides** a if for some m have $a=mb$ (a, b, m all integers).
- That is b divides into a with no remainder
- denote this $b \mid a$
- and say that b is a **divisor** of a
- eg. all of 1,2,3,4,6,8,12,24 divide 24

Modular Arithmetic

- define **modulo operator** "`a mod n`" to be remainder when a is divided by n
- Two integers a and b are said to be **congruent modulo** if (a mod n) = (b mod n) and written as: `a = b mod n`
  - ie. use the term **congruence** when divided by *n,* a & b have same remainder
  - eg. 100 = 34 mod 11
- b is called a **residue** of a mod n
  - since with integers can always write: `a = qn + b`
  - usually chose smallest positive remainder as residue
    - ie. `0 <= b <= n-1`

Modular Arithmetic Operations

- is 'clock arithmetic'
- uses a finite number of values, and loops back from either end
- modular arithmetic is when do addition & multiplication and modulo reduce answer
- 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. $[(a \bmod n) * (b \bmod n)] \bmod n = (a * b) \bmod n$

Division Algorithm

- 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 remainder r that obey the following relationship:

$$a = qn + r \qquad 0 \leq r < n;\ q = [a|n]$$

- The remainder r is often referred to as a **residue**.

The Euclidean Algorithm

- For any integers a, b with a ≥b ≥0,

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

- Example:

  gcd(55, 22) = gcd(22, 55 mod 22) = gcd(22, 11) = 11

- we can define the Euclidean algorithm concisely as the following recursive function.

```
Euclid(a, b)
        if (b = 0) then return a;
        else return Eculid(b, a mod b)
```

Greatest Common Divisor

- A common problem in number theory
- GCD (a,b) of a and b is the largest positive number that divides both a and b
  - eg GCD(60,24) = 12
- An equivalent definition is the following:
  gcd(a, b) = max[k, such that $k|a$ 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|).

  e.g. gcd(60, 24) = gcd(60, -24) = 12

- often want **no common factors** (except 1) and hence numbers are **relatively prime**
  - eg GCD(8,15) = 1
  - hence 8 & 15 are relatively prime

Euclidean Algorithm

- an efficient way to find the GCD(a, b)
- uses theorem that:
  - `GCD(a,b) = GCD(b, a mod b)`
- Euclidean Algorithm to compute GCD(a,b) is:

```
EUCLID(a,b)
1. A = a; B = b
2. if B = 0 return  A = gcd(a , b)
3. R = A mod B
4. A = B
5. B = R
6. goto 2
7. stop
```

Example GCD(1970,1066)

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

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

Dividend	Divisor	Quotient	Remainder
$a = 1160718174$	$b = 316258250$	$q_1 = 3$	$r_1 = 211943424$
$b = 316258250$	$r_1 = 211943434$	$q_2 = 1$	$r_2 = 104314826$
$r_1 = 211943424$	$r_2 = 104314826$	$q_3 = 2$	$r_3 = 3313772$
$r_2 = 104314826$	$r_3 = 3313772$	$q_4 = 31$	$r_4 = 1587894$
$r_3 = 3313772$	$r_4 = 1587894$	$q_5 = 2$	$r_5 = 137984$
$r_4 = 1587894$	$r_5 = 137984$	$q_6 = 11$	$r_6 = 70070$
$r_5 = 137984$	$r_6 = 70070$	$q_7 = 1$	$r_7 = 67914$
$r_6 = 70070$	$r_7 = 67914$	$q_8 = 1$	$r_8 = 2156$
$r_7 = 67914$	$r_8 = 2156$	$q_9 = 31$	$r_9 = 1078$
$r_8 = 2156$	$r_9 = 1078$	$q_{10} = 2$	$r_{10} = 0$

Extended Euclidean Algorithm

- Extended Euclidean algorithm the following equation.

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

where x and y are two integers of opposite signs.

- Example: For some of the values of x and y where a = 42 and b = 30

y \ x	−3	−2	−1	0	1	2	3
−3	−216	−174	−132	−90	−48	−6	36
−2	−186	−144	−102	−60	−18	24	66
−1	−156	−114	−72	−30	12	54	96
0	−126	−84	−42	0	42	84	126
1	−96	−54	−12	30	72	114	156
2	−66	−24	18	60	102	144	186
3	−36	6	48	90	132	174	216

Observe that all of the entries are divisible by 6. This is not surprising, because both 42 and 30 are divisible by 6, so every number of the form 42x + 30y = 6(7x + 5y) is a multiple of 6. Note also that **gcd(42, 30) = 6** appears in the table. In general, it can be shown that for given integers a and b, the smallest positive value of ax + by is equal to gcd(a, b).

Residue Classes

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

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

- This is referred to as the set of **residues**, or **residue classes (mod n)**.
- To be more precise, each integer in $\mathbb{Z}_n$ represents a **residue class**. We can label the residue classes (mod n) as [0], [1], [2], . . ., [n – 1], where,

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

The residue classes (mod 4) are

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

- Start with the number: -13
- Divide -13 by 4: -13 / 4 = -3 with a remainder of -1
- Now, add the divisor (4) to the remainder (-1): -1 + 4 = 3

Mathematically, this can be represented as: -13 mod 4 = ((-13) % 4 + 4) % 4 = (-1 + 4) % 4 = 3 % 4 = 3

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 $w + z \equiv 0 \bmod n$

Figure: Properties of Modular Arithmetic for Integers in Z_n

Additive inverse and multiplicative inverse

if (a + b) ≡ (a + c) (mod n) then b ≡ c (mod n)

e.g (5 + 23) ≡ (5 + 7) (mod 8) then 23 ≡ 7 (mod 8)

Additive Inverse:

((-a) + a + b) ≡ ((-a) + a + c) (mod n)
b ≡c (mod n)

However, the following statement is true only with the attached condition:
if (a x b) ≡ (a x c)(mod n) then b ≡ c (mod n) if a is *relatively prime* to n

Two integers are relatively prime if their only common positive integer factor is 1

Multiplicative Inverse:

$((a^{-1})ab) \equiv ((a^{-1})ac) \pmod{n}$
$b \equiv c \pmod{n}$

Finite Fields of the Form GF(p): Galois Fields

- Infinite fields are not of particular interest in the context of cryptography.
- *Finite fields play a crucial role in cryptographic algorithms*.
- A finite field (number of elements in the field) must be a power of a prime p^n, where n is a positive integer.
- The finite field of order p^n is generally denoted $GF(p^n)$; GF stands for **Galois field**, in honor of the mathematician who first studied finite fields.
- A field with order m only exists if m is a prime power, i.e., $m = p^n$, for some positive integer n and prime integer p. p is called the characteristic of the finite field.
- Two special cases are of interest for our purposes.
  - For $n = 1$, we have the finite field GF(p);.
  - For finite fields of the form GF(p^n) where n > 1, GF(2^n) fields are of particular cryptographic interest

Types of Fields

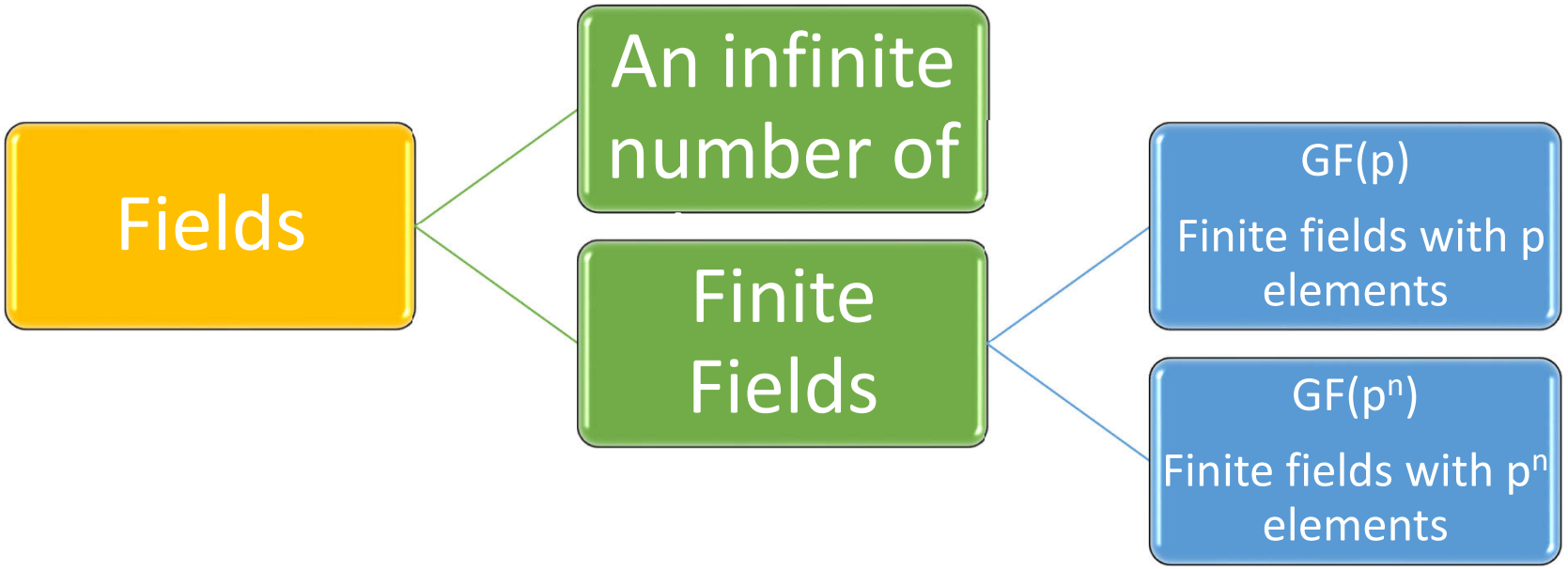


Figure: Types of fields

Finite field of order p: Galois Field GF(p)

- For a given prime, p, we define the finite field of order p, $GF(p)$, as the set Z_p of integers $\{0, 1, \ldots, p - 1\}$ together with the arithmetic operations **modulo *p***.
- have a multiplicative inverses **if and only if** that integer is *relatively prime to n.*
- can do addition, subtraction, multiplication, and division without leaving the field GF(p)

Multiplicative inverse (w^{-1})	**For each w ∈ Z p, w ≠ 0, there exists a z ∈ Zp such that w x z ≡ 1 (mod p)**

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

Multiplicative Inverse

- Because w is relatively prime to p, if we multiply all the elements of Z_p by w, the resulting residues are all of the elements of Z_p permuted.
- Thus, exactly one of the residues has the value 1. Therefore, there is some integer in Z_p that, when multiplied by w, yields the residue 1.
- That integer is the multiplicative inverse of w, designated w^{-1}. Therefore, Z_p is in fact a finite field.
- Equation is written as:

if (a x b) ≡ (a x c)(mod p) **then** b ≡ c(mod p)

Multiplying both sides of Equation by the multiplicative inverse of a, we have

$((a^{-1}) \times a \times b) \equiv ((a^{-1}) \times a \times c) \pmod{p}$

$b \equiv c \pmod{p}$

Finite field of order p: Galois Field GF(p)

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

+	0	1
0	0	1
1	1	0

Addition

×	0	1
0	0	0
1	0	1

Multiplication

w	$-w$	w^{-1}
0	0	–
1	1	1

Inverses

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

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

(a) Addition modulo 8

+	0	1	2	3	4	5	6
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

(d) Addition modulo 7

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

(b) Multiplication modulo 8

×	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

(e) Multiplication modulo 7

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

(c) Additive and multiplicative inverses modulo 8

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

(f) Additive and multiplicative inverses modulo 7

Figure: Arithmetic Modulo 8 and Modulo 7 examples

Finding the Multiplicative Inverse in GF(p)

- It is easy to find the multiplicative inverse of an element in $GF(p)$ for small values of p. You simply construct a multiplication table, such as shown in Table in previous slide, and the desired result can be read directly.
- However, for large values of p, this approach is not practical.
- 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 such that $bb^{-1} = 1 \mod a$.
- 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.

Finding the Multiplicative Inverse in GF(p)

- 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 have,

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

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

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

Polynomial Arithmetic

- Polynomial arithmetic is a branch of algebra dealing with some properties of polynomials which share strong analogies with properties of number theory relative to integers.
- Three classes of polynomial arithmetic:
  1. Ordinary polynomial arithmetic, using the basic rules of algebra.
  2. Polynomial arithmetic in which the arithmetic on the coefficients is performed modulo p; that is, the coefficients are in $\mathrm{GF}(p)$.
  3. Polynomial arithmetic in which the coefficients are in $\mathrm{GF}(p)$, and the polynomials are defined modulo a polynomial $m(x)$ whose highest power is some integer n.

Ordinary Polynomial Arithmetic

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

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

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

- A zero-degree polynomial is called a **constant polynomial** and is simply an element of the set of coefficients. An n^{th}-degree polynomial is said to be a **monic polynomial** if $a_n = 1$.
- In the context of abstract algebra, we are usually not interested in evaluating a polynomial for a particular value of *x* [e.g., $f(7)$]. To emphasize this point, the variable *x* is sometimes referred to as the **indeterminate** .

Treatment of Polynomials

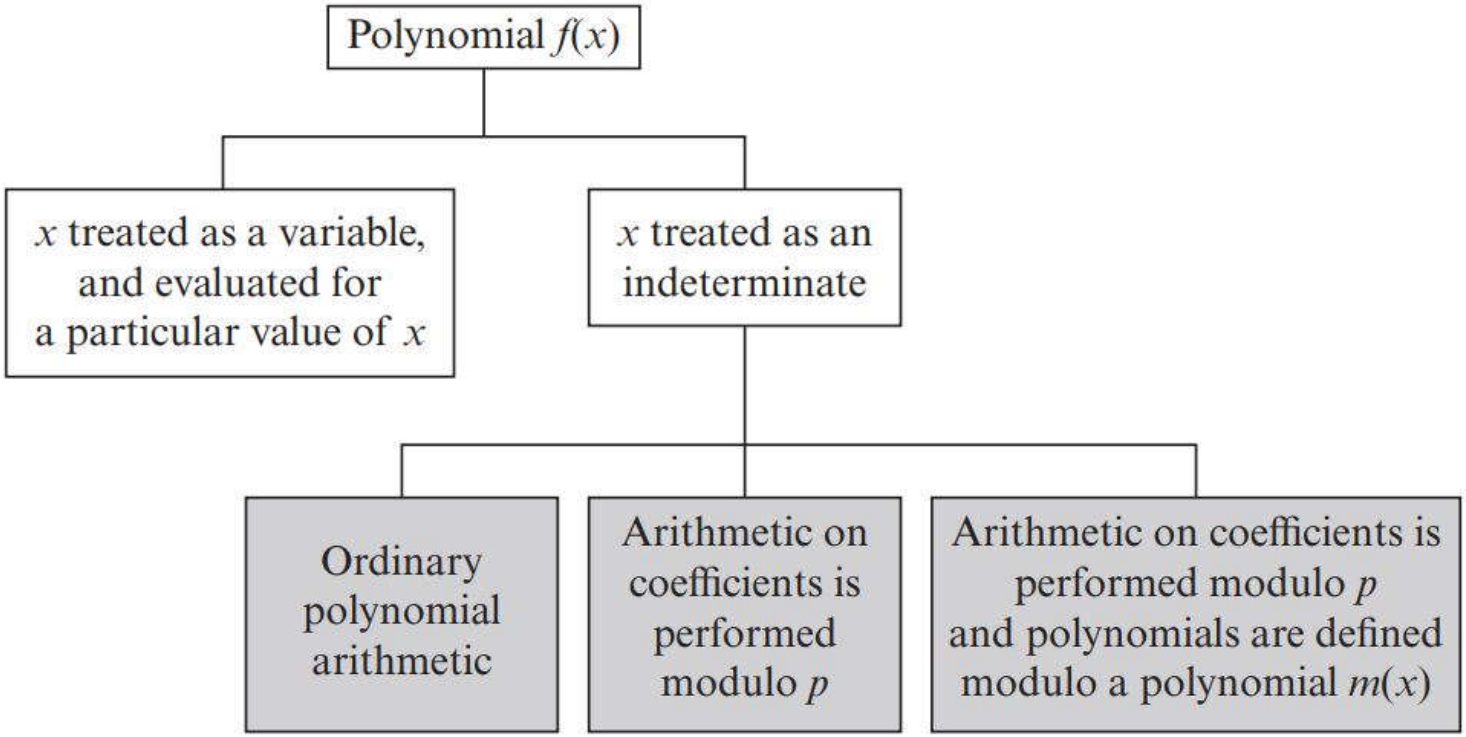


Figure: Treatment of Polynomials

Ordinary Polynomial Arithmetic

- add or subtract corresponding coefficients
- multiply all terms by each other
- eg

  let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$

  $f(x) + g(x) = x^3 + 2x^2 - x + 3$

  $f(x) - g(x) = x^3 + x + 1$

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

Polynomial Arithmetic with Modulo Coefficients

- when computing value of each coefficient do calculation modulo some value
  - forms a polynomial ring
- could be modulo any prime
- but we are most interested in mod 2
  - ie all coefficients are 0 or 1
  - eg. let $f(x) = x^3 + x^2$ and $g(x) = x^2 + x + 1$

$$f(x) + g(x) = x^3 + x + 1$$

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

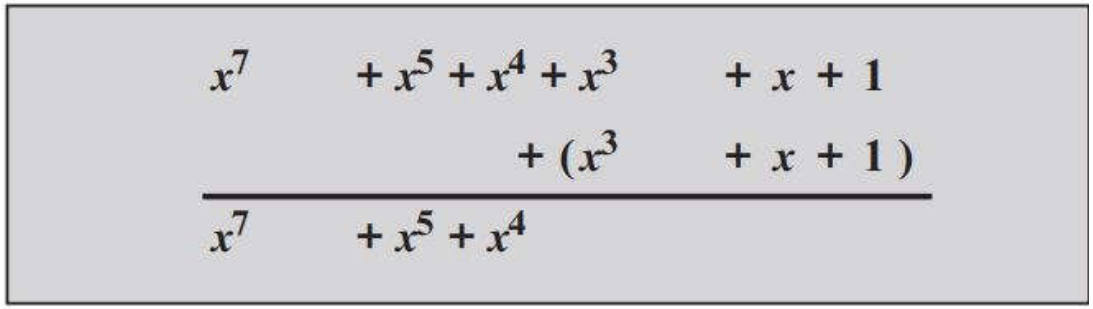
Polynomial over GF(2)

- addition is equivalent to XOR operation
- multiplication is equivalent to the logical AND operation

$$1 + 1 = 1 - 1 = 0$$

$$1 + 0 = 1 - 0 = 1$$

$$0 + 1 = 0 - 1 = 1$$



$$\begin{array}{rl} & x^7 + x^5 + x^4 + x^3 + x + 1 \\ & \quad + (x^3 + x + 1) \\ \hline & x^7 + x^5 + x^4 \end{array}$$

(a) Addition

$$\begin{array}{rl} & x^7 + x^5 + x^4 + x^3 + x + 1 \\ & \quad - (x^3 + x + 1) \\ \hline & x^7 + x^5 + x^4 \end{array}$$

(b) Subtraction

Figure: Examples of Polynomial Arithmetic over GF(2)

$$
\begin{array}{rrrrrrrrrrr}
 & & & x^7 & + x^5 & + x^4 & + x^3 & & + x & + 1 \\
 & & & & & & \times (x^3 & & + x & + 1) \\
\hline
 & & & x^7 & + x^5 & + x^4 & + x^3 & & + x & + 1 \\
 & & x^8 & & + x^6 & + x^5 & + x^4 & + x^2 & + x & \\
x^{10} & & + x^8 & + x^7 & + x^6 & & + x^4 & + x^3 & & \\
\hline
x^{10} & & & & & & + x^4 & + x^2 & & + 1
\end{array}
$$

(c) Multiplication

$$
\begin{array}{r|l}
 & x^4 + 1 \\
\cline{2-2}
x^3 + x + 1 & x^7 + x^5 + x^4 + x^3 + x + 1 \\
 & x^7 + x^5 + x^4 \\
\cline{2-2}
 & x^3 + x + 1 \\
 & x^3 + x + 1 \\
\cline{2-2}
 &
\end{array}
$$

(d) Division

Figure: Examples of Polynomial Arithmetic over GF(2)

Polynomial Division

- Given polynomials $f(x)$ of degree n and $g(x)$ of degree (m), ($n \geq m$), if we divide $f(x)$ by $g(x)$, we get a quotient $q(x)$ and a remainder $r(x)$ that obey the relationship

$$f(x) = q(x)\ g(x) + r(x)$$

  - can interpret $r(x)$ as being a remainder
  - $r(x) = f(x) \bmod g(x)$

with polynomial degrees:

Degree $f(x) = n$
Degree $g(x) = m$
Degree $q(x) = n - m$
Degree $r(x) \leq m - 1$

If there is no remainder [i.e., $r(x) = 0$], then we can say $g(x)$ **divides** $f(x)$, written as $g(x) | f(x)$. Equivalently, we can say that $g(x)$ is a **factor** of $f(x)$ or $g(x)$ is a **divisor** of $f(x)$.

Irreducible Polynomial

- A polynomial **f(x)** over a field **F** is called irreducible if and only if **f(x)** cannot be expressed as a product of two polynomials, both over **F**, and both of degree lower than that of **f(x).**
- An **irreducible polynomial** is one that cannot be factored into simpler (lower degree) polynomials using the kind of coefficients we are allowed to use, or is not factorisable at all.
- By analogy to integers, an irreducible polynomial is also called a **prime polynomial**.
- The polynomial $f(x) = x^4 + 1$ over GF(2) is reducible, because $x^4 + 1 = (x + 1)(x^3 + x^2 + x + 1)$.

Irreducible/Prime Polynomial

Consider the polynomial $f(x) = x^3 + x + 1$. It is clear by inspection that x is not a factor of $f(x)$. We easily show that $x + 1$ is not a factor of $f(x)$:

$$
\begin{array}{r}
x^2 + x \\
x + 1 \overline{)\, x^3 \qquad + x + 1} \\
\underline{x^3 + x^2} \\
x^2 + x \\
\underline{x^2 + x} \\
1
\end{array}
$$

Thus, $f(x)$ has no factors of degree 1. But it is clear by inspection that if $f(x)$ is reducible, it must have one factor of degree 2 and one factor of degree 1. Therefore, $f(x)$ is irreducible.

International Data Encryption Standard (IDEA)

- a symmetric key **block cipher** encryption algorithm
- first introduced in 1991 to replace DES.
- designed by James Massey of ETH Zurich and Xuejia Lai.
- originally called Proposed Encryption standard, then changed to Improved Proposed Encryption Standard and eventually to IDEA.
- avoids the use of any lookup tables or S-boxes
- is not a Feistel structure.
- used in early version of Pretty Good Privacy (PGP).
- Encryption and decryption are similar.
- IDEA derives most of its security from multiple interleaved mathematical operations:
  - modular addition 2^{16}, $\boxplus$
  - modular multiplication $2^{16} + 1$, $\odot$
  - bitwise exclusive-OR (XOR) $\oplus$

IDEA: Encryption

- IDEA uses a block cipher with a block size of 64 bits. The 64 bit plain text is divided into **4 sub blocks**, each of **16 bits in size**.
- A key size of 128 bits is used and performs 8 identical rounds for encryption in which 6 different subkeys are used.
- After the 8 rounds comes a final "half-round", (the output transformation), in which another 4 keys are used.
- Finally generates 64 bit ciphertext same size as plaintext.

IDEA: Encryption

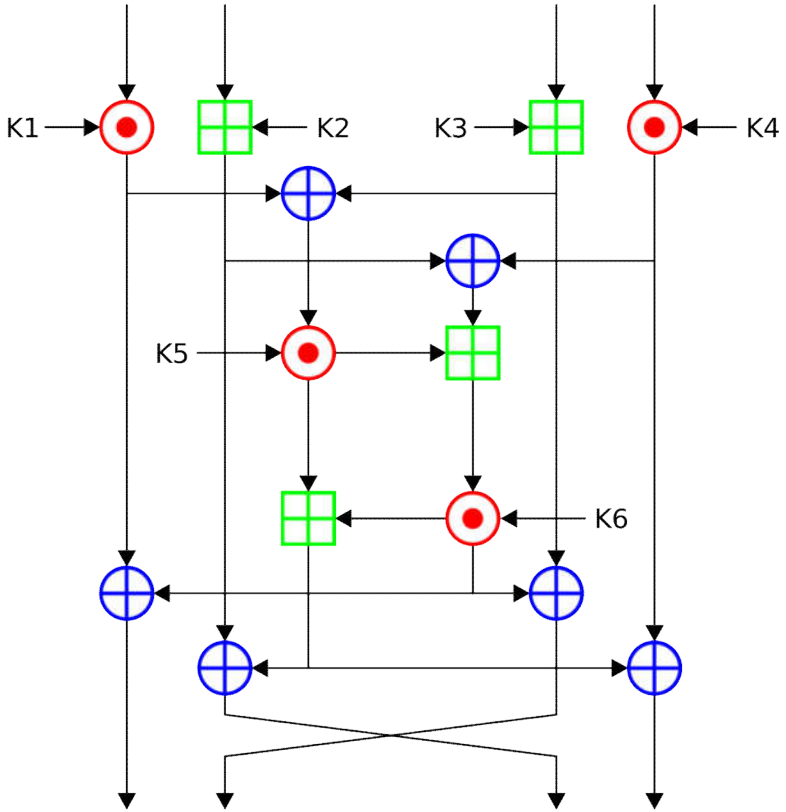


Figure: An encryption round of IDEA

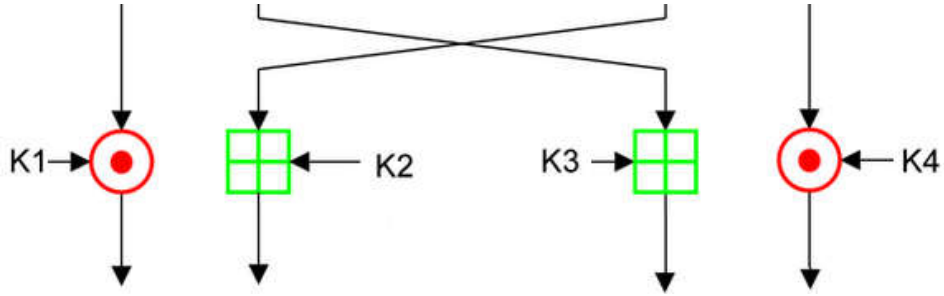


Figure: Half Round in IDEA

IDEA: Key Schedule

- First, the 128-bit key partitioned into eight 16-bit sub-blocks which are the directly used as the first 8 key sub-blocks. These subkeys are referred to as "K1" through "K8".
- The 128-bit key is then cyclically shifted to the left by 25 positions , after which the resulting 128-bit block is again partitioned into eight 16-bit sub-blocks to be directly used as the next eight key sub-blocks. This process creates round-specific subkeys.
- The cyclic shift procedure described above is repeated until all of the required 52 (8*6 +4 = 52) 16-bit key sub-blocks have been generated.

IDEA: Decryption

- **Decryption used same steps as in encryption, but the order of the round keys is inverted, and the subkeys for the odd rounds are inversed**.
- For instance, the values of subkeys K1–K4 are replaced by the inverse of K49–K52 for the respective group operation, K5 and K6 of each group should be replaced by K47 and K48 for decryption.
- So, the computational process used for decryption ciphertext is essentially same as that used for encryption.
- Decryption in IDEA works on the shoes and socks principle, i.e., the last encryption is the first to be removed.

3DES

- The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DEA was designed for mid-1970s hardware implementation and does not produce efficient software code.
- 3DES, which has three times as many rounds as DEA, is correspondingly slower.
- A secondary drawback is that both DEA and 3DES use a 64-bit block size.
- Because of these drawbacks, 3DES is not a reasonable candidate for long-term use.

Advanced Encryption Standard (AES): Origin

- clear a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug-99
- Rijndael was selected as the AES in Oct-2000
  - The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.
- issued as FIPS PUB 197 standard in Nov-2001

AES Criteria

- Security
  - The main emphasis was on security. Because NIST explicitly demanded a 128-bit key, this criterion focused on resistance to cryptanalysis attacks other than brute-force attack.
- Cost
  - The second criterion was cost, which covers the computational efficiency and storage requirement for different implementations such as hardware, software, or smart cards.
- Implementation
  - This criterion included the requirement that the algorithm must have flexibility (be implementable on any platform) and simplicity.

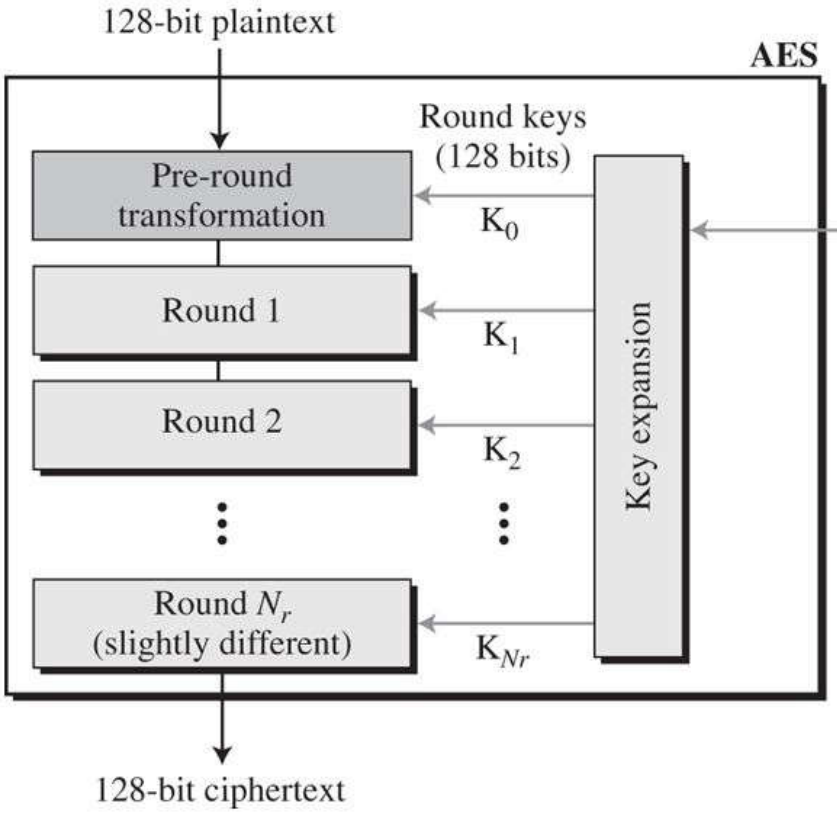
AES Requirements

- private key symmetric block cipher
- 128-bit data,
- 128/192/256-bit keys
- The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length
- AES is a **byte-oriented cipher**. This is in contrast to DES, which makes heavy use of bit permutation and can thus be considered to have a bit-oriented structure.
- stronger & faster than Triple-DES

The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data.
- unlike Feistel cipher, AES encrypts all 128 bits in one iteration.
- an **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round (called state)
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES Structure

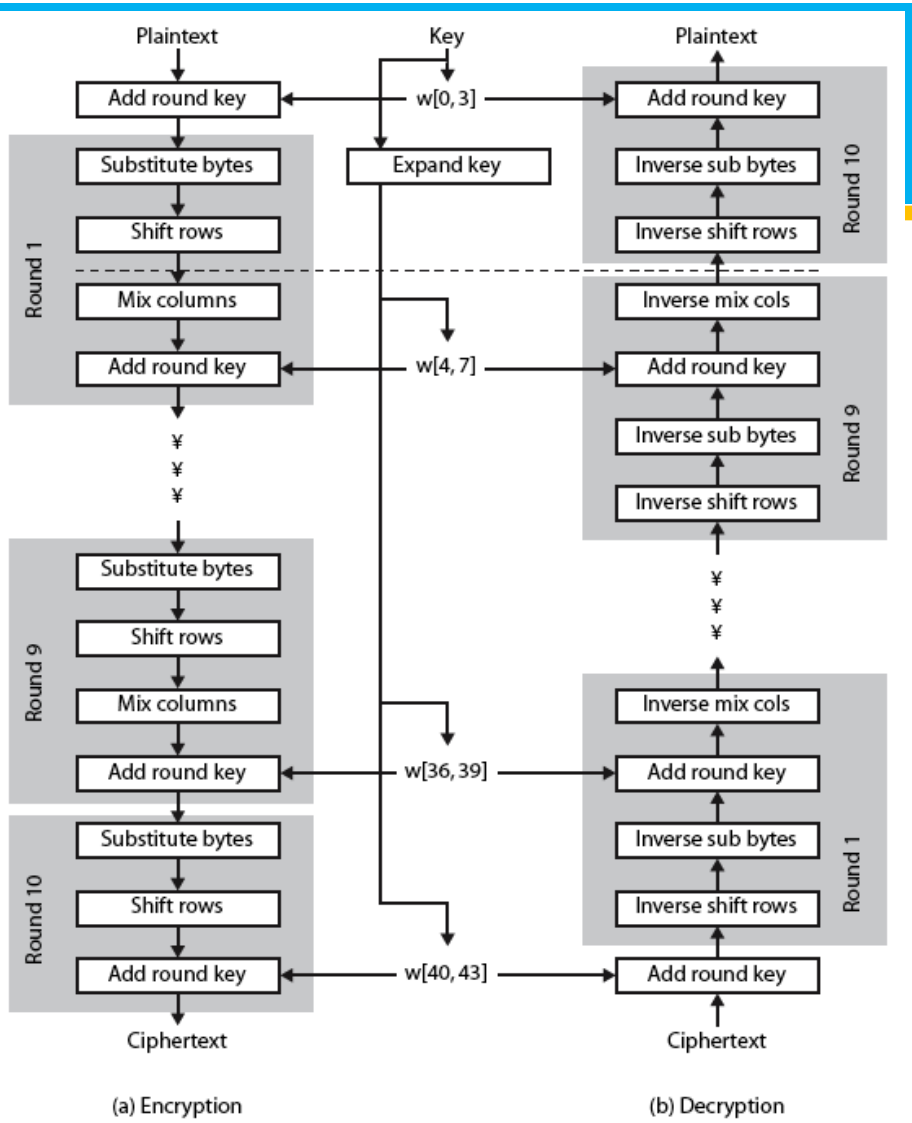


Cipher key
(128, 192, or 256 bits)

Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

Rijndael

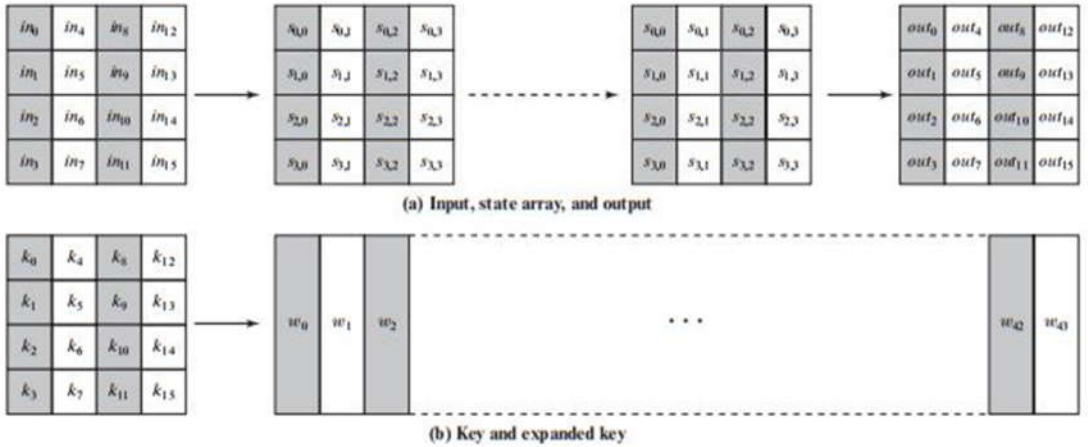


- AES consists of three different types of layers, each of them manipulates all 128 bits on the data path (also called states).
- (Each round, with the exception of the first, consists of all three layers.
  1. Key Addition Layer
  2. Byte Substitution layer (S-box)
  3. Diffusion layer
- Key Scheduling

- **Key Addition Layer**
  - A 128 bit round key, or subkey, which has been derived from the main key in the key schedule, is XORed to the state (or data path).
- **Byte Substutution Layer (S-Box)**
  - Each element of the state is non-linearly transformed (by providing confusion) using lookup tables with special mathematical properties.
- **Diffusion Layer**
  - It provides diffusion over all state bits. It further consists of two sub layers:
  - The ShiftRows layer permutes the data on a byte level.
  - The MixColumn layer combines blocks of four bytes.
- **Key Scheduling**
  - Similar to DES, the key schedule computes round keys or subkeys (K_0 … K_{nr}) form the original key.

AES Structure

- Data block of 4 columns of 4 bytes is state
- Key is expanded to array of words
  - 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.



(a) Input, state array, and output

(b) Key and expanded key

AES Structure

- 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(2^8).
  - **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded 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

AES Structure

- 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.
- Only the AddRoundKey stage makes use of the key.
- Each stage is easily reversible. . For the **Substitute Byte**, **ShiftRows**, and **MixColumns** stages, an **inverse function** is used in the decryption algorithm. For the **AddRoundKey** stage, the inverse is achieved by **XORing** the same round key to the block, using the result that A ⊕B ⊕B = A
- The decryption algorithm makes use of the expanded key in reverse order.
- At each horizontal point, (e.g. the dashed line in the figure), State is the same for both encryption and decryption.
- The final round of both encryption and decryption consists of only three stages.

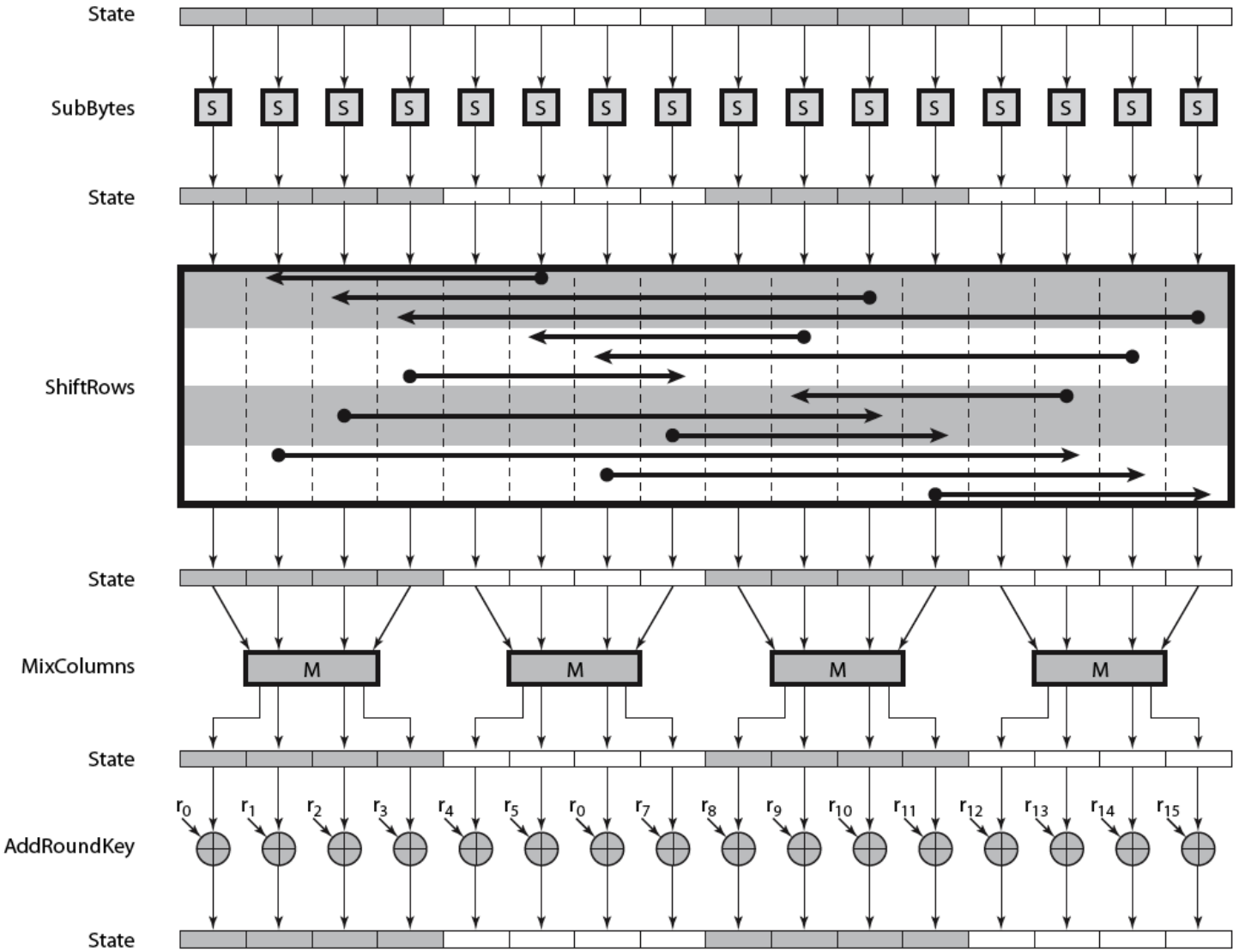


Figure: AES encryption round

Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes (called **S-box**) containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- These row and column values serve as indexes into the S-box to select a unique 8-bit output value.
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks

Byte Substitution

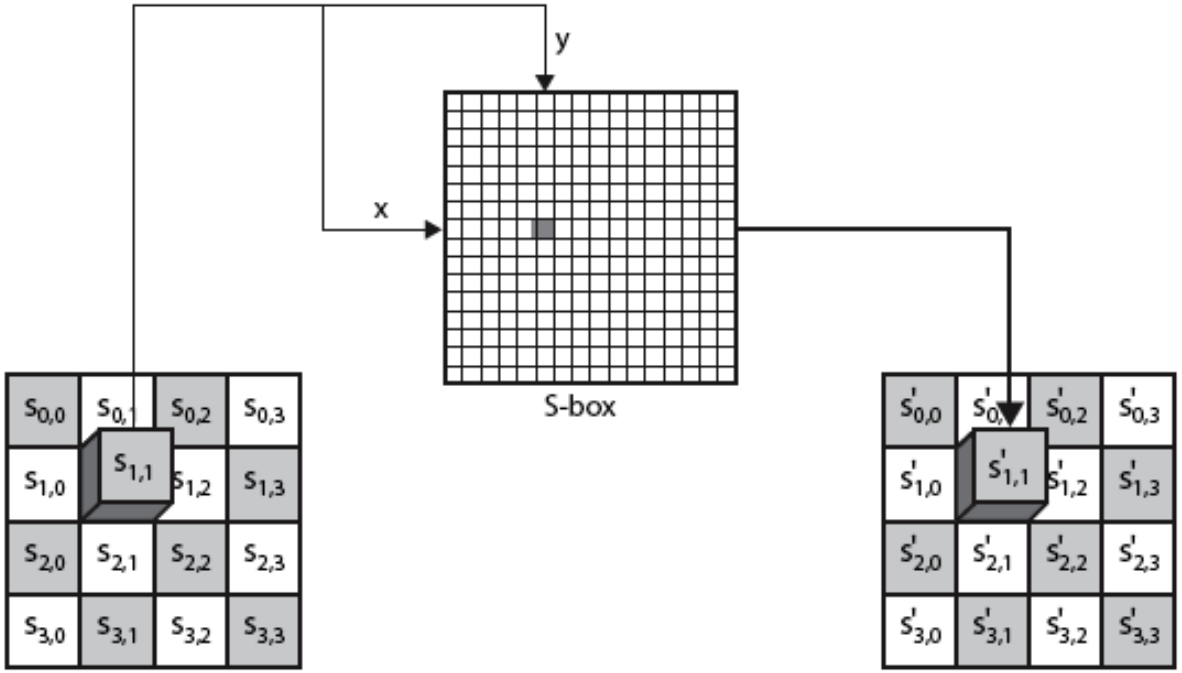


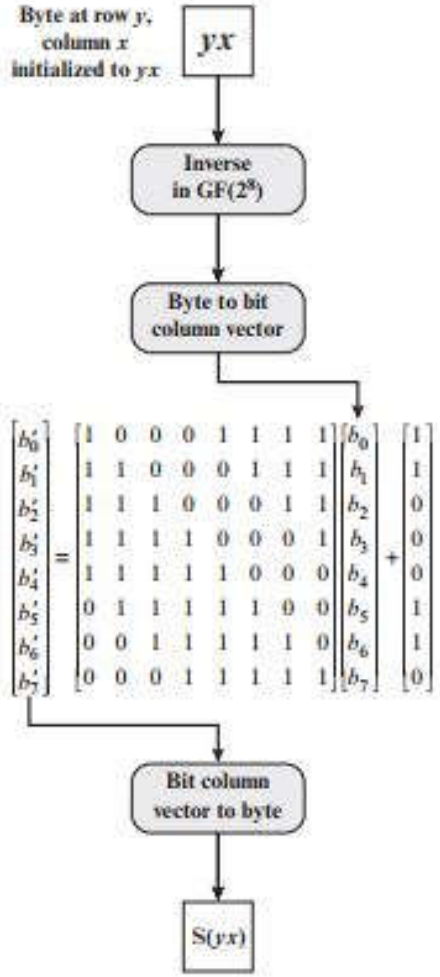
Figure: As this diagram shows, the Byte Substitution operates on each byte of state independently, with the input byte used to index a row/col in the table to retrieve the substituted value.

AES S-Box

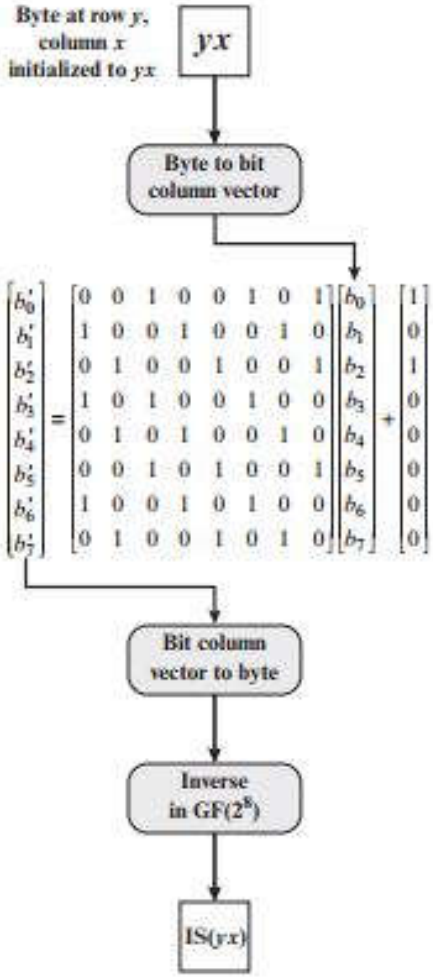
		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	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

AES inverse 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



(a) Calculation of byte at row y, column x of S-box



(a) Calculation of byte at row y, column x of IS-box

Figure: Construction of S-Box and IS Box

Example

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

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

Diffusion Layer

- Diffusion Layer
  - Shift Rows
  - Mix Column

Shift Rows

- The **forward shift row transformation**, called ShiftRows.
- The ShiftRows transformation cyclically shifts the rows of the state matrix by certain number of bytes to the right, except the first row.
- a circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Shift Rows

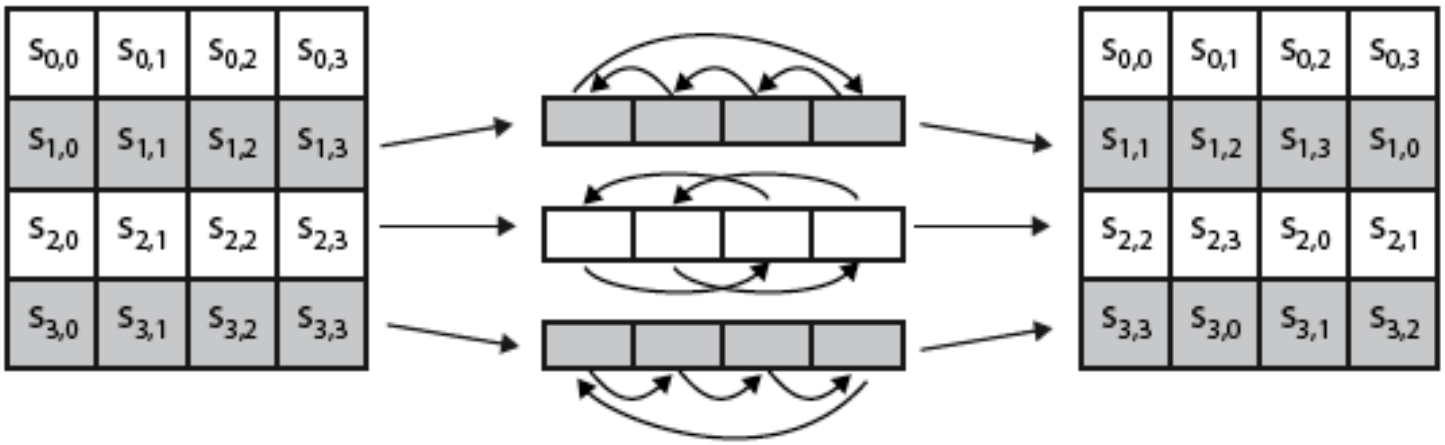


Figure: Shift Row Transformation

Mix Columns

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

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

Mix Columns

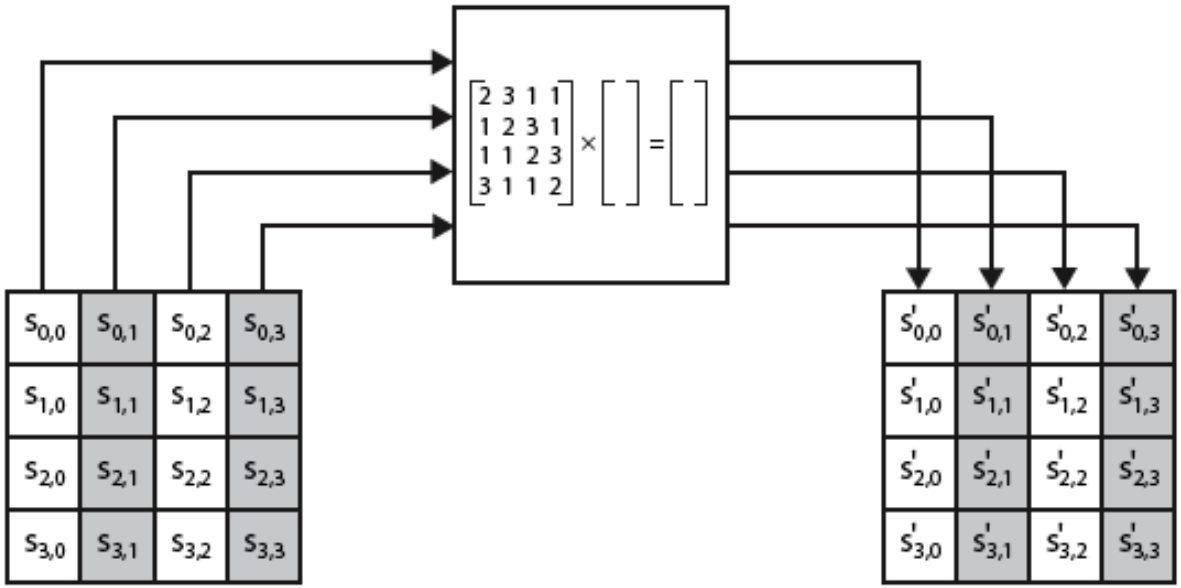


Figure: Mix column transformation

Example: Mix Columns

- The MixColumns transformation in AES involves multiplying the state matrix (a 4x4 matrix of bytes) by a fixed matrix called the **MixColumns matrix**.
- Each byte in the state matrix is *treated as a polynomial* over the finite field GF(2^8), and the multiplication is performed modulo a specific polynomial defined in AES.

Suppose we have 4x4 state matrix as:

S = | 0x32 0x88 0x31 0xe0 |
| 0x43 0x5a 0x31 0x37 |
| 0xf6 0x30 0x98 0x07 |
| 0xa8 0x8d 0xa2 0x34 |

And here's the MixColumn matrix:

M = | 0x02 0x03 0x01 0x01 |
| 0x01 0x02 0x03 0x01 |
| 0x01 0x01 0x02 0x03 |
| 0x03 0x01 0x01 0x02 |

Example: Mix Columns

1. To perform the MixColumns transformation:

- Take the first column of the state matrix and treat it as a polynomial:

| 0x32 |
| 0x43 |
| 0xf6 |
| 0xa8 |

2. Multiply each element of the column by the corresponding element in the MixColumns matrix and perform the calculations in the finite field GF(2^8):

- Multiply 0x32 by 0x02 and reduce modulo 0x11b (AES polynomial): 0x32 * 0x02 = 0x64, reduced modulo 0x11b gives 0x64 % 0x11b = 0x64.
- Multiply 0x43 by 0x03 and reduce modulo 0x11b: 0x43 * 0x03 = 0xc9, reduced modulo 0x11b gives 0xc9 % 0x11b = 0xc9.
- Multiply 0xf6 by 0x01 (no change).
- Multiply 0xa8 by 0x01 (no change).

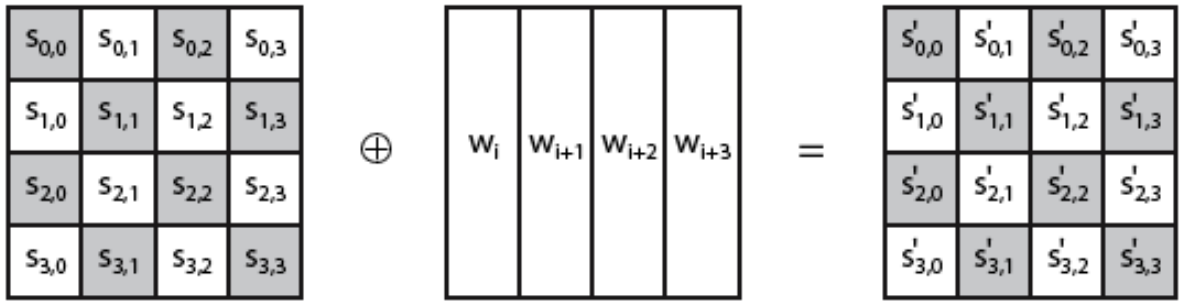
3. Replace the original values in the first column with the results:

| 0x64 |
| 0xc9 |
| 0xf6 |
| 0xa8 |

4 Repeat the same process for each of the remaining columns in the state matrix using the MixColumns matrix. After the MixColumns transformation, you will have a new state matrix with the columns modified to provide diffusion in the data. This is a crucial step in the AES encryption process that helps achieve the encryption's security properties.

Add Round Key

- The two inputs to the Key Addition layer are the current 16-byte state matrix and a subkey which also consists of 16 bytes (128 bits).
- The two inputs are combined through a bitwise XOR operation. Note that the XOR operation is equal to addition in the Galois field GF(2).
  - Subkey is obtained from key schedule
- 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. The following is an example of AddRoundKey:



The first matrix is **State**, and the second matrix is the round key.
The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

Example: Add Round Key

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

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

$=$

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

Key Scheduling

- see your self.

Modes of Block Cipher Encryptions

- A symmetric block cipher processes one block of data at a time.
- In the case of DES and 3DES, the block length is 64 bits. For longer amounts of plaintext, it is necessary to break the plaintext into 64-bit blocks (padding the last block if necessary).

A block cipher takes a fixed-length block of text of length *b* bits and a key as input and produces a *b*-bit block of ciphertext. If the amount of plaintext to be encrypted is greater than *b* bits, then the block cipher can still be used by breaking the plaintext up into *b*-bit blocks.

- To apply a block cipher in a variety of applications, five *modes of operation* have been defined by NIST.
  1. Electronics Code Block
  2. Cipher Block Chaining
  3. Cipher Feedback Mode
  4. Output Feedback Mode
  5. Counter Mode

Electronic Codebook Book (ECB)

- plaintext is handled one block at a time and each block of plaintext is encrypted using the same key.
- The term *codebook* is used because, for a given key, there is a unique ciphertext for every *b*-bit block of plaintext.
- 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.

$$\text{Encryption: } C_j = E(K, P_j), \quad j = 0, 1, 2, \ldots N$$

$$\text{Decryption: } P_j = D(K, C_j), \quad j = 0, 1, 2, \ldots N$$

uses: secure transmission of single values

Electronic Codebook Book (ECB)

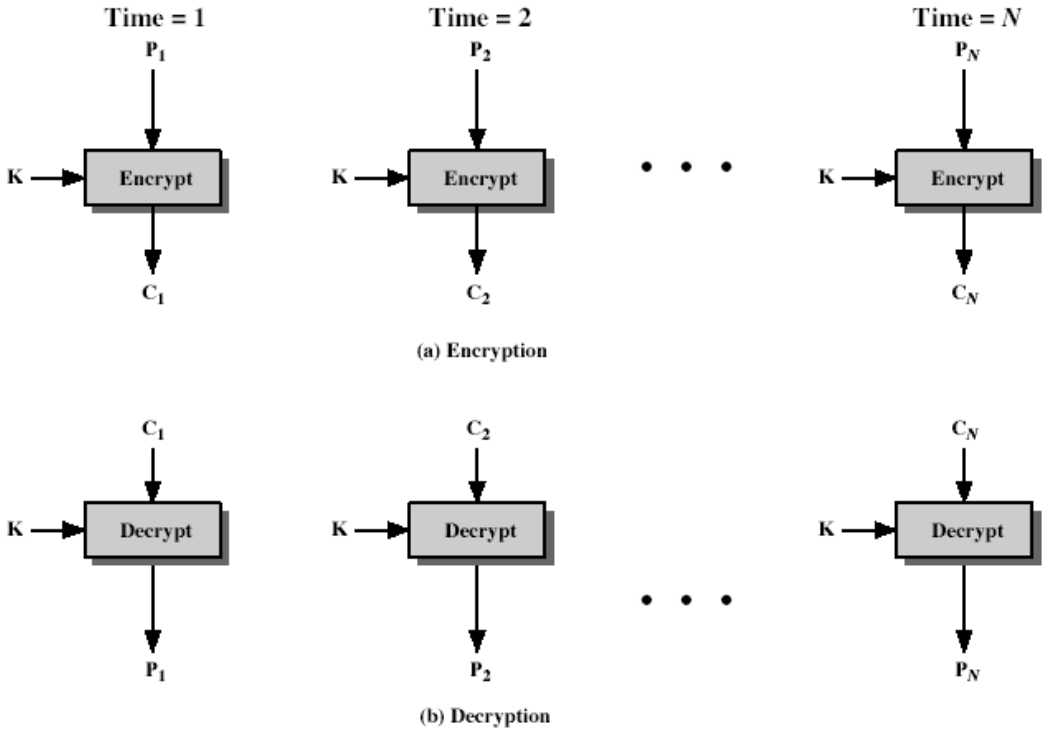


Figure: Electronic Codebook (ECB) Mode

Advantages and Limitations of ECB

- message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness is due to the encrypted message blocks being independent
- main use is sending a few blocks of data

Cipher Block Chaining Mode

- overcome security deficiencies of ECB → the same plaintext block, **if repeated**, produces different ciphertext blocks.
- message is broken into blocks and linked together in encryption operation.
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process
- uses: bulk data encryption, authentication

Cipher Block Chaining Mode

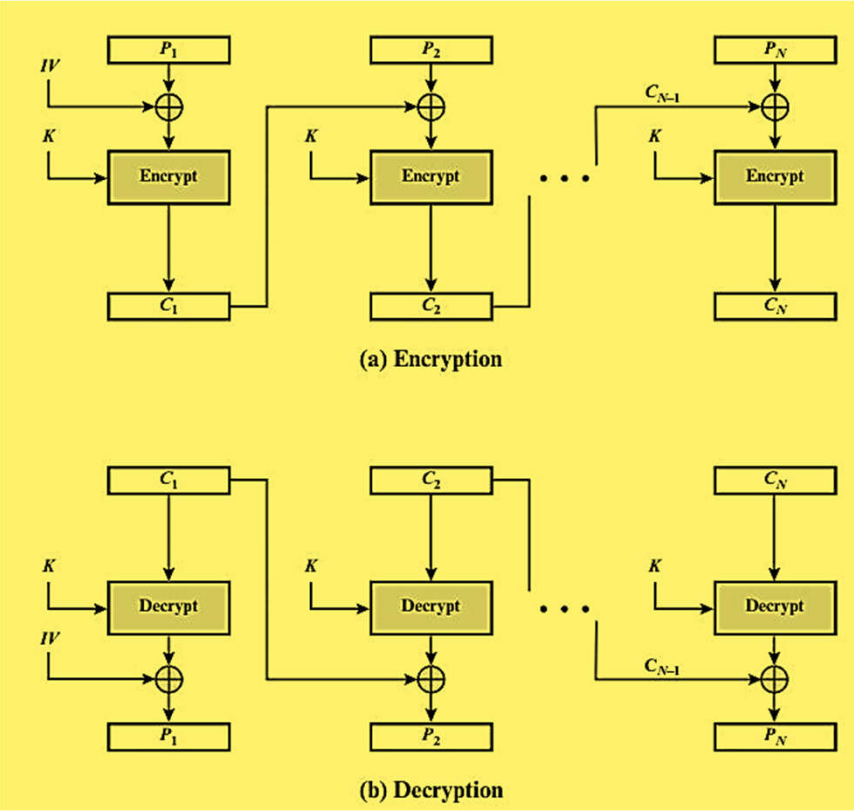


Figure: Cipher Block Chaining (CBC) Mode

Encryption:

$C_1 = E(K, IV \oplus P_j)$

$C_j = E(K, [C_{j-1} \oplus P_j]), \; j = 2, 3 \ldots, N$

Then

Decryption:

$P_1 = D(K, C1) \oplus IV$

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

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

$C_{j-1} \oplus D(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P$

Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks
- need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value
  - or must be sent encrypted in ECB mode before rest of message

Cipher FeedBack (CFB) Mode

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)
- uses: stream data encryption, authentication

CFB Mode

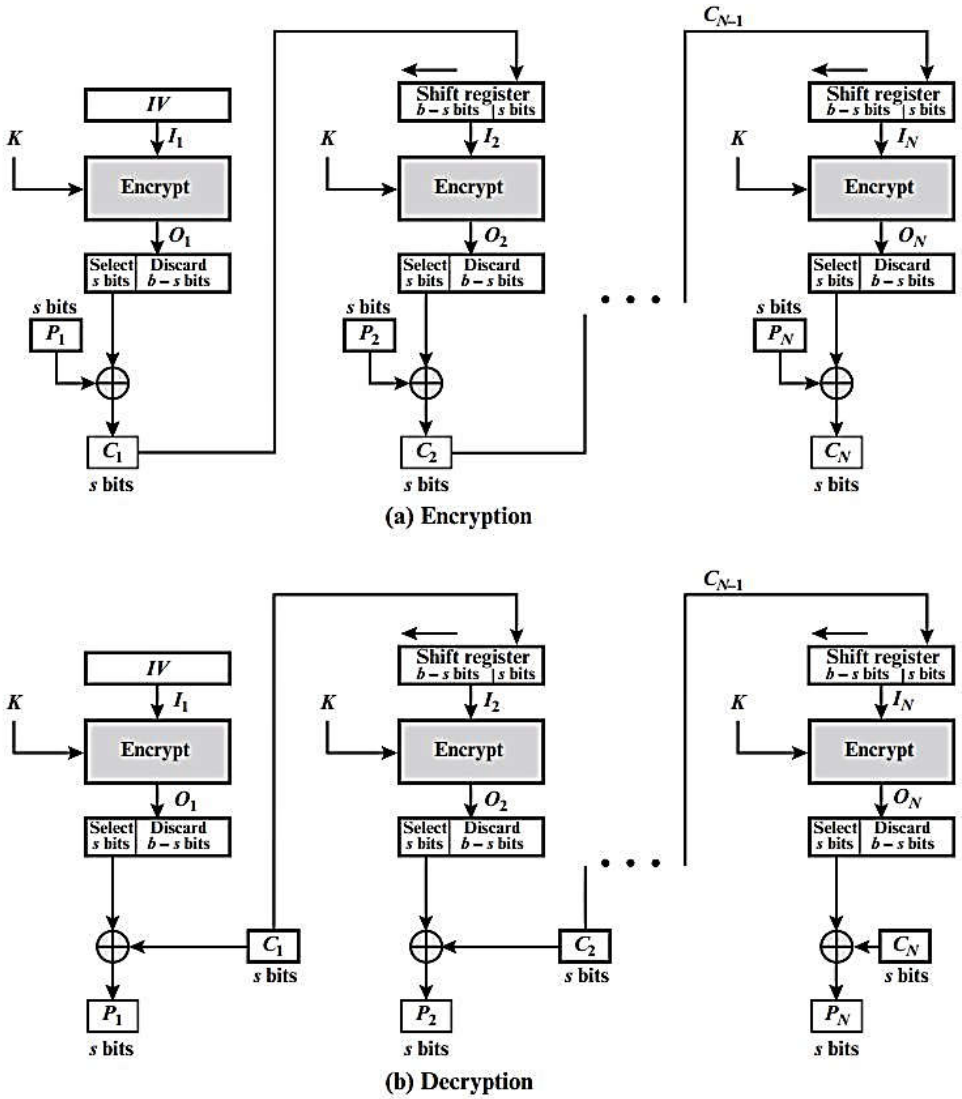


Figure: s-bit CFB

- Figure depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is **s** bits; a common value is **s = 8**.
- As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext.
- In this case, rather than blocks of **b** bits, the plaintext is divided into segments of **s** bits.

CFB Mode: 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.

CFB Mode: Decryption

- The same scheme is used as in encryption, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.
- We define CFB mode as follows:

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

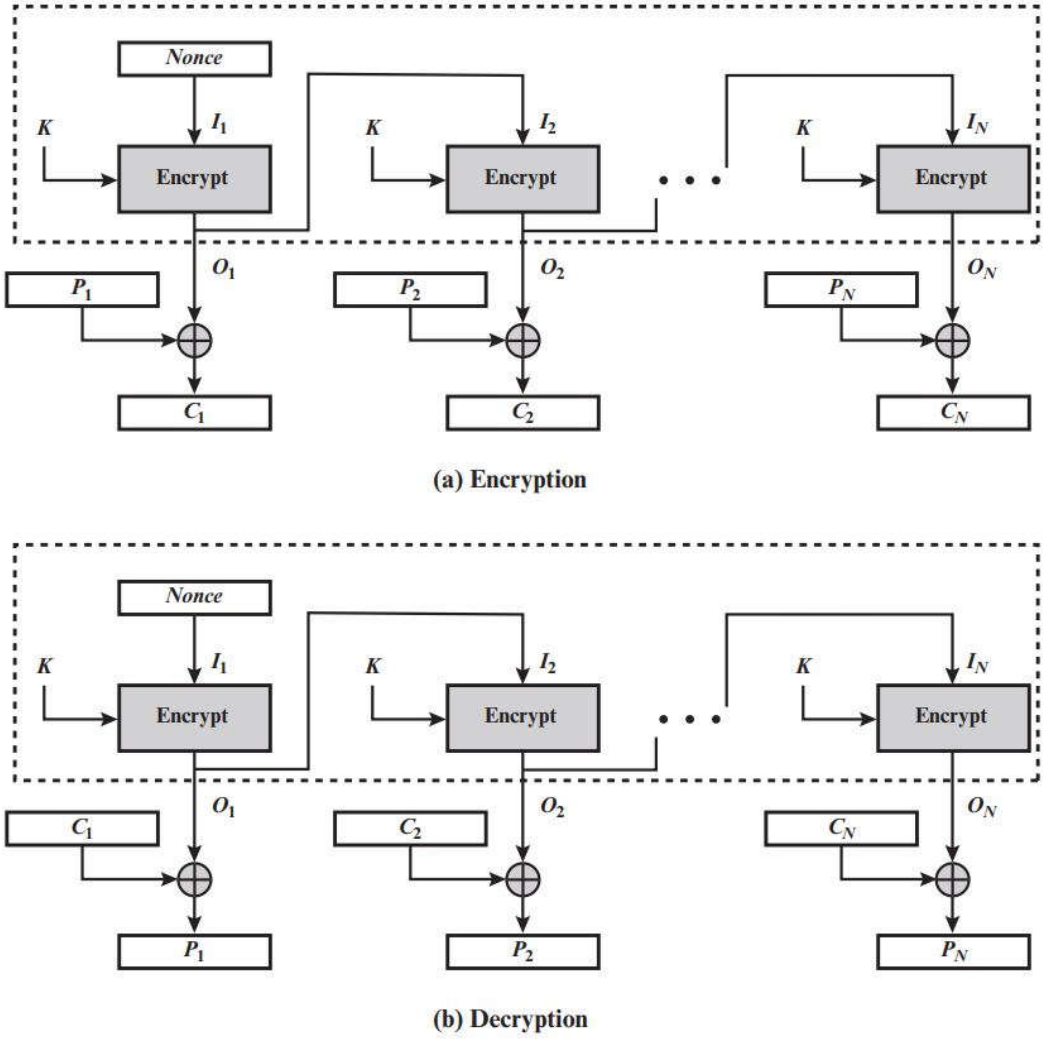
Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- In CFB encryption, like CBC encryption, the input block to each forward cipher function (except the first) depends on the result of the previous forward cipher function; therefore, multiple forward cipher operations cannot be performed in parallel.
- In CFB decryption, the required forward cipher operations can be performed in parallel if the input blocks are first constructed (in series) from the IV and the ciphertext.
- errors propogate for several blocks after the error

Output Feedback Mode

- The **output feedback (OFB)** mode is similar in structure to that of CFB.
- For OFB, the output of the encryption function is fed back to become the input for encrypting the next block of plaintext.
- In CFB, the output of the XOR unit is fed back to become input for encrypting the next block.
- The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, whereas CFB operates on an *s*-bit subset.

Output Feedback Mode



Encryption:

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

where

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

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

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

Decryption:

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

Figure: Output Feedback (OFB) Mode

Advantages and Limitations of OFB

- Advantage: bit errors do not propagate
  - For example, if a bit error occurs in *C*1, only the recovered value of *P*1 is affected; subsequent plaintext units are not corrupted.
- The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.
- a variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- originally specified with m-bit feedback
- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

Counter (CTR) Mode

- a “new” mode, though proposed early in 1979.
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

$$C_i = P_i \text{ XOR } O_i$$
$$O_i = DES_{K1}(i)$$

- uses: ATM (asynchronous transfer mode) network security and IPsec (IP security)

Counter (CTR) Mode

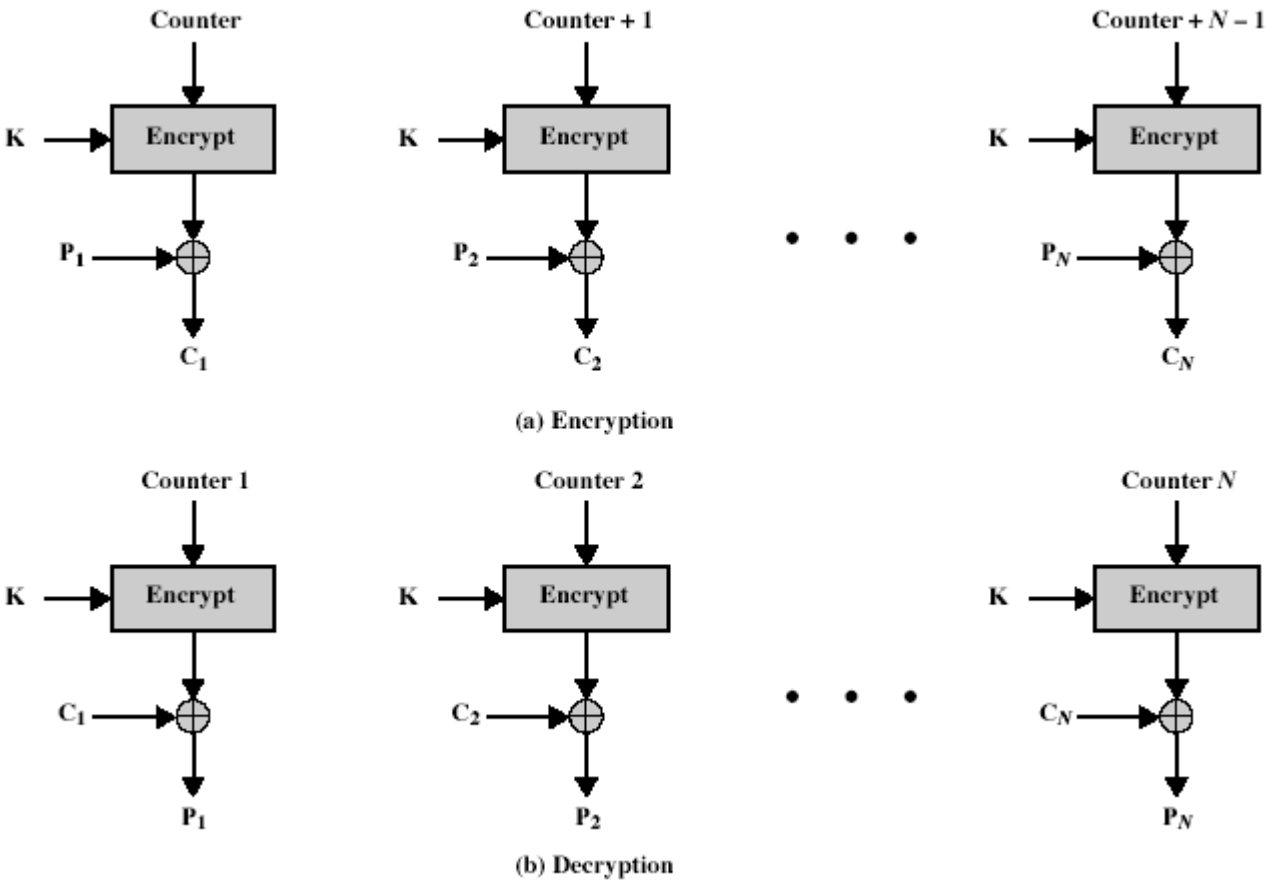


Figure: Counter (CTR) mode

- Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2*b*, where *b* is the block size).
- **For encryption**, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining.
- **For decryption**, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.
- **Thus**, the initial counter value must be made available for decryption

Counter (CTR) Mode

- Given a sequence of counters T_1, T_2, … , TN, we can define CTR mode as follows:

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

For the last plaintext block, which may be a partial block of u bits, the most significant u bits of the last output block are used for the XOR operation; the remaining $b - u$ bits are discarded. Unlike the ECB, CBC, and CFB modes, we do not need to use padding because of the structure of the CTR mode

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

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

Figure: Block Cipher Modes of Operation