Unit-2: Symmetric Ciphers

- In Symmetric Ciphers single key is used for both encryption and decryption and the key is exchanged through secure channel.
- Also known as private key cryptosystem.

Concept of Confusion and Diffusion

- The terms diffusion and confusion were introduced by the famous information theorist Claude Shannon to capture the two basic building blocks for any cryptographic system
- According to the Shannon, there are two primitive operations with which strong encryption algorithms can be built:
    - **Confusion** is an encryption operation where the relationship between key and ciphertext is obscured (making unclear and difficult to understand)
      Example: Substitution table (look-up table)
    - **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
      Example: Permutation

      Modern block ciphers possess excellent diffusion properties. On a cipher level this means that changing of one bit of plaintext results on average in the change of half the output bits and similarly, if we change one bit of the ciphertext, then approximately one half of the plaintext bits should change

      Example : Let's assume a small block cipher with a block length of 8 bits. Encryption of two plaintexts x1 and x2, which differ only by one bit, should roughly result in something as shown in figure below

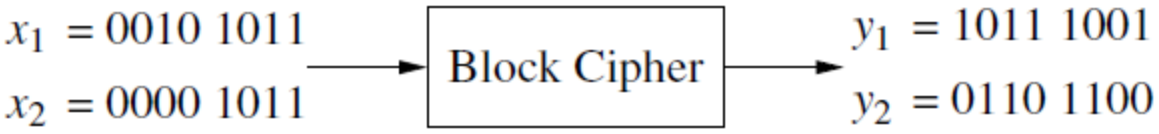


Figure: Principle of diffusion of a block cipher

- Strong block cipher can be built by combining confusion and diffusion many times.

Fiestel Cipher Structure

- Proposed by German-American Cryptographer Horst Feistel in 1973.
- A symmetric structure used in the construction of block ciphers
- A practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions
- 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.
- Feistel cipher alternates substitutions and permutations, where these terms are defined as follows:
    - Substitution: Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.
    - Permutation: A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

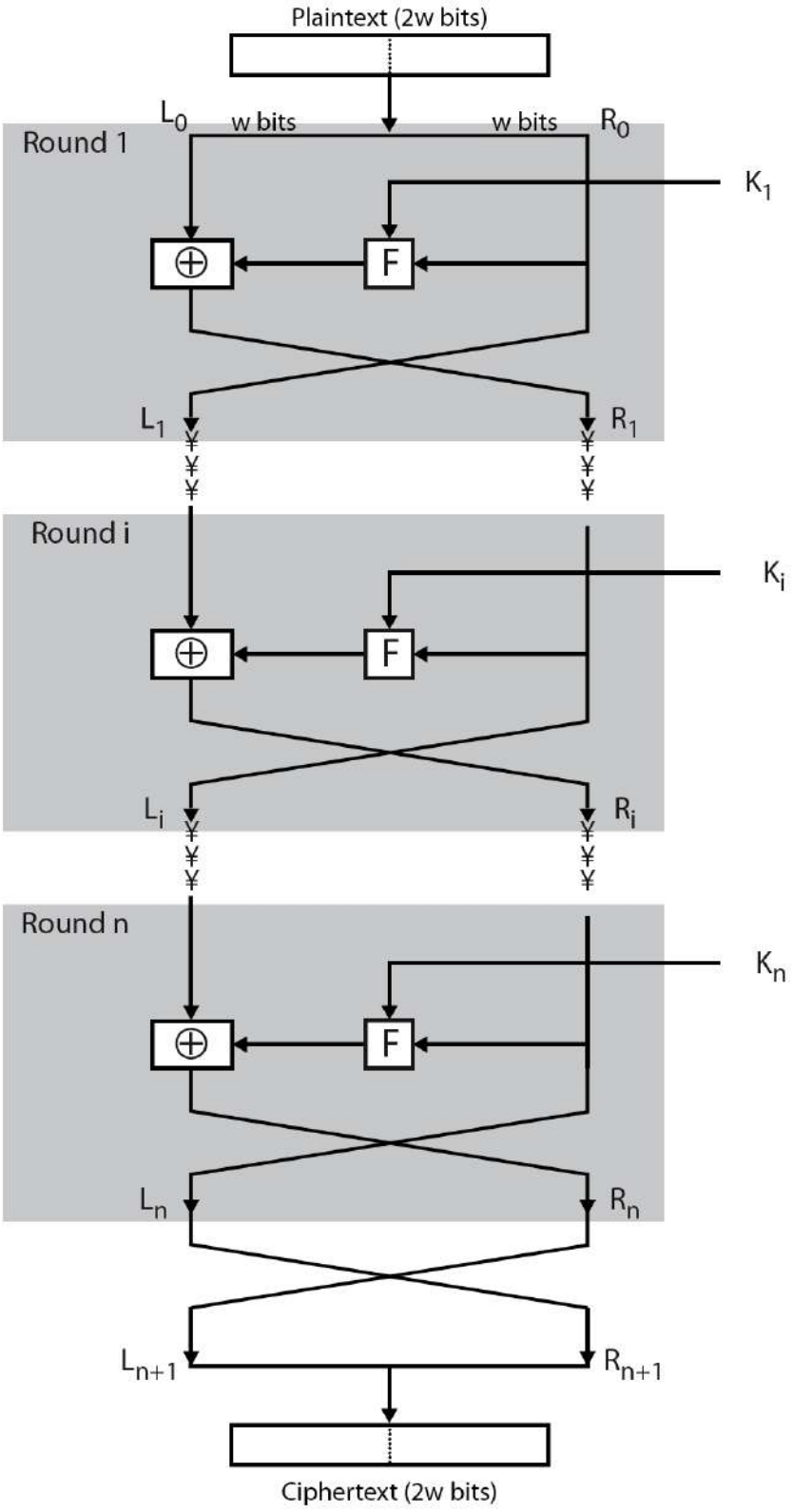


Fig: Fiestel Cipher Structure

Encryption Process

- The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K
- The input block to each round is divided into two halves that can be denoted as L and R for the left half and the right half
- The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block.
- Each round i has as inputs L_{i-1} and R_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys K_i are different from K and from each other
- All rounds have the same structure. A **substitution** is performed on the **left half** of the data. This is done by applying a **round function F** to the **right half** of the data and then taking the **exclusive-OR** of the output of that function and the left half of the data.
- The round function has the same general structure for each round but is parameterized by the round subkey K_i.
    - Another way to express this is to say that F is a function of right-half block of w bits and a subkey of y bits, which produces an output value of length w bits: **$F(RE_i, K_{i+1})$**
- Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data
- Once the last round is completed then the two sub blocks, 'R' and 'L' are concatenated in this order to form the ciphertext block.

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

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm.
- **Key size:** Larger key size means greater security but may decrease encryption/ decryption speed.

- **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function F**: Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

Decryption Process

- The process of decryption with a Feistel cipher is essentially the same as the encryption process.
- The rule is as follows:
    - Use the ciphertext as input to the algorithm, but use the subkeys Ki in reverse order.
    - That is, use K_n in the first round, K_{n-1} in the second round, and so on, until K_1 is used in the last round

Following figure shows the encryption and decryption process in Fiestel Structure with 16 rounds. The encryption process going down the left-hand side and the decryption process going up the right-hand side of the figure for a 16-round algorithm.

For clarity, notation LE_i and RE_i are used for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped

For 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,

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

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

Feistel structure really only encrypts (decrypts) half of the input bits per each round, namely the left half of the input. The right half is copied to the next round unchanged

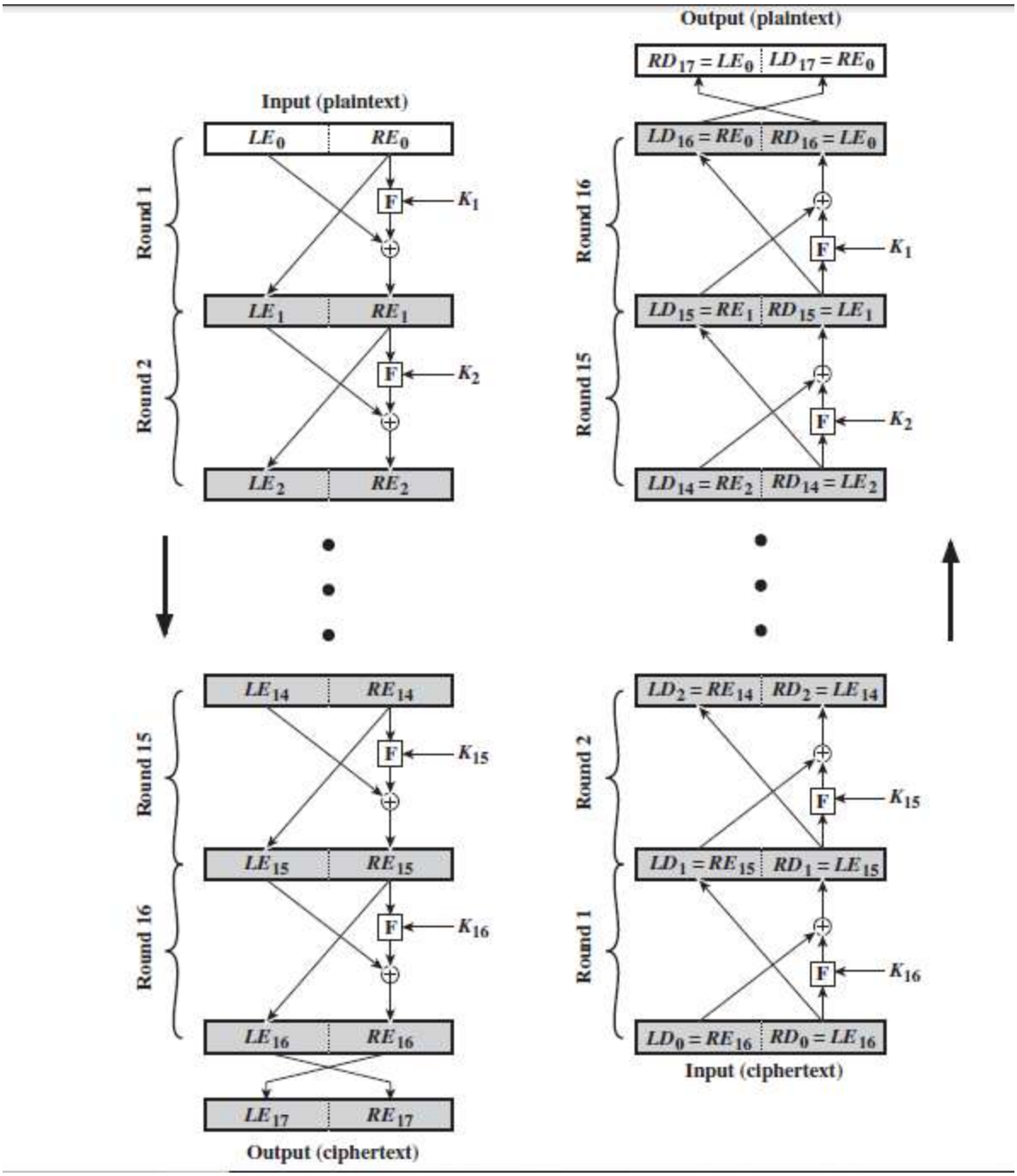


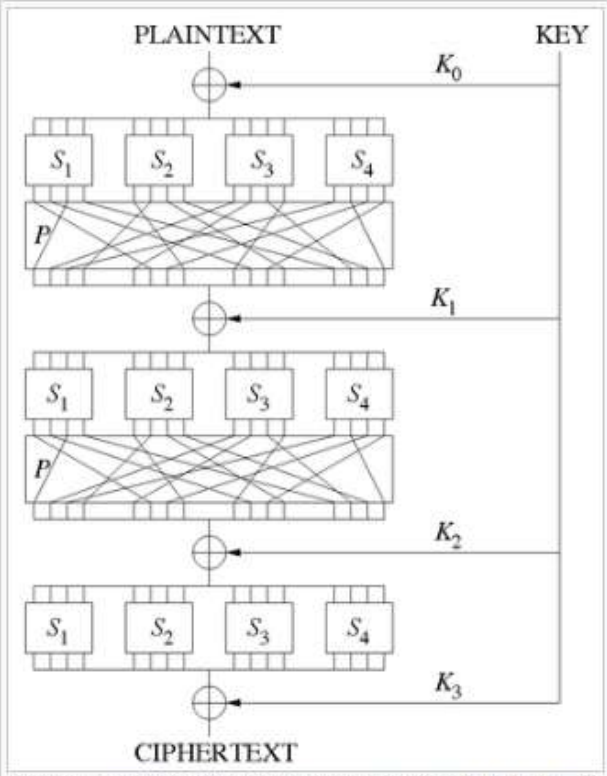
Figure: Fiestel Encryption and Decryption (16 rounds)

Substitution Permutation Network

- An SP-network, or substitution–permutation network (SPN), is a series of linked mathematical operations used in block cipher algorithms such as AES. Such a network takes a block of the plaintext and the key as inputs, and applies several alternating "rounds" or "layers" of substitution boxes (S-boxes) and permutation boxes (P-boxes) to produce the ciphertext block. The S-boxes and P-boxes transform (sub-)blocks of input bits into output bits. It is common for these transformations to be operations that are efficient to perform in hardware, such as exclusive or (XOR) and bitwise rotation. The key is introduced in each round, usually in the form of "round keys" derived from it. Decryption is done by simply reversing the process (using the inverses of the S-boxes and P-boxes and applying the round keys in reversed order).
- An S-box substitutes a small block of bits (the input of the S-box) by another block of bits (the output of the S-box). This substitution should be one-to-one, to ensure invertibility (hence decryption). In particular, the length of the output should be the same as the length of the input (the picture below has S-boxes with 4 input and 4 output bits), which is different from S-boxes in general that could also change the length, as in DES (Data Encryption Standard), for example. Rather, a good S-box will have the property that changing one input bit will change about half of the output bits (or an avalanche effect). It will also have the property that each output bit will depend on every input bit.
- A P-box is a permutation of all the bits: 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.
- A single typical S-box or a single P-box alone does not have much cryptographic strength: an S-box could be thought of as a substitution cipher, while a P-box could be thought of as a transposition cipher. However, a well-designed SP network with several alternating rounds of S- and P-boxes already satisfies Shannon's confusion and diffusion properties:
- The reason for diffusion is the following: If one changes one bit of the plaintext, then it is fed into an S-box, whose output will change at several bits, then all these changes are distributed by the P-box among several S-boxes, hence the outputs of all of these S-boxes are again changed at several bits, and so on. Doing several rounds, each bit changes several times back

and forth, therefore, by the end, the ciphertext has changed completely, in a pseudorandom manner. In particular, for a randomly chosen input block, if one flips the i-th bit, then the probability that the j-th output bit will change is approximately a half, for any i and j, which is the Strict Avalanche Criterion. Vice versa, if one changes one bit of the ciphertext, then attempts to decrypt it, the result is a message completely different from the original plaintext—SP ciphers are not easily malleable.

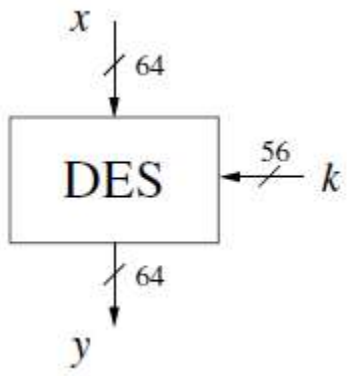
- The reason for confusion is exactly the same as for diffusion: changing one bit of the key changes several of the round keys, and every change in every round key diffuses over all the bits, changing the ciphertext in a very complex manner.
- Even if an attacker somehow obtains one plaintext corresponding to one ciphertext—a known-plaintext attack, or worse, a chosen plaintext or chosen-ciphertext attack—the confusion and diffusion make it difficult for the attacker to recover the key.



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's, the P-boxes are the same P, and the round keys are the K_i's.

Data Encryption Standards (DES)

- Data Encryption Standard (DES) was the most widely used encryption scheme.
- DES was issued in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST).
- The algorithm itself is referred to as the Data Encryption Algorithm (DEA)
- DES is a symmetric-key block cipher. It is based on Feistel Cipher structure.
- DES is a cipher which encrypts blocks of length of 64 bits with a key of size of 56 bits.
- DES encrypts data in blocks of size of 64 bit each, means 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences



DES Encryption

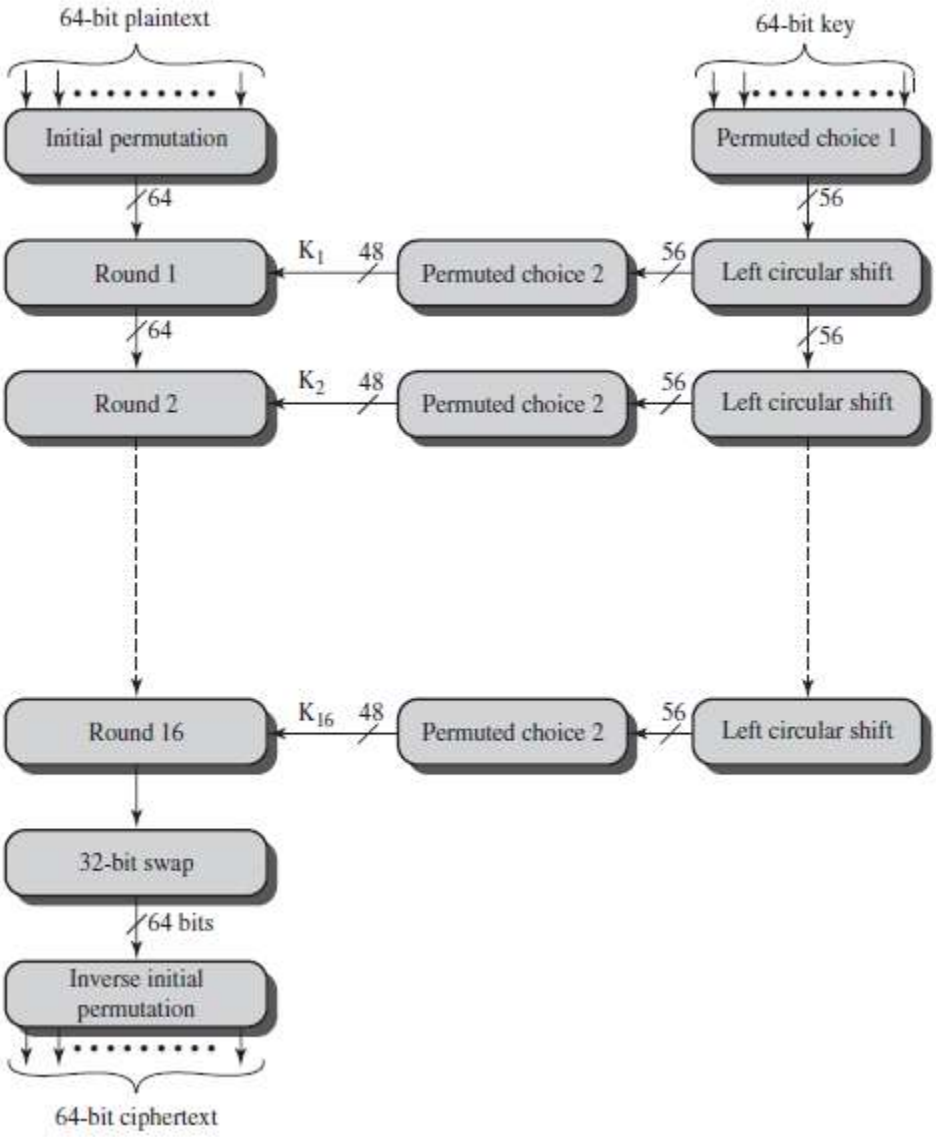


Figure: General Depiction of DES Encryption Algorithm

⇒ The overall scheme for DES encryption is illustrated in above figure.

⇒ As with any encryption scheme, there are two inputs to the encryption function:

- *The plaintext to be encrypted* and
- *The key*.

⇒ In DES, the plaintext must be 64 bits in length and the key is 56 bits in length.

⇒ Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases.

1. Initial permutation (Permute 64 bits in a block)
2. Sixteen rounds of the same function, which involves both permutation and substitution function (Apply given operation 16 times on the 64 bits)
3. Inverse permutation (Permute the 64 bits using the inverse of initial permutation)

→ 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 preoutput.

→ Finally, the preoutput is passed through a permutation [IP^{-1}] that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

⇒ With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

⇒ 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.

⇒ Compared to encryption, only the key schedule is reversed, i.e., in decryption round 1, subkey 16 is needed; in round 2, subkey 15; etc.

⇒ Thus, when in decryption mode, the key schedule algorithm has to generate the round keys as the sequence k16,k15, . . . ,k1.

⇒ Additionally, the initial and final permutations are reversed.

Internal Structure of DES

- The building blocks of DES :
  - The initial and final permutation
  - The F–function
  - The key schedule

Initial and Final Permutation

- Initial permutation IP and the final permutation IP^{-1} are bitwise permutations.
- A bitwise permutation can be viewed as simple crosswiring
- The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other.
- They have no cryptography significance in DES

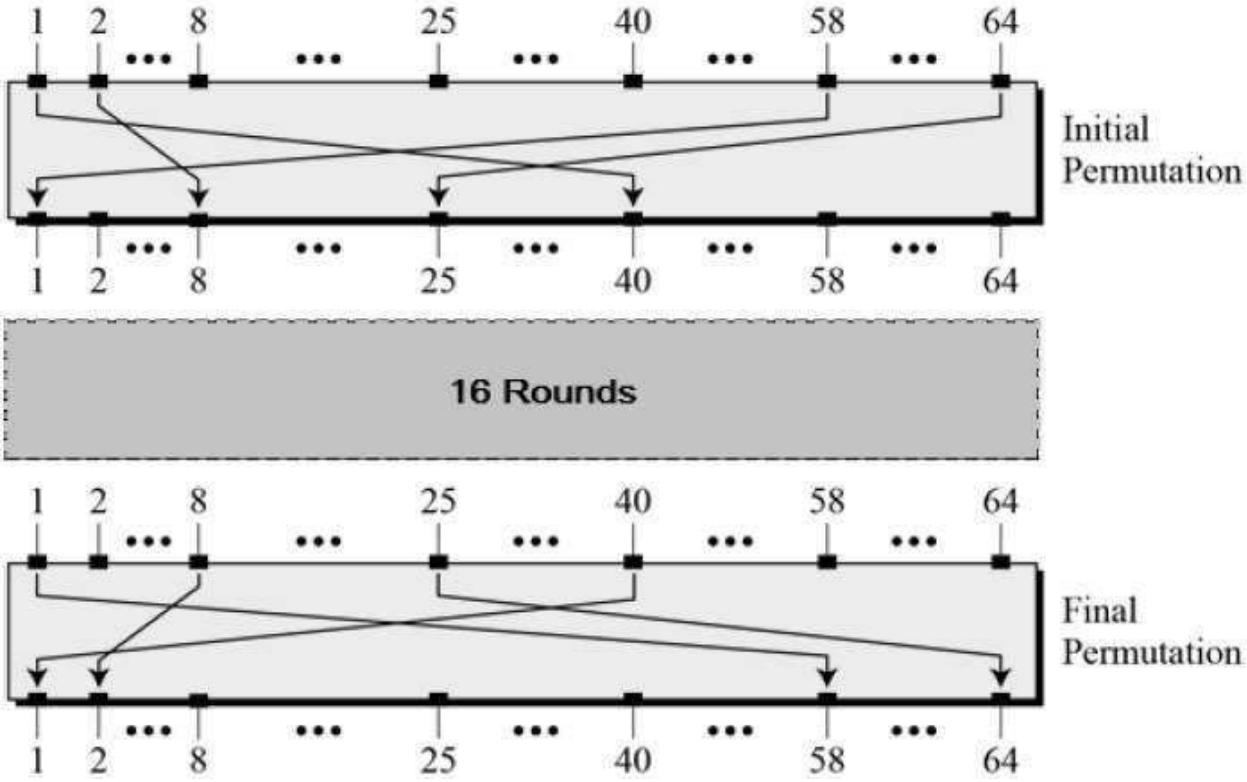


Figure: Examples for the bit swaps of the initial and final permutation

→ The permutation rules for these P-boxes are shown below. Each box can be thought of as a 64-element array

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

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

This table should be read from left to right, top to bottom. The table (IP) indicates that input bit 58 is mapped to output position 1, input bit 50 is mapped to the second output position, and so forth.

The F-Function

- The F -function plays a crucial role for the security of DES.
- In round i it takes the right half R_{i-1} of the output of the previous round and the current round key k_i as input.
- The output of the F -function is used as an XOR-mask for encrypting the left half input bits L_{i-1}

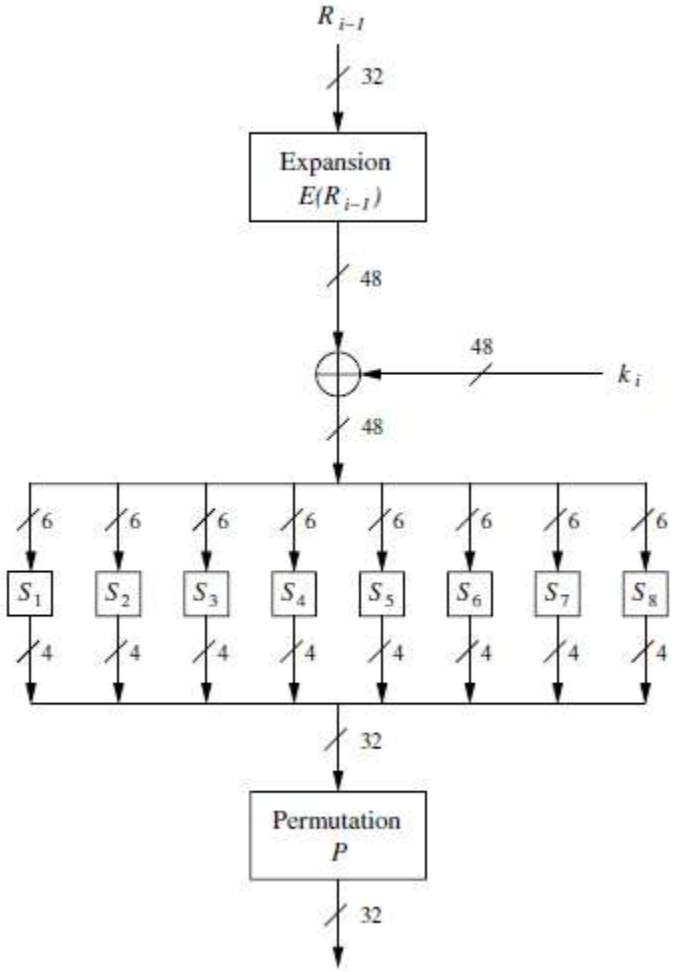


Figure : Block diagram of the F –function in DES

Four steps of F function

- Expansion E
- XOR with round key
- S-Box substitution
- Permutation

→ First, the 32-bit input is expanded to 48 bits by partitioning the input into eight 4-bit blocks and by expanding each block to 6 bits. This happens in the E-box (Expansion Box).

→ E-box is a special type of permutation. The first block consists of the bits (1,2,3,4), the second one of (5,6,7,8), etc. The expansion to six bits can be seen following figure

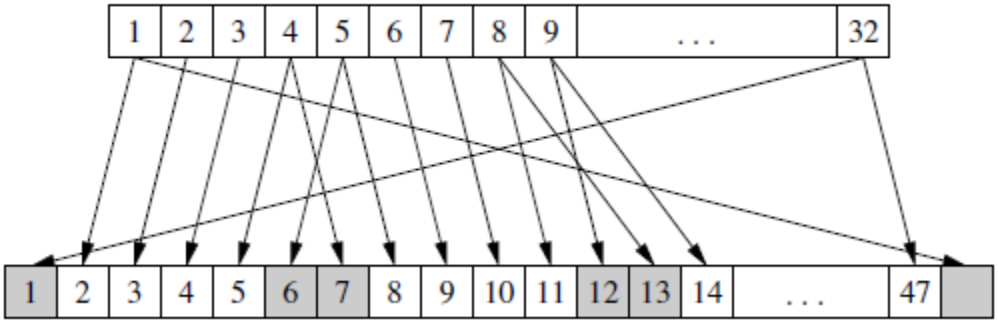


Figure: Examples for the bit swaps of the expansion function E

→ *Main purpose of E- Box: increases diffusion*

→ As can be seen from the following table , exactly 16 of the 32 input bits appear twice in the output

Table: Expansion permutation E

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

→ Next, the 48-bit result of the expansion is XORed with the round key k_i, and the eight 6-bit blocks are fed into eight different **substitution boxes**, which are often referred to as **S-boxes.**

→ Each ***S-box is a lookup table that maps a 6-bit input to a 4-bit output.***

→ *S-Box is the "heart" of DES!*

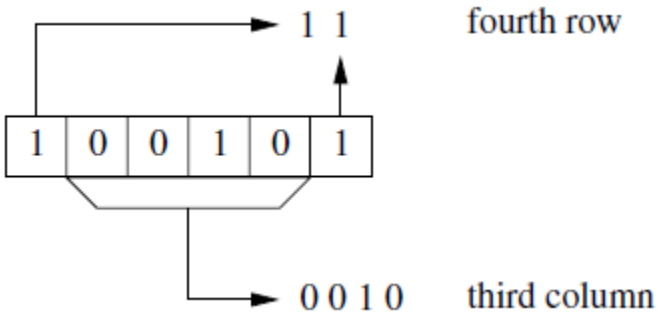
→ *Main Purpose of S-Box: provides confusion*

→ Each S-box contains 2^6 =64 entries, which are typically represented by a table with 16 columns and 4 rows. Each entry is a 4-bit value.
(Note: All S-boxes are listed in tables below)

→ The tables are to be read as follows :

- The most significant bit (MSB) and the least significant bit (LSB) of ***each 6-bit input select the row of the table***, while the ***four inner bits select the column***.
- The integers 0,1,. . . ,15 of each entry in the table represent the decimal notation of a 4-bit value.

Example of the decoding of the input 100101_2 by S-box 1



⇒ The S-box input b = $(100101)_2$ indicates the row $11_2 = 3$ (i.e., fourth row, numbering starts with 00_2) and the column $0010_2 = 2$ (i.e., the third column).

⇒ If the input b is fed into S-box 1, the output is

$S_1(37 = 100101_2) = 8 = 1000_2$.

S-box S_1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-box S_2

S_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

S-box S_3

S_3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

S-box S_4

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

S-box S_5

S_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

S-box S_6

S_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

S-box S_7

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

S-box S_8

S_8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

⇒ ***The S-boxes are the core of DES in terms of cryptographic strength. They are the only nonlinear element in the algorithm and provide confusion.***

(Even though the entire specification of DES was released by NBS/NIST in 1977, the motivation for the choice of the S-box tables was never completely revealed.)

⇒ **The S-boxes are the most crucial elements of DES because they introduce a nonlinearity to the cipher, i.e.,**

$$S(a) \oplus S(b) \neq S(a \oplus b).$$

Assignment: List the criteria which were used for designing of S-Boxes in DES.

(You can refer the book "Understanding Cryptography" by Christof Paar and Jan Pelzl)

→ Finally, the 32-bit output is permuted bitwise according to the P permutation, which is given in following table

Table: The permutation P within the f -function

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

.

- ✓ Unlike the initial permutation IP and its inverse IP−1, the permutation P introduces *diffusion* because ***the four output bits of each S-box are permuted in such a way that they affect several different S-boxes in the following round***.
- ✓ The diffusion caused by the expansion, S-boxes and the permutation P guarantees that every bit at the end of the fifth round is a function of every plaintext bit and every key bit. This behavior is known as the **avalanche effect**

Assignment: What do you mean by avalanche effect? How it is achieved in DES? Explain

(You can refer the books ,"Understanding Cryptography "by Cristof Paar and "Cryptography and Network Security " by William Stallings)

The Key Schedule

- The key schedule derives *16 round keys* k_i, each consisting of 48 bits, from the original 56-bit key. Another term for round key is *subkey*

  (Note: The DES input key is often stated as 64-bit, where every eighth bit is used as an odd parity bit over the preceding seven bits. It is not quite clear why DES was specified that way. In any case, the eight parity bits are not actual key bits and do not increase the security. DES is a 56-bit cipher, not a 64-bit one)

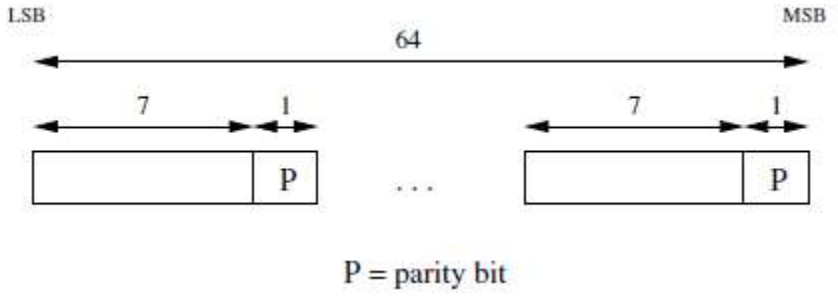


Figure: Location of the eight parity bits for a 64-bit input key

- The 64-bit key is first reduced to 56 bits by ignoring every eighth bit, i.e., the parity bits are stripped in the initial PC−1 permutation. Again the parity bits certainly do not increase the key space! The name PC−1 stands for "permuted choice one". The exact bit connections that are realized by PC−1 are given in following table.

Table : Initial key permutation PC−1

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

- The resulting 56-bit key is split into two halves C_0 and D_0, and the actual key schedule starts as shown in following figure.

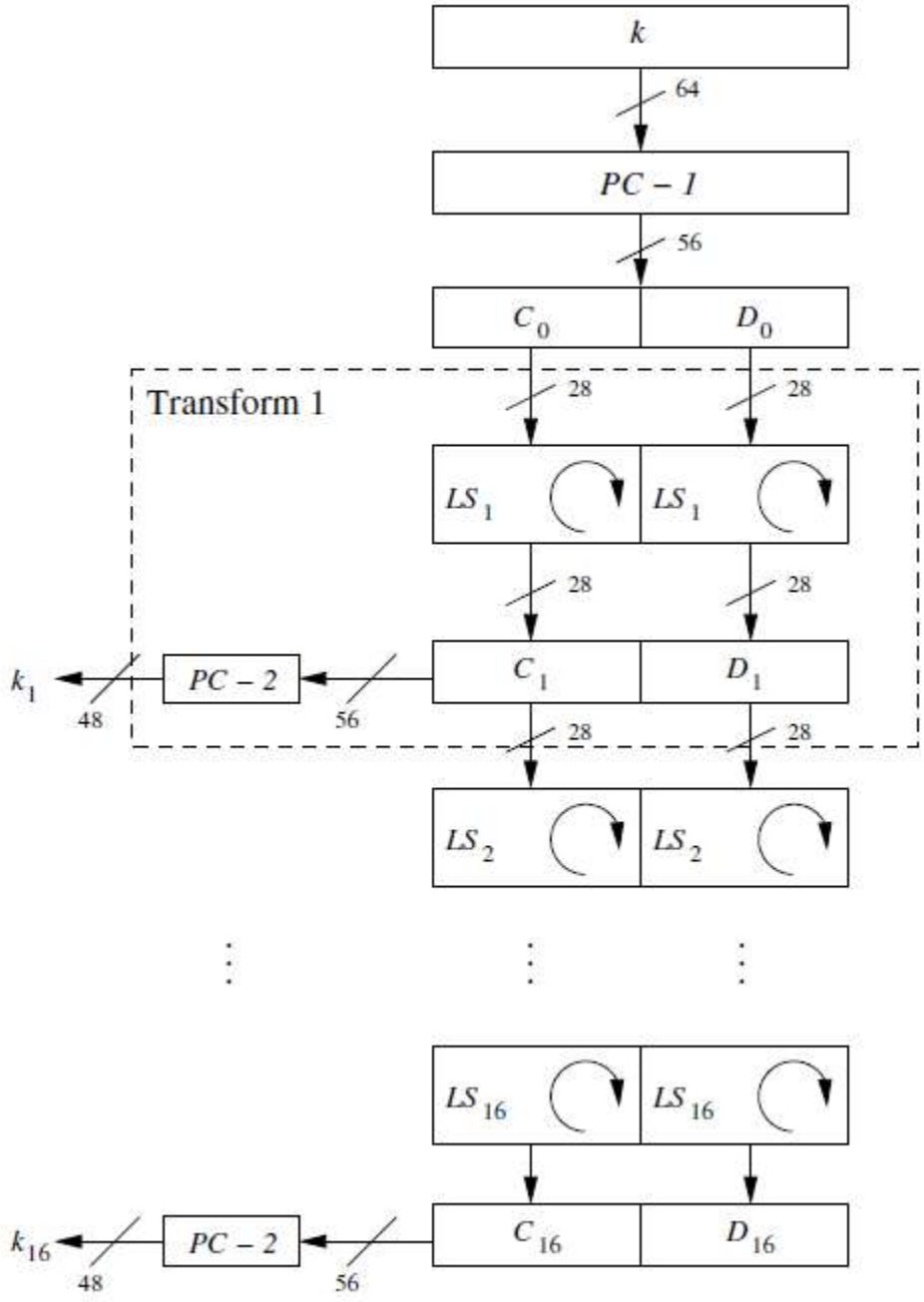


Figure: Key schedule for DES encryption

- The two 28-bit halves are cyclically shifted, i.e., rotated, left by one or two bit positions depending on the round i according to the following rules:
  - In rounds i = 1,2,9, 16, the two halves are rotated left by one bit.
  - In the other rounds where i ≠ 1,2,9, 16, the two halves are rotated left by two bits.
- The rotations only take place within either the left or the right half.
- The total number of rotation positions is 4 × 1+12 × 2 = 28. This leads to the interesting property that $C_0 = C_{16}$ and $D_0 = D_{16}$
- To derive the 48-bit round keys k_i, the two halves are permuted bitwise again with PC−2, which stands for "permuted choice 2". PC−2 permutes the 56 input bits coming from C_i and D_i and ignores 8 of them. The exact bit-connections of PC−2 are given in following table.

Table : Round key permutation PC−2

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

- Every round key is a selection of 48 permuted bits of the input key k. The key schedule is merely a method of realizing the 16 permutations systematically.

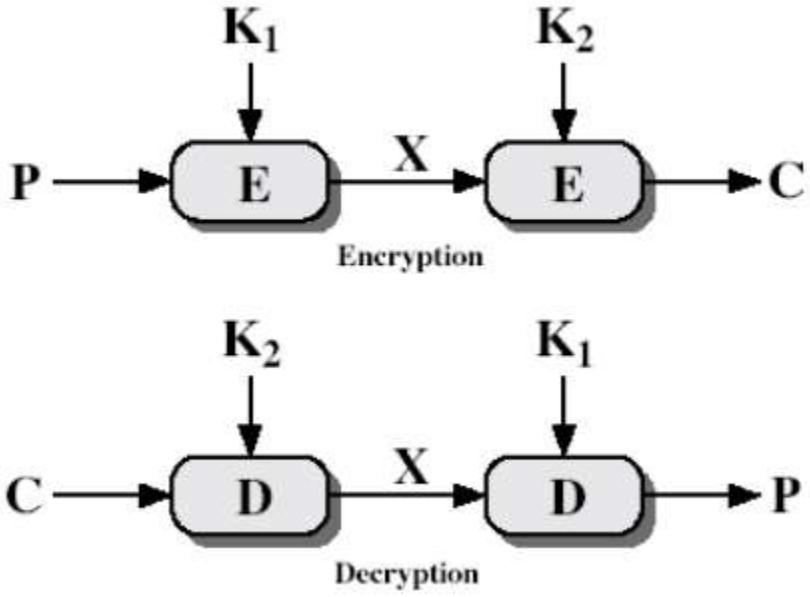
Double DES

- DES uses a 56-bit key, this raised concerns about brute force attacks.
- One proposed solution: double DES (2DES)
- In double DES, DES is applied twice using two keys, K_1 and K_2.

  Encryption: $C = E(K_2, (E(K_1,P))$

  Decryption: $P = D(K_1, (D(K_2, C))$

  Here E and D are DES ciphers for Encryption and Decryption respectively

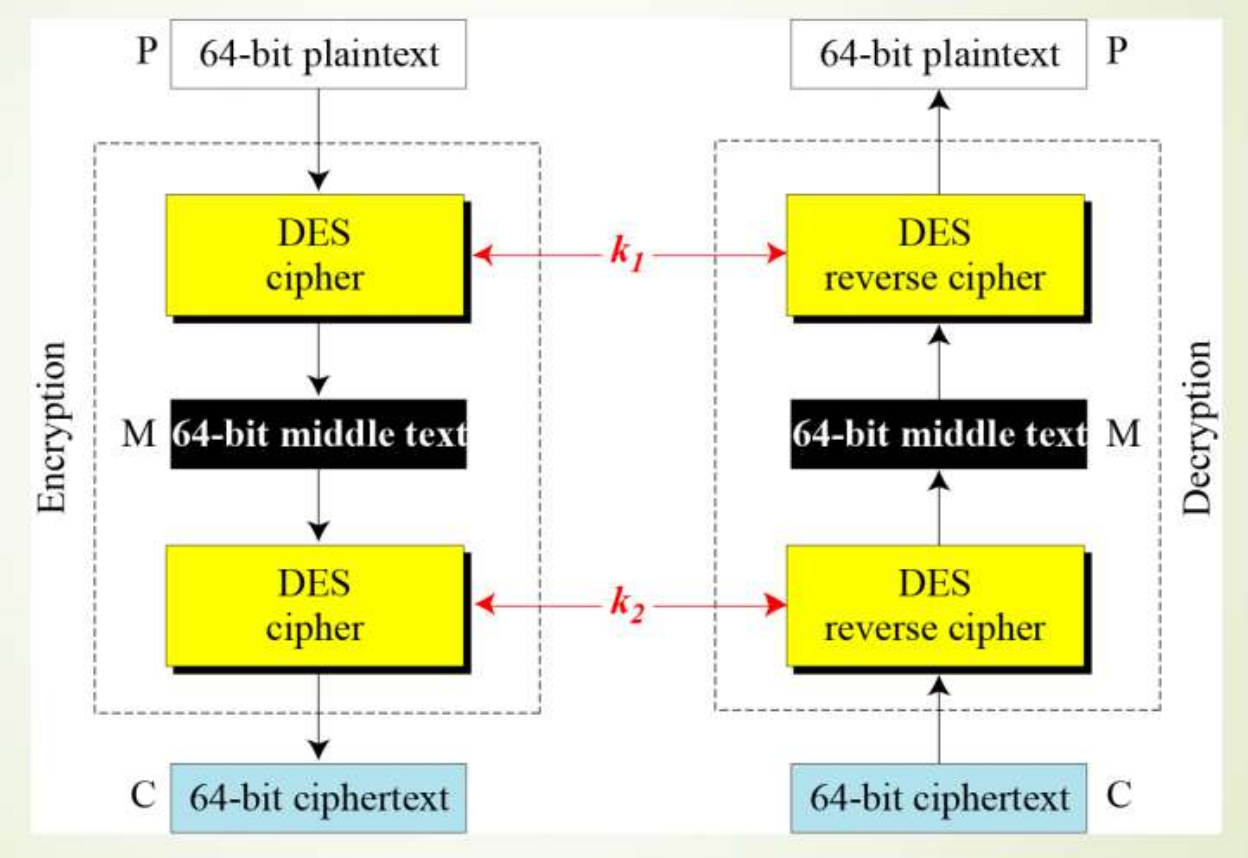


.

- This leads to a 2×56=112 bit key in double DES
- Security does increase by double encryption, but it does not increase much.

Meet-in-the–middle attack on Double DES

This attack requires knowing some plaintext/ciphertext pairs. Let's assume that we have a plaintext/ciphertext pair; i.e., we know the plaintext P and the corresponding (double DES enciphered) ciphertext C. Attacks on DES have typically been brute force attacks.



— The middle text, the text created by the first encryption or the first decryption, M, should be same

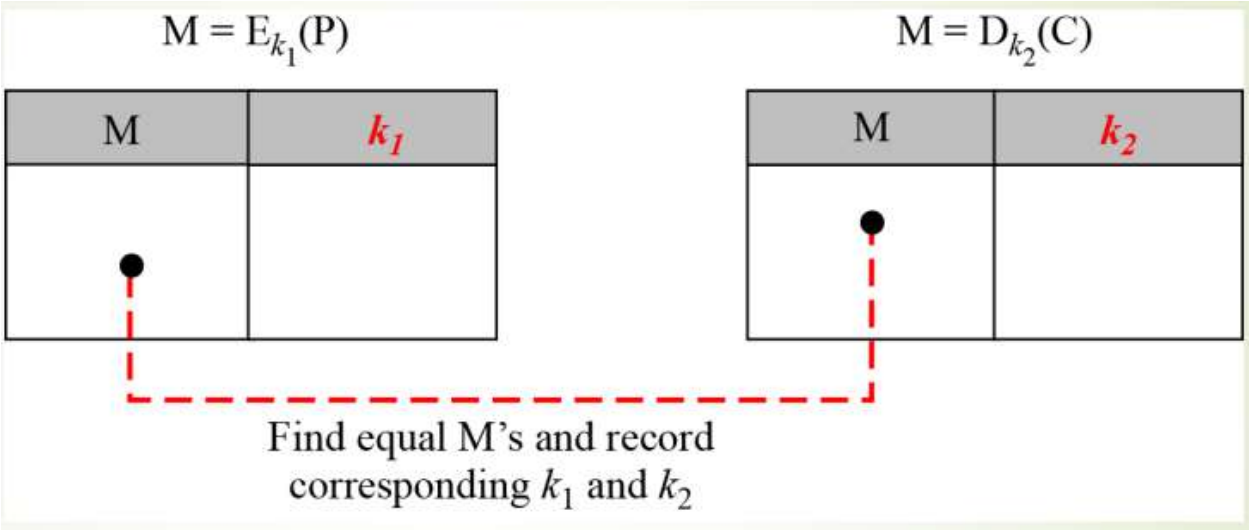
$$M = (K_1,P) \qquad\qquad M = D(K_2,C)$$

— Encrypt P using all possible values of K_1 and record all values obtained for M.

— Decrypt C using all possible values of K_2 and records all values obtained for M.

— Create two tables sorted by M values.

— Now compares the values for M until we finds those pairs of K_1 & K_2 for which the value of M is same in both tables.



— Instead of using 2 ^{112} key search tests, we have to use 2 ^{56} key search tests two times.

— Moving from a Single DES to Double DES, the strength is increased from 2^{56} to 2^{57} attempts.

Triple DES

— Also known as TDES / 3DES

— Uses DES three times for encryption and decryption with different keys.

— 3DES uses 3 or 2 keys

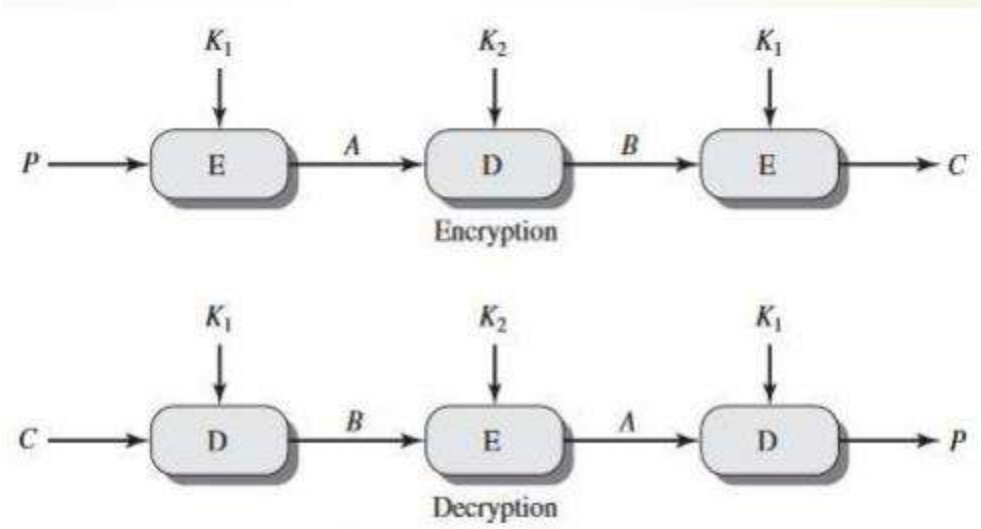
Triple DES with 2-key

— Use three stages of DES for encryption and decryption.

— The 1 st , 3 rd stage use K_1 key and 2 nd stage use K_2 key.

— To make triple DES compatible with single DES, the middle stage uses decryption in the encryption side and encryption in the decryption side.

— It's much stronger than double DES.



— The function follows an encrypt-decrypt-encrypt (EDE) sequence

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

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

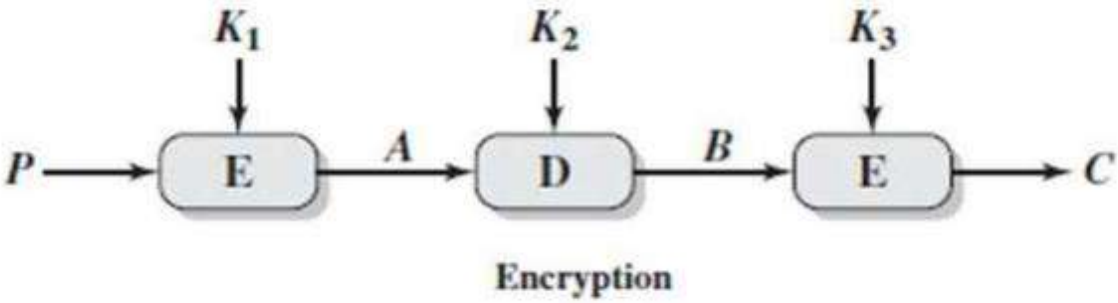
— By the use of triple DES with 2-key encryption, it raises the cost of meet-in-the-middle attack to 2^{112} attempts.

Triple DES with 3-key

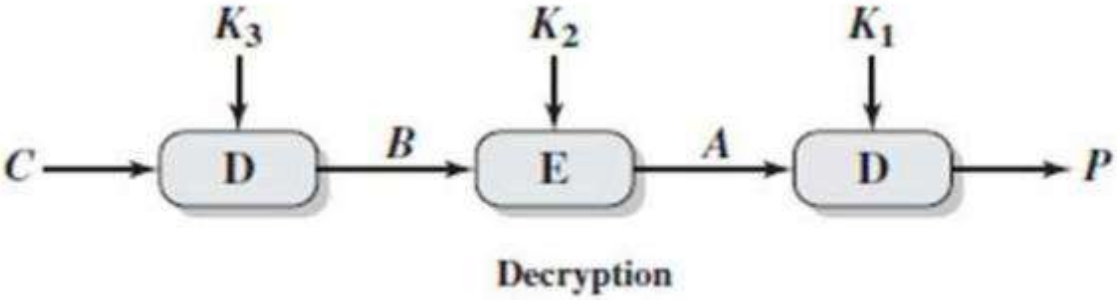
— Uses three stages of DES for encryption and decryption with three different keys.

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

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



Encryption



Decryption

Basic Concepts in Number Theory and Finite Fields

Divisibility

→ We say that a nonzero **b** divides **a** if **a = mb** for some **m**, where **a, b,** and **m** are integers. That is, **b divides a** if there is **no remainder on division**.

→ The notation **b|a** is commonly used to mean **b divides a**. Also, if **b|a**, we say that **b is a divisor of a**.

The positive divisors of 24 are 1, 2, 3, 4, 6, 8, 12, and 24.
$13|182; -5|30; 17|289; -3|33; 17|0$

Simple properties of divisibility for integers

- If $a|1$, then $a = \pm 1$.
- If $a|b$ and $b|a$, then $a = \pm b$.
- Any $b \neq 0$ divides 0.
- If $a|b$ and $b|c$, then $a|c$:

$$11|66 \text{ and } 66|198 = 11|198$$

- If $b|g$ and $b|h$, then $b|(mg + nh)$ for arbitrary integers m and n.

To see this last point, note that

- If $b|g$, then g is of the form $g = b \times g_1$ for some integer g_1.
- If $b|h$, then h is of the form $h = b \times h_1$ for some integer h_1.

So

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

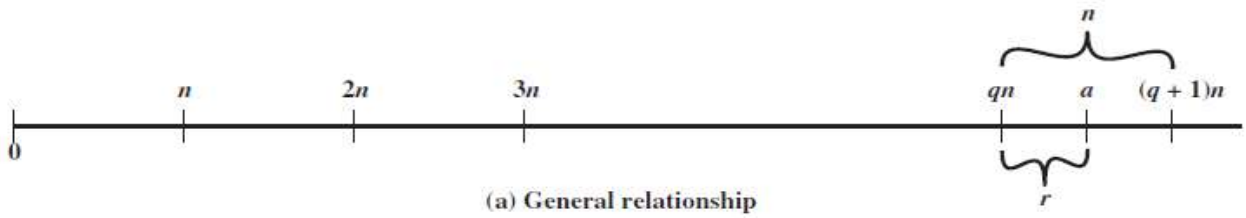
and therefore b divides $mg + nh$.

$b = 7; g = 14; h = 63; m = 3; n = 2$
$7 | 14$ and $7 | 63$.
To show $7 | (3 \times 14 + 2 \times 63)$,
we have $(3 \times 14 + 2 \times 63) = 7(3 \times 2 + 2 \times 9)$,
and it is obvious that $7 | (7(3 \times 2 + 2 \times 9))$.

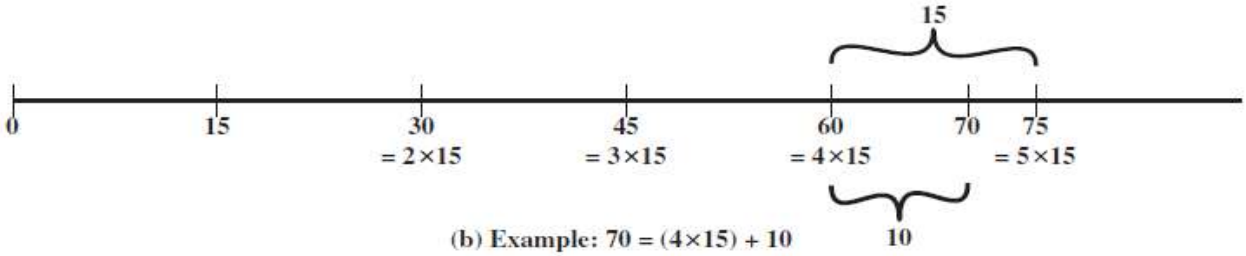
The Division Algorithm

→ 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:

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



(a) General relationship



(b) Example: 70 = (4×15) + 10

The Euclidean Algorithm

→ Euclidean algorithm is a simple procedure for determining the greatest common divisor of two positive integers.

→ ***Relatively prime integers :***

- Two integers are **relatively prime** if their only common positive integer factor is 1
- Two integers are relatively prime (or coprime) if there is no integer greater than one that divides them both.

[no common factors other than 1]

Example:

21 and 22 are relatively prime

The factors of 21 are 1, 3, 7 and 21
The factors of 22 are 1, 2, 11 and 22
(the only common factor is 1)

But 21 and 24 are NOT relatively prime

The factors of 21 are 1, 3, 7 and 21
The factors of 24 are 1, 2, 3, 4, 6, 8, 12 and 24
(the common factors are 1 and 3)

→ Relatively Prime is also called "coprime" or "mutually prime"

Greatest Common Divisor

⇒ Greatest Common Divisor *(GCD)* of two positive numbers *a* and *b* is the largest integer that divides both *a* and *b*. It can be denoted as *gcd(a,b)*

⇒ ***gcd(0, 0) = 0***.

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

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

An equivalent definition is the following:

gcd(a, b) = max[k, such that k|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 |).**

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

⇒ ***gcd(a, 0) = | a |***

⇒ ***a*** and ***b*** are **relatively prime** if ***gcd(a, b) = 1***.

Euclidian Algorithm to find GCD

The Euclidean algorithm can be based on the following theorem:

For any integers a, b, with $a \geq b \geq 0$, gcd(a, b) = gcd(b, a mod b)

Core idea:

— *We can reduce the problem of finding the gcd of two given numbers to that of the gcd of two smaller numbers.*

— *Finding the gcd of two integers a and b can be done by recursively reducing the operands*

Euclidean Algorithm	
Calculate	**Which satisfies**
$r_1 = a \bmod b$	$a = q_1 b + r_1$
$r_2 = b \bmod r_1$	$b = q_2 r_1 + r_2$
$r_3 = r_1 \bmod r_2$	$r_1 = q_3 r_2 + r_3$
• • •	• • •
$r_n = r_{n-2} \bmod r_{n-1}$	$r_{n-2} = q_n r_{n-1} + r_n$
$r_{n+1} = r_{n-1} \bmod r_n = 0$	$r_{n-1} = q_{n+1} r_n + 0$ $d = \gcd(a, b) = r_n$

Algorithm

Input: Two positive integers, a and b.

Output: The greatest common divisor, g, of a and b.

Internal computation

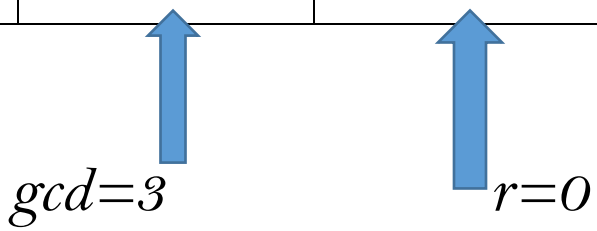
- If a<b, exchange a and b.
- Divide a by b and get the remainder, r.
- If r=0, print b as the GCD of a and b.
- Otherwise replace a by b and replace b by r then go to previous step.

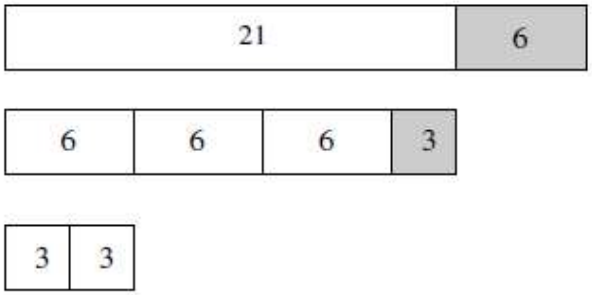
We can define the Euclidean algorithm concisely as the following recursive function.

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

Example : To compute gcd(27,21)

a	b	r=a%b
27	21	6
21	6	3
6	3	0





gcd(27, 21) = gcd(1 · 21+6, 21) = gcd(21, 6)

gcd(21, 6) = gcd(3 · 6+3, 6) = gcd(6, 3)

gcd(6, 3) = gcd(2 · 3+0, 3) = gcd(3, 0) = 3

gcd(27, 21) = gcd(21, 6) = gcd(6, 3) = gcd(3, 0) = 3

Try yourself

gcd(913,301) =?

gcd(1220,516) =?

Modular Arithmetic

- ✓ Almost all crypto algorithms, both symmetric ciphers and asymmetric ciphers, are based on arithmetic within a finite number of elements.
- ✓ Most number sets we are used to, such as the set of natural numbers or the set of real numbers, are infinite.
- ✓ Modular arithmetic is a simple way of performing arithmetic in a finite set of integers

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.**

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

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

- ✓ Two integers **a** and **b** are said to be **congruent modulo n**, if

  (a mod n) = (b mod n).

  This is written as,

  a ≡ b (mod n)

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

- ✓ If **a≡ 0 (mod n),** then **n|a**. (i.e. n divides 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) = kn for some k.
```

So we can write

```
a = b + kn
```

Therefore,

```
(a mod n) = (remainder when b + kn is divided by n) =
(remainder when b is divided by n) = (b mod n).
```

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

Modular Arithmetic Operations

- The **(mod n)** operator maps all integers into the set of integers {0, 1, . . . , (n - 1)}
- We can perform arithmetic operations within the confines of this set. 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. $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

Examples

$11 \bmod 8 = 3; 15 \bmod 8 = 7$
$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$
$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$
$[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = -4 \bmod 8 = 4$
$(11 - 15) \bmod 8 = -4 \bmod 8 = 4$
$[(11 \bmod 8) \times (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$
$(11 \times 15) \bmod 8 = 165 \bmod 8 = 5$

- The rules for ordinary arithmetic involving addition, subtraction, and multiplication carry over into modular arithmetic.

Addition modulo 8

+	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

- As in ordinary addition, there is an **additive inverse**, **or negative**, to each integer in modular arithmetic.
  - In this case, the negative (additive inverse) of an integer **x** is the integer **y** such that **(x + y) mod 8 = 0.**
  - Since (2 + 6) mod 8 = 0; 6 is the additive inverse of 2 in modulo 8.

Multiplication modulo 8

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

- As in ordinary arithmetic, there is a **multiplicative inverse, or reciprocal**, to each integer in modular arithmetic.
  - In modular arithmetic mod 8, the multiplicative inverse of **x** is the integer **y** such that

    (x × y) mod 8 = 1.
  - Not all integers mod 8 have a multiplicative inverse.
  - In general, **an integer has a multiplicative inverse** in $\mathbf{Z_n}$ if that integer is **relatively prime** to **n**

Additive and multiplicative inverse modulo 8

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

Residue classes

Define the set Z_n 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 Z_n represents a **residue class**.

We can label the residue classes (mod n) as [0], [1], [2], . . ., [n - 1], where

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

The residue classes (mod 4) are

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

- Of all the integers in a residue class, **the smallest nonnegative integer** is the one used to represent the residue class.
- Finding the smallest nonnegative integer to which k is congruent modulo n is called **reducing k modulo n**.

Properties of Modular Arithmetic

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 $w + z \equiv 0 \bmod n$

Extended Euclidean Algorithm

- An extension to the Euclidean algorithm that will be important for computations in the area of finite fields and in encryption algorithms, such as RSA.
- For given integers **a** and **b**, the extended Euclidean algorithm not only calculate the greatest common divisor **d** but also two additional integers **x** and **y** that satisfy the following equation.

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

- This equation is often referred to as Diophantine equation.

Extended Euclidean Algorithm			
Calculate	**Which satisfies**	**Calculate**	**Which satisfies**
$r_{-1} = a$		$x_{-1} = 1; y_{-1} = 0$	$a = ax_{-1} + by_{-1}$
$r_0 = b$		$x_0 = 0; y_0 = 1$	$b = ax_0 + by_0$
$r_1 = a \bmod b$ $q_1 = \lfloor a/b \rfloor$	$a = q_1 b + r_1$	$x_1 = x_{-1} - q_1 x_0 = 1$ $y_1 = y_{-1} - q_1 y_0 = -q_1$	$r_1 = ax_1 + by_1$
$r_2 = b \bmod r_1$ $q_2 = \lfloor b/r_1 \rfloor$	$b = q_2 r_1 + r_2$	$x_2 = x_0 - q_2 x_1$ $y_2 = y_0 - q_2 y_1$	$r_2 = ax_2 + by_2$
$r_3 = r_1 \bmod r_2$ $q_3 = \lfloor r_1/r_2 \rfloor$	$r_1 = q_3 r_2 + r_3$	$x_3 = x_1 - q_3 x_2$ $y_3 = y_1 - q_3 y_2$	$r_3 = ax_3 + by_3$
• • •	• • •	• • •	• • •
$r_n = r_{n-2} \bmod r_{n-1}$ $q_n = \lfloor r_{n-2}/r_{n-1} \rfloor$	$r_{n-2} = q_n r_{n-1} + r_n$	$x_n = x_{n-2} - q_n x_{n-1}$ $y_n = y_{n-2} - q_n y_{n-1}$	$r_n = ax_n + by_n$
$r_{n+1} = r_{n-1} \bmod r_n = 0$ $q_{n+1} = \lfloor r_{n-1}/r_n \rfloor$	$r_{n-1} = q_{n+1} r_n + 0$		$d = \gcd(a, b) = r_n$ $x = x_n; y = y_n$

Algorithm (EEA)

Input: positive integers **a** and **b** with **a> b**

Output: d=gcd(a, b), as well as x and y such that gcd(a, b) =a· x+b·y

Initialization:

$r_{-1} = a$

$r_0 = b$

$x_{-1} = 1$

$x_0 = 0$

$y_{-1} = 0$

$y_0 = 1$

$i = 0$

Compute:

DO

$i = i + 1$

$r_i = r_{i-2} \bmod r_{i-1}$

$q_i = \text{floor}(r_{i-2} / r_{i-1})$

$x_i = x_{i-2} - q_i \cdot x_{i-1}$

$y_i = y_{i-2} - q_i \cdot y_{i-1}$

WHILE ($r_i \neq 0$)

$d = \gcd(a, b) = r_{i-1}$

$x = x_{i-1}$

$y = y_{i-1}$

RETURN d,x,y

Example:

Let us use **a** = 1759 and **b** = 550 and solve for *1759x + 550y = gcd(1759, 550).*

i	r_i	q_i	x_i	yi
−1	1759		1	0
0	550		0	1
1	109	3	1	−3
2	5	5	−5	16
3	4	21	106	−339
4	1	1	−111	355
5	0	4		

Result: $d = 1; x = -111; y = 355$

Try yourself

- Determine integers x and y such that gcd(421, 111) = 421x + 111y
  (Answer: d=1, x = -29 and y = 110)
- Determine a solution to the linear Diophantine equation gcd(93, 219) = 219x + 93y.
  (Answer: d=3, x = -14 and y = 33)
- Find the greatest common divisor d of 540 and 192, and find integers x and y solving the equation 540x + 192y = d.
  (Answer. d = 12, x = 5 and y = -14)

The Extended Euclidean Algorithm for finding the inverse of a number mod n

- Finding the gcd is not the main application of the Euclidean algorithm. An extension of the algorithm allows us to compute modular inverses, which is of major importance in public-key cryptography.
- The main application of the EEA in asymmetric cryptography is to compute the inverse modulo of an integer.
- Inverse of an integer **a** in modulo **n** exists **if gcd(n, a)=1.**
- The extended Euclidean algorithm can be used to find a multiplicative inverse in Z_n for any **n**.
    - If we apply the extended Euclidean algorithm to the equation ***nx + by = d***, and the algorithm yields d = 1, then $\mathbf{y = b^{-1}}$ in Z_n. (b is a number and b^{-1} is inverse of b in Z_n)

Example: Compute 12^{-1} mod 67. (Find multiplicative inverse of 12 in modulo 67 or Z_{67})

Here, gcd(67,12) = 1 so, inverse of 12 exists in modulo 67.

Using EEA to compute 12^{-1} mod 67

67x + 12y = gcd(67,12)

i	$\mathbf{r_i}$	$\mathbf{q_i}$	$\mathbf{x_i}$	$\mathbf{y_i}$
-1	67		1	0
0	12		0	1
1	7	5	1	-5
2	5	1	-1	6
3	2	1	2	-11
4	1	2	-5	**28**
5	0	2		

∴ x = -5 , y = 28

Here, y is the inverse of 12 in modulo 67

Hence , 12^{-1} mod 67 = 28

Try yourself

1. Find the inverse of 111 modulo 421

   (Answer: 110)
2. Find the inverse of 15 mod 26

   (Answer: 7)
3. 25^{-1} mod 128 = ?

   (Answer: 41)
4. 21^{-1} mod 26 = ?

   (Answer: 5)

Groups, Rings, and Fields

- Groups, rings, and fields are the fundamental elements of a branch of mathematics known as **abstract algebra, or modern algebra**.
- In abstract algebra, we are concerned with sets on whose elements we can operate algebraically; that is, we can combine two elements of the set, perhaps in several ways, to obtain a third element of the set.
- These operations are subject to specific rules, which define the nature of the set.
- By convention, the notation for the two principal classes of operations on set elements is usually the same as the notation for addition and multiplication on ordinary numbers.
- However, it is important to note that, in abstract algebra, we are not limited to ordinary arithmetical operations

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:

(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**.

— *Note: The operator • is generic and can refer to addition, multiplication, or some other mathematical operation.*

Example: the integers

— One of the most familiar groups is the set of integers **Z** which consists of the numbers {..., −4, −3, −2, −1, 0, 1, 2, 3, 4, ...} together with addition.

— The following properties of integer addition serve as a model for the group axioms given in the definition.

- For any two integers **a** and **b**, the sum **a + b** is also an integer. That is, addition of integers always yields an integer. This property is known as **closure under addition**. (satisfies A1)
- For all integers **a, b** and **c**, **(a + b) + c** = **a + (b + c)**. Expressed in words, adding **a** to **b** first, and then adding the result to **c** gives the same final result as adding **a** to the sum of **b** and **c**, a property known as **associativity**.(satisfies A2)
- If **a** is any integer, then **0 + a = a + 0 = a**. Zero is called the **identity element of addition** because adding it to any integer returns the same integer.(satisfies A3)
- For every integer **a**, there is an integer **b** such that **a + b = b + a = 0**. The integer **b** is called the **inverse element** of the integer **a** and is denoted **-a**. (satisfies A4)

Another Example of a Group

Let N_n denote a set of n distinct symbols that, for convenience, we represent as $\{1, 2, \ldots, n\}$. A **permutation** of n distinct symbols is a one-to-one mapping from N_n to N_n. Define S_n to be the set of all permutations of n distinct symbols. Each element of S_n is represented by a permutation of the integers π in $1, 2, \ldots, n$. It is easy to demonstrate that S_n is a group:

A1: If $(\pi, \rho \in S_n)$, then the composite mapping $\pi \cdot \rho$ is formed by permuting the elements of ρ according to the permutation π. For example, $\{3, 2, 1\} \cdot \{1, 3, 2\} = \{2, 3, 1\}$. Clearly, $\pi \cdot \rho \in S_n$.

A2: The composition of mappings is also easily seen to be associative.

A3: The identity mapping is the permutation that does not alter the order of the n elements. For S_n, the identity element is $\{1, 2, \ldots, n\}$.

A4: For any $\pi \in S_n$, the mapping that undoes the permutation defined by π is the inverse element for π. There will always be such an inverse. For example $\{2, 3, 1\} \cdot \{3, 1, 2\} = \{1, 2, 3\}$.

Finite and Infinite 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**.

Abelian Group

- A group is said to be **abelian** if it satisfies the following additional condition:

  (A5) Commutative: a ●b = b ● a for all **a, b** in **G**.

- Also called a commutative group

The set of integers (positive, negative, and 0) under addition is an abelian group. The set of nonzero real numbers under multiplication is an abelian group. The set S_n from the preceding example is a group but not an abelian group for $n > 2$.

Cyclic Group

- We define exponentiation within a group as a repeated application of the group operator, so that $\mathbf{a^3 = a \bullet a \bullet a}$. Furthermore, we define $\mathbf{a^0 = e}$ as the identity element, and $\mathbf{a^{-n} = (\ a^{'}\ )^n}$, where $\mathbf{a^{'}}$ is the inverse element of **a** within the group.
- **Cyclic Group** is a group generated by a single element, and that element is called generator of that cyclic group.
- A group **G** is cyclic if every element of **G** is a power $\mathbf{a^k}$ (k is an integer) of a fixed element $\mathbf{a \in G}$. The element **a** is said to **generate** the group **G** or to be a **generator** of **G**.

  i.e. A cyclic group **G** is one in which every element is a power of a particular element **a**, in the group
- A cyclic group is always abelian and may be finite or infinite

> The additive group of integers is an infinite cyclic group generated by the element 1. In this case, powers are interpreted additively, so that *n* is the *n*th power of 1.

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**.

- In essence, a ring is a set in which we can do addition, subtraction [a - b = a + (-b)], and multiplication without leaving the set.

> With respect to addition and multiplication, the set of all *n*-square matrices over the real numbers is a ring.

Commutative Ring

- 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.**

Integral Domain

- An **integral domain** 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**.

> Let *S* be the set of integers, positive, negative, and 0, under the usual operations of addition and multiplication. *S* is an integral domain.

What is zero divisor??

- A nonzero element of a ring such that its product with some other nonzero element of the ring equals zero is called zero divisor.
- In a ring **R,** a nonzero element **a** $\in R$ is said to be a **zero divisor** if there exists a nonzero **b** $\in R$ such that **ab=0**.

  Ex: in Z_4 , $2\times2 = 0$

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 $\mathbf{a^{-1}}$ in **F** such that $\mathbf{aa^{-1} = (a^{-1})a = 1}$.

- 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:

  $\mathbf{a/b = a(b^{-1})}$ **.**

Familiar examples of fields are the rational numbers, the real numbers, and the complex numbers. Note that the set of all integers is not a field, because not every element of the set has a multiplicative inverse; in fact, only the elements 1 and −1 have multiplicative inverses in the integers.

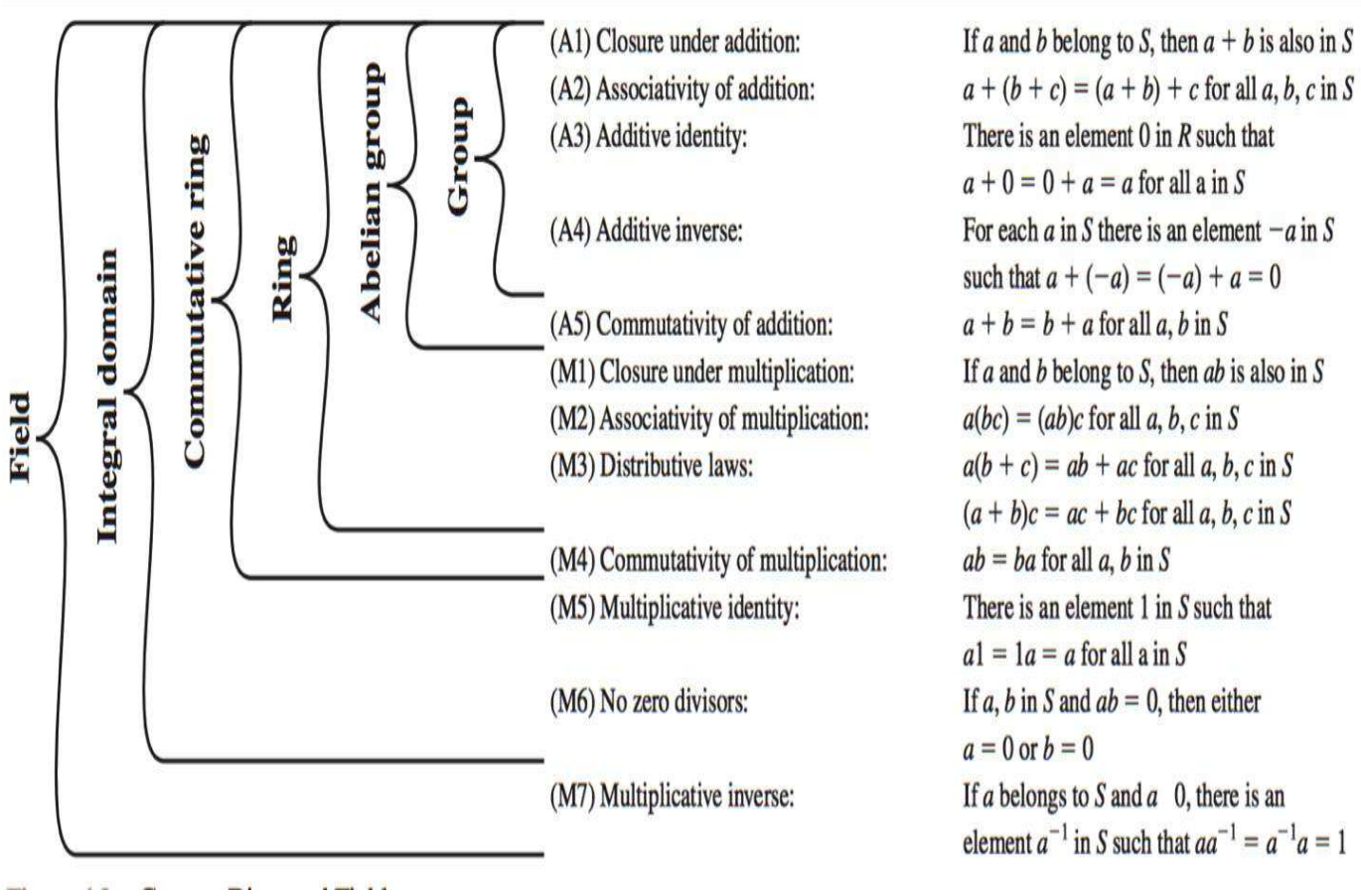


Figure: Group, Ring, and Field

Finite Fields of The Form GF(*p*)

→ Infinite fields are not of particular interest in the context of cryptography.

→ However, finite fields play a crucial role in many cryptographic algorithms.

→ It can be shown that the order of a finite field (number of elements in the field) must be a power of a prime $\mathbf{p^n}$, where **n** is a positive integer

> A field with order **m** only exists if **m** is a prime power, i.e., $\mathbf{m = p^n}$, for some positive integer **n** and prime integer **p**. **p** is called the characteristic of the finite field.

→ Prime number is an integer whose only positive integer factors are itself and 1.
i.e. the only positive integers that are divisors of p are p and 1.

→ The finite field of order $\mathbf{p^n}$ is generally written $\mathbf{GF(p^n)}$**; GF** stands for Galois field, in honor of the French mathematician (Évariste Galois)who first studied finite fields.

→ Two special cases are of interest for our purposes.

- Finite field **GF(p)** when n = 1
- Finite fields with **n >1**

Finite Fields of Order p

⇒ For a given prime, **p**, we define the finite field of order p, **GF(p),** as the set **Zp** of integers {0, 1, . . . , p - 1} together with the arithmetic operations **modulo 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.

⇒ The Arithmetic in GF(7) is given below. This is a field of order 7 using modular arithmetic modulo 7. As can be seen, it satisfies all of the properties required of a field

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

(a) Addition modulo 7

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

(b) Multiplication modulo 7

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

(c) Additive and multiplicative inverses modulo 7

- In fine field of order p [GF(p)] (where p is prime number) :
  - ✓ GF(p) consists of p elements.
  - ✓ The binary operations + and × are defined over the set. The operations of addition, subtraction, multiplication, and division can be performed without leaving the set.
  - ✓ Arithmetic in GF(p) is done modulo p.
  - ✓ Each element of the set other than 0 has a multiplicative inverse.

Finding the Multiplicative Inverse in GF(p)

- Same as we discussed earlier in EEA

Assignment - Prove: Zp is a finite field.

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.

Ordinary Polynomial Arithmetic

- A polynomial of degree n (integer n >= 0) is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

  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 *nth-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.**
- Polynomial arithmetic includes the operations of addition, subtraction, and multiplication.
  - These operations are defined in a natural way as though the variable **x** was an element of **S**.
  - Division is similarly defined, but requires that S be a field.
  - Examples of fields include the real numbers, rational numbers, and $\mathbf{Z_p}$ for **p** prime.
  - Note that the set of all integers is not a field and does not support polynomial division.
- Addition and subtraction are performed by adding or subtracting corresponding coefficie nts. Thus, if

$$f(x) = \sum_{i=0}^{n} a_i x^i; \qquad g(x) = \sum_{i=0}^{m} b_i x^i; \qquad n \geq m$$

then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^{m} (a_i + b_i)x^i + \sum_{i=m+1}^{n} a_i x^i$$

and multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{k-1} b_1 + a_k b_0$$

— In the last formula, we treat a_i as zero for i > n and b_i as zero for i > m.

— Note that the degree of the product is equal to the sum of the degrees of the two polynomials

As an example, let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$, where S is the set of integers. Then

$$f(x) + g(x) = x^3 + 2x^2 - x + 3$$
$$f(x) - g(x) = x^3 + x + 1$$
$$f(x) \times g(x) = x^5 + 3x^2 - 2x + 2$$

Following figure show the manual calculations

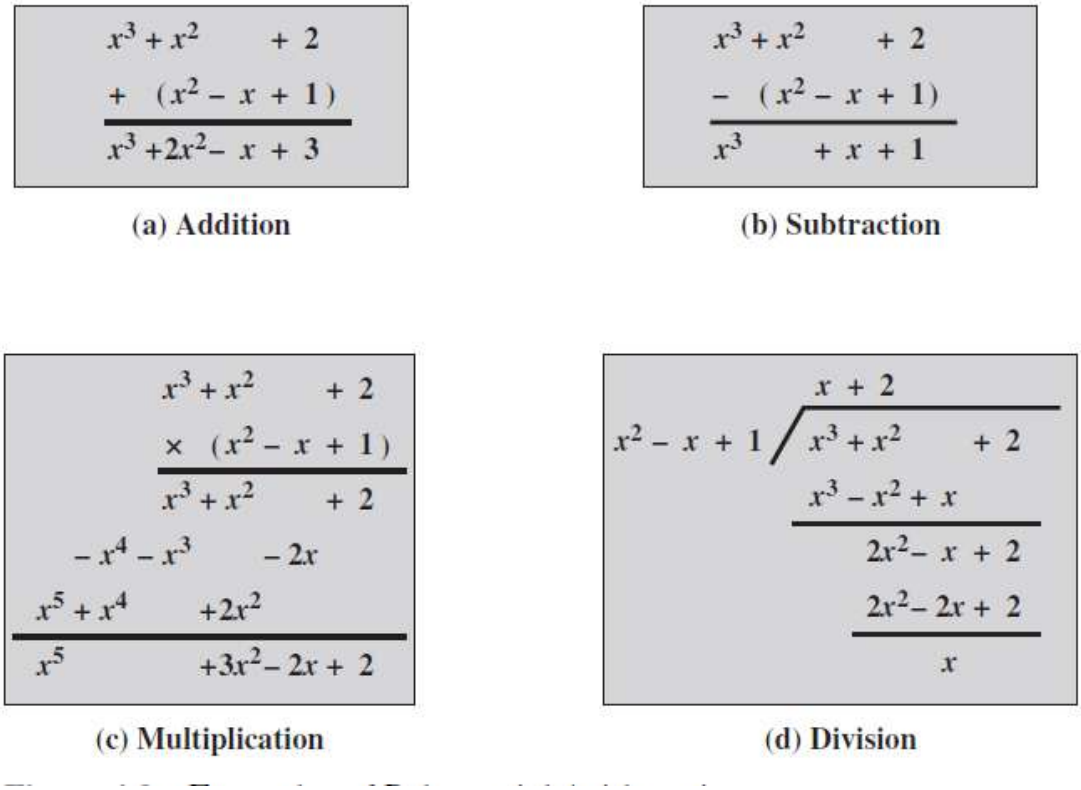


Figure: Examples of polynomial arithmetic

Polynomial Arithmetic with Coefficients in Z_p

⇒ Let us now consider polynomials in which the coefficients are elements of some field F; we refer to this as a polynomial over the field F.

⇒ In that case, it is easy to show that the set of such polynomials is a ring, referred to as a **polynomial ring**.

⇒ That is, if we consider each distinct polynomial to be an element of the set, then that set is a ring

⇒ When polynomial arithmetic is performed on polynomials over a field, then division is possible.

— Note that this does not mean that exact division is possible.

— Let us clarify this distinction. Within a field, given two elements **a** and **b**, the quotient **a/b** is also an element of the field. However, given a ring **R** that is not a field, in general, division will result in both a quotient and a remainder; this is not exact division.

Consider the division 5/3 within a set S. If S is the set of rational numbers, which is a field, then the result is simply expressed as 5/3 and is an element of S. Now suppose that S is the field Z_7. In this case, we calculate

$$5/3 = (5 \times 3^{-1}) \bmod 7 = (5 \times 5) \bmod 7 = 4$$

which is an exact solution. Finally, suppose that S is the set of integers, which is a ring but not a field. Then 5/3 produces a quotient of 1 and a remainder of 2:

$$5/3 = 1 + 2/3$$
$$5 = 1 \times 3 + 2$$

Thus, division is not exact over the set of integers.

— Now, if we attempt to perform polynomial division over a coefficient set that is not a field, we find that division is not always defined.

If the coefficient set is the integers, then $(5x^2)/(3x)$ does not have a solution, because it would require a coefficient with a value of 5/3, which is not in the coefficient set. Suppose that we perform the same polynomial division over Z_7. Then we have $(5x^2)/(3x) = 4x$, which is a valid polynomial over Z_7.

— Even if the coefficient set is a field, polynomial division is not necessarily exact.

— In general, division will produce a quotient and a remainder. We can restate the division algorithm of Equation for polynomials over a field as follows

Given polynomials **f(x)** of degree **n** and **g(x)** of degree **(m)**, (n >= m), if we divide **f(x)** by **g(x),** we get a quotient **q(x)** and a remainder **r(x)** that obey the relationship

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

with polynomial degrees:

Degree f(x) = n

Degree g(x) = m

Degree q(x) = n - m

Degree r(x) <= m – 1

— With the understanding that remainders are allowed, we can say that polynomial division is possible if the coefficient set is a field.

— We can write f(x) mod g(x) for the remainder r(x) in above equation.

- That is, **r(x) = f(x) mod g(x).**
- 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).**

- **For our purposes, polynomials over GF(2) are of most interest.**
  - In **GF(2),** addition is equivalent to the **XOR** operation, and multiplication is equivalent to the logical **AND** operation
  - Further, addition and subtraction are equivalent mod 2

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

Examples of Polynomial Arithmetic over GF(2):

— For $f(x) = (x^7 + x^5 + x^4 + x^3 + x + 1)$ and $g(x) = (x^3 + x + 1)$, the following figure shows

→ f(x) + g(x);

→ f(x) - g(x); f(x) × g(x); and

→ f(x)/g(x).

— Note that g(x) | f(x).

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

(a) Addition

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

(b) Subtraction

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

(c) Multiplication

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

(d) Division

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

$$x^4 + 1 = (x + 1)(x^3 + x^2 + x + 1).$$

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 $f(x)$:

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

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.

Examples of Polynomial Arithmetic over GF(2)

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

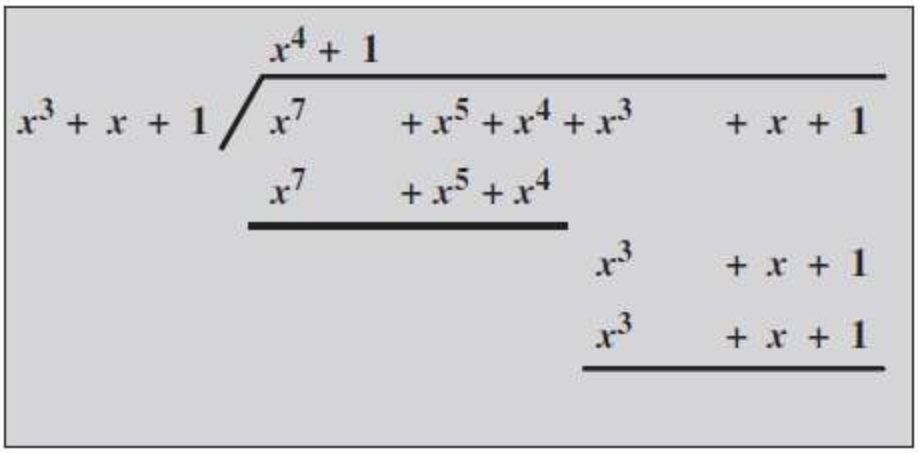
(a) Addition

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

(b) Subtraction

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

(c) Multiplication



(d) Division

Try yourself :

Perfrom following operations in GF(11)

- $(x^5 + 3x^3 + 4)+(6x^6 + 4x^3)$

  Answer: $6x^6 + x^5 + 7x^3 + 4$
- $(x^5 + 3x^3 + 4)-(6x^6 + 4x^3)$

  Answer: $5x^6 + x^5 + 10x^3 + 4$
- $(x^5 + 3x^3 + 4)*(6x^6 + 4x^3)$

  Answer: $6x^{11} + 7x^9 + 4x^8 + 3x^6 + 5x^3$
- $(3x^6 + 7x^4 + 4x^3 + 5) \div (x^4 + 3x^3 + 4)$

  Answer: $3x^2 + 3x + 3$ with remainder $x^3 + 10x^2 + 4x + 1$

Finding the Greatest Common Divisor

- We can extend the analogy between polynomial arithmetic over a field and integer arithmetic by defining the greatest common divisor as follows.

  The polynomial **c(x)** is said to be the greatest common divisor of **a(x)** and **b(x)** if the following are true.

  - **c(x)** divides both **a(x)** and **b(x).**
  - Any divisor of **a(x)** and **b(x)** is a divisor of **c(x).**

- We can find greatest common divisor for polys
  - **c(x) = GCD(a(x), b(x))** if **c(x)** is the poly of greatest degree which divides both **a(x), b(x)**

gcd [a(x), b(x)] is the polynomial of maximum degree that divides both a(x) and b(x).

❖ We can adapt the Euclidean algorithm to compute the greatest common divisor of two polynomials.

$$\gcd[a(x), b(x)] = \gcd[b(x), a(x) \bmod b(x)]$$

Euclidean Algorithm for Polynomials	
Calculate	**Which satisfies**
$r_1(x) = a(x) \bmod b(x)$	$a(x) = q_1(x)b(x) + r_1(x)$
$r_2(x) = b(x) \bmod r_1(x)$	$b(x) = q_2(x)r_1(x) + r_2(x)$
$r_3(x) = r_1(x) \bmod r_2(x)$	$r_1(x) = q_3(x)r_2(x) + r_3(x)$
• • •	• • •
$r_n(x) = r_{n-2}(x) \bmod r_{n-1}(x)$	$r_{n-2}(x) = q_n(x)r_{n-1}(x) + r_n(x)$
$r_{n+1}(x) = r_{n-1}(x) \bmod r_n(x) = 0$	$r_{n-1}(x) = q_{n+1}(x)r_n(x) + 0$ $d(x) = \gcd(a(x), b(x)) = r_n(x)$

At each iteration, we have $d(x) = \gcd(r_{i+1}(x), r_i(x))$ until finally $d(x) = \gcd(r_n(x), 0) = r_n(x)$. Thus, we can find the greatest common divisor of two integers by repetitive application of the division algorithm. This is the Euclidean algorithm for polynomials. The algorithm assumes that the degree of $a(x)$ is greater than the degree of $b(x)$.

The Euclidean algorithm can be extended to find the greatest common divisor of two polynomials whose coefficients are elements of a field!!

Find $\gcd[a(x), b(x)]$ for $a(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ and $b(x) = x^4 + x^2 + x + 1$. First, we divide $a(x)$ by $b(x)$:

$$
\begin{array}{r}
x^2 + x \\
x^4 + x^2 + x + 1 \overline{)\, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1} \\
\underline{x^6 \qquad + x^4 + x^3 + x^2} \\
x^5 \qquad\qquad\quad + x + 1 \\
\underline{x^5 \qquad\quad x^3 + x^2 + x} \\
x^3 + x^2 \qquad + 1
\end{array}
$$

This yields $r_1(x) = x^3 + x^2 + 1$ and $q_1(x) = x^2 + x$.

Then, we divide $b(x)$ by $r_1(x)$.

$$
\begin{array}{r}
x + 1 \\
x^3 + x^2 + 1 \overline{)\, x^4 \qquad + x^2 + x + 1} \\
\underline{x^4 + x^3 \qquad + x} \\
x^3 + x^2 \qquad + 1 \\
\underline{x^3 + x^2 \qquad + 1}
\end{array}
$$

This yields $r_2(x) = 0$ and $q_2(x) = x + 1$.

Therefore, $\gcd[a(x), b(x)] = r_1(x) = x^3 + x^2 + 1$.

Try yourself:

Find gcd of f(x) and g(x) where $f(x) = 2x^3 - 4x^2 + x - 2$ and $g(x) = x^2 - x - 2$

Finite Fields of The Form GF(2^n)

- ✓ The order of a finite field must be of the form $\mathbf{p^n}$, where **p** is a prime and **n** is a positive integer
- ✓ Using modular arithmetic in $\mathbf{Z_p}$, all of the axioms for a field are satisfied.
- ✓ For polynomials over $\mathbf{p^n}$, with $\mathbf{n > 1}$, operations modulo $\mathbf{p^n}$ do not produce a field.
- ✓ In this section, we show what structure satisfies the axioms for a field in a set with $\mathbf{p^n}$ elements and concentrate on **GF(2^n).**
- ✓ Also called Extension Field

Motivation

- Virtually all encryption algorithms, both symmetric and public key, involve arithmetic operations on integers.
- If one of the operations that is used in the algorithm is division, then we need to work in arithmetic defined over a field.
- For convenience and for implementation efficiency, we would also like to work with integers that fit exactly into a given number of bits with no wasted bit patterns.
- That is, we wish to work with integers in the range **0 through 2^n - 1**, which fit into an **n-bit word**.

> Suppose we wish to define a conventional encryption algorithm that operates on data 8 bits at a time, and we wish to perform division. With 8 bits, we can represent integers in the range 0 through 255. However, 256 is not a prime number, so that if arithmetic is performed in Z_{256} (arithmetic modulo 256), this set of integers will not be a field. The closest prime number less than 256 is 251. Thus, the set Z_{251}, using arithmetic modulo 251, is a field. However, in this case the 8-bit patterns representing the integers 251 through 255 would not be used, resulting in inefficient use of storage.

- ✓ If all arithmetic operations are to be used and we wish to represent a full range of integers in **n** bits, then arithmetic modulo $\mathbf{2^n}$ will not work. Equivalently, *the set of integers modulo 2^n for n >1, is not a field.*
- ✓ Furthermore, even if the encryption algorithm uses only addition and multiplication, but not division, the use of the set $\mathbf{Z_{2^n}}$ is questionable

- ✓ Intuitively, it would seem that ***an algorithm that maps the integers unevenly onto themselves might be cryptographically weaker than one that provides a uniform mapping.*** Thus, the finite fields of the form **GF(2^n)** are attractive for cryptographic algorithms.

Integer	1	2	3	4	5	6	7
Occurrences in Z_8	4	8	4	12	4	8	4
Occurrences in GF(2^3)	7	7	7	7	7	7	7

- ✓ We need to find a set consisting of **2^n** elements, together with a definition of addition and multiplication over the set that define a field.
  - We can assign a unique integer in the range **0** through **2^n - 1** to each element of the set
  - We do not use modular arithmetic, because this does not result in a field.
  - Instead, we use polynomial arithmetic which provides a means for constructing the desired field

Modular Polynomial Arithmetic

- Consider the set **S** of all polynomials of degree **n - 1** or less over the field **Z_p**. Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

where each **a_i** takes on a value in the set {0, 1, . . . , p - 1}.

There are a total of **p^n** different polynomials in **S**.

For $p = 3$ and $n = 2$, the $3^2 = 9$ polynomials in the set are

0	x	$2x$
1	$x + 1$	$2x + 1$
2	$x + 2$	$2x + 2$

For $p = 2$ and $n = 3$, the $2^3 = 8$ polynomials in the set are

0	$x + 1$	$x^2 + x$
1	x^2	$x^2 + x + 1$
x	$x^2 + 1$	

With the appropriate definition of arithmetic operations, each such set **S** is a finite field. The definition consists of the following elements

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.
2. Arithmetic on the coefficients is performed **modulo p.** That is, we use the rules of arithmetic for the finite field $\mathbf{Z_p}$.
3. If multiplication results in a polynomial of degree greater than **n - 1**, then **the polynomial is reduced modulo some irreducible polynomial m(x)** of degree **n**. That is, we divide by **m(x)** and keep the remainder.

   For a polynomial **f(x),** the remainder is expressed as **r(x) = f(x) mod m(x).**

   Note: There can be more than one irreducible polynomial in GF(2^n)

Addition

We have seen that addition of polynomials is performed by adding corresponding coefficients, and, in the case of polynomials over Z_2, addition is just the XOR operation. So, addition of two polynomials in GF(2^n) corresponds to a bitwise XOR operation.

Consider the two polynomials in GF(2^8) from our earlier example:

$f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$.

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1)$	$= x^7 + x^6 + x^4 + x^2$	(polynomial notation)
(01010111) ⊕ (10000011)	= (11010100)	(binary notation)
{57} ⊕ {83}	= {D4}	(hexadecimal notation)

Multiplication

If multiplication results in a polynomial of degree greater than **n - 1**, then **the polynomial is reduced modulo some irreducible polynomial m(x)** of degree **n**. That is, we divide by **m(x)** and keep the remainder.

For a polynomial **f(x),** the remainder is expressed as **r(x) = f(x) mod m(x).**

The Advanced Encryption Standard (AES) uses arithmetic in the finite field GF(2^8), with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Consider the two polynomials $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$. Then

$$\begin{aligned} f(x) + g(x) &= x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1 \\ &= x^7 + x^6 + x^4 + x^2 \end{aligned}$$

$$\begin{aligned} f(x) \times g(x) &= x^{13} + x^{11} + x^9 + x^8 + x^7 \\ &\quad + x^7 + x^5 + x^3 + x^2 + x \\ &\quad + x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \end{aligned}$$

$$\begin{array}{r} x^5 + x^3 \\ x^8 + x^4 + x^3 + x + 1 \overline{)\, x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1} \\ \underline{x^{13} + x^9 + x^8 + x^6 + x^5} \\ x^{11} + x^4 + x^3 \\ \underline{x^{11} + x^7 + x^6 + x^4 + x^3} \\ x^7 + x^6 + 1 \end{array}$$

Therefore, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$.

Try yourself

If A(x) = x^2+x+1 and B(x) =x^2+1 then compute C(x) = A(x)*B(x) in GF(2^3).

Consider P(x)= x3+x+1 as irreducible polynomial

Answer : C(x)= x^2+x ➔ A(x)*B(x) mod P(x)

Example: Polynomial Arithmetic Modulo ($x^3 + x + 1$) in GF(2^3)

	+	000	001	010	011	100	101	110	111
		0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+1	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

	×	000	001	010	011	100	101	110	111
		0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+1	x^2	$x+1$

(b) Multiplication

Inversion in GF(2^n)

For a given finite field **GF(2^n)** and the corresponding irreducible reduction polynomial P(x), the inverse A^{-1} of a nonzero element A(x) ∈ GF(2^n) is defined as:

$$A^{-1}(x) \cdot A(x) = 1 \bmod P(x).$$

- Just as the Euclidean algorithm can be adapted to find the greatest common divisor of two polynomials, the extended Euclidean algorithm can be adapted to find the multiplicative inverse of a polynomial. Specifically, the algorithm will find the multiplicative inverse of **b(x) modulo a(x)** if the degree of **b(x)** is less than the degree of **a(x)** and

  gcd[a(x), b(x)] = 1.

- The algorithm can be characterized in the same way as we did for the extended Euclidean algorithm for integers. Given polynomials **a(x)** and **b(x)** with the degree of **a(x)** greater than the degree of b(x), we wish to solve the following equation for the values **v(x), w(x),** and **d(x),** where **d(x) = gcd[a(x), b(x)]:**

  a(x)v(x) + b(x)w(x) = d(x)

  If d(x) = 1, then **w(x)** is the multiplicative inverse of **b(x) modulo a(x).**

In summary,

- GF(2^n) consists of 2^n elements.
- The binary operations + and × are defined over the set. The operations of addition, subtraction, multiplication, and division can be performed without leaving the set. Each element of the set other than 0 has a multiplicative inverse.
- The elements of GF(2^n) can be defined as the set of all polynomials of degree n - 1 or less with binary coefficients. Each such polynomial can be represented by a unique n-bit value.
- Arithmetic is defined as polynomial arithmetic modulo some irreducible polynomial of degree n.

Advanced Encryption Standards (AES) Cipher

The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. The AES cipher (& other candidates) form the latest generation of block ciphers, and now we see a significant increase in the block size - from the old standard of 64-bits up to 128-bits; and keys from 128 to 256-bits. In part this has been driven by the public demonstrations of exhaustive key searches of DES. Whilst triple-DES is regarded as secure and well understood, it is slow, especially in s/w. In a first round of evaluation, ***15 proposed algorithms were accepted****. A* ***second round narrowed the field to 5 algorithms.*** *NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001.* ***NIST selected Rijndael as the proposed AES algorithm****. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium:* ***Dr. Joan Daemen*** *and* ***Dr.Vincent Rijmen****.*

- *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*
- *issued as FIPS PUB 197 standard in Nov-2001*

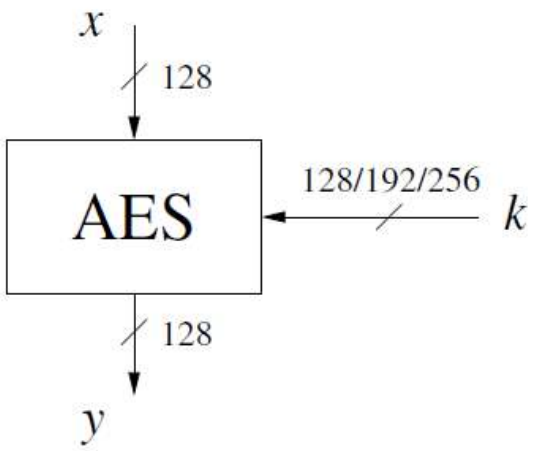
→ The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001. **AES is a symmetric block cipher** that is intended to replace DES as the approved standard for a wide range of applications

→ A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

→ 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 GF(2^8).**

→ The AES 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.



→ 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.

- AES was designed to have:
  - resistance against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES Structure

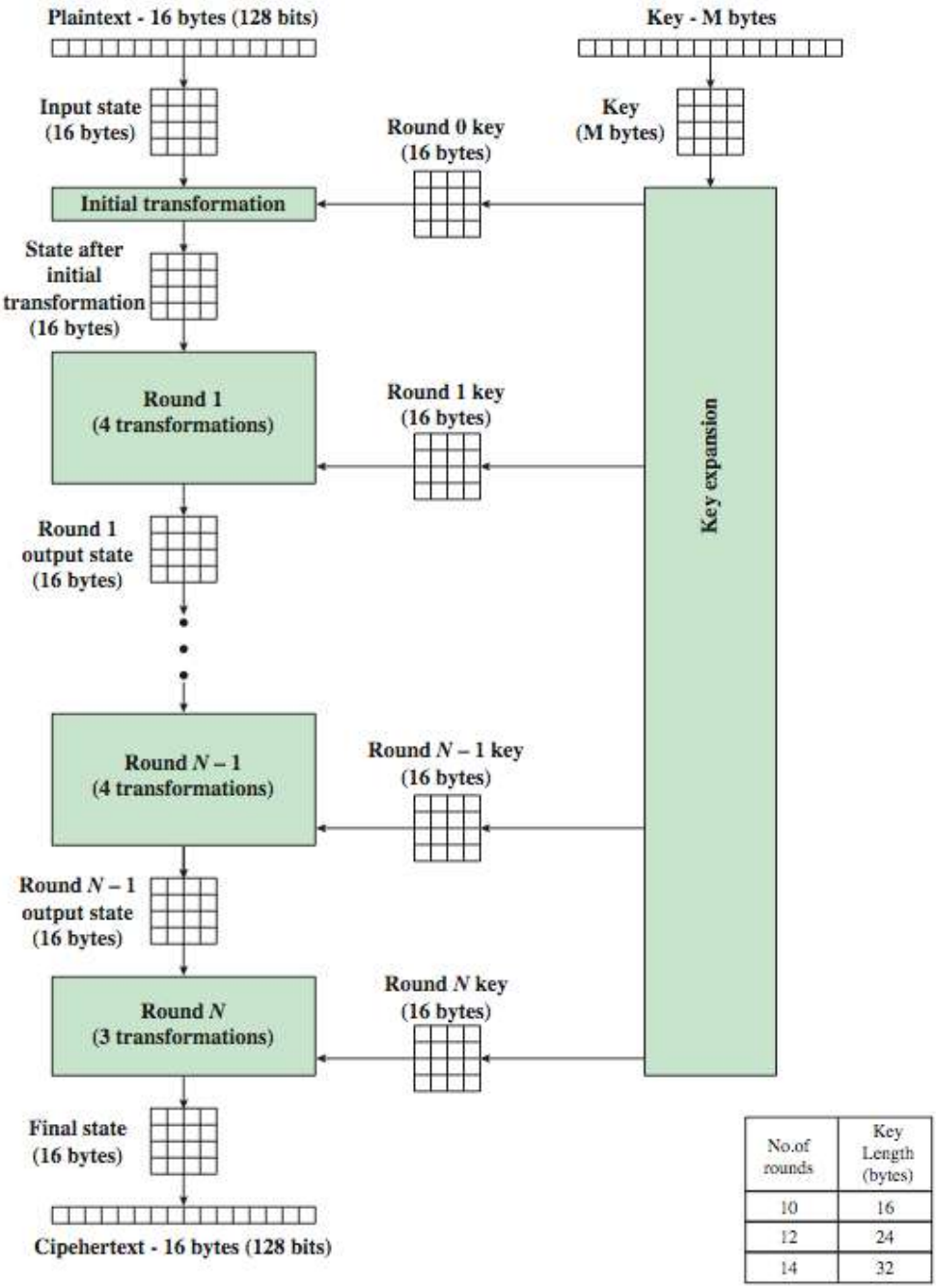


Figure: AES Encryption Process

- AES Cipher is an **iterative** rather than **Feistel** cipher

  — processes data as block of 4 columns of 4 bytes
  — operates on entire data block in every round

  One noteworthy feature of this structure is that it is not a Feistel structure. Recall that, in the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation

- 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

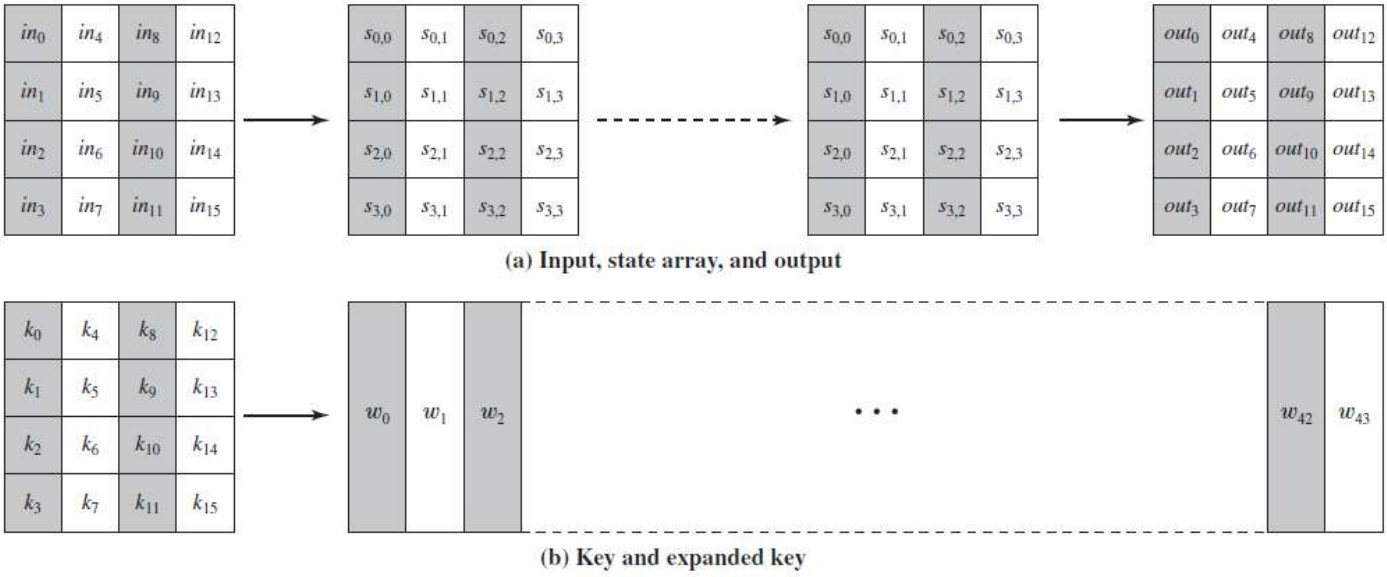


Figure: AES Data Structure

- 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

❖ The first N - 1 rounds consist of four distinct transformation functions**:**

1. **SubBytes**
2. **ShiftRows**
3. **MixColumns**
4. **AddRoundKey**

❖ 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. Figure above shows that the output of each round is a4 × 4 matrix, with the output of the final round being the ciphertext.

❖ Also, the key expansion function generates N + 1 round keys, each of which is a distinct 4 × 4 matrix.

❖ Each round key serves as one of the inputs to the **AddRoundKey** transformation in each round.

Detailed Structure of AES (Encryption and Decryption)

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

— 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. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure

— 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

$$A \oplus B \oplus B = A$$

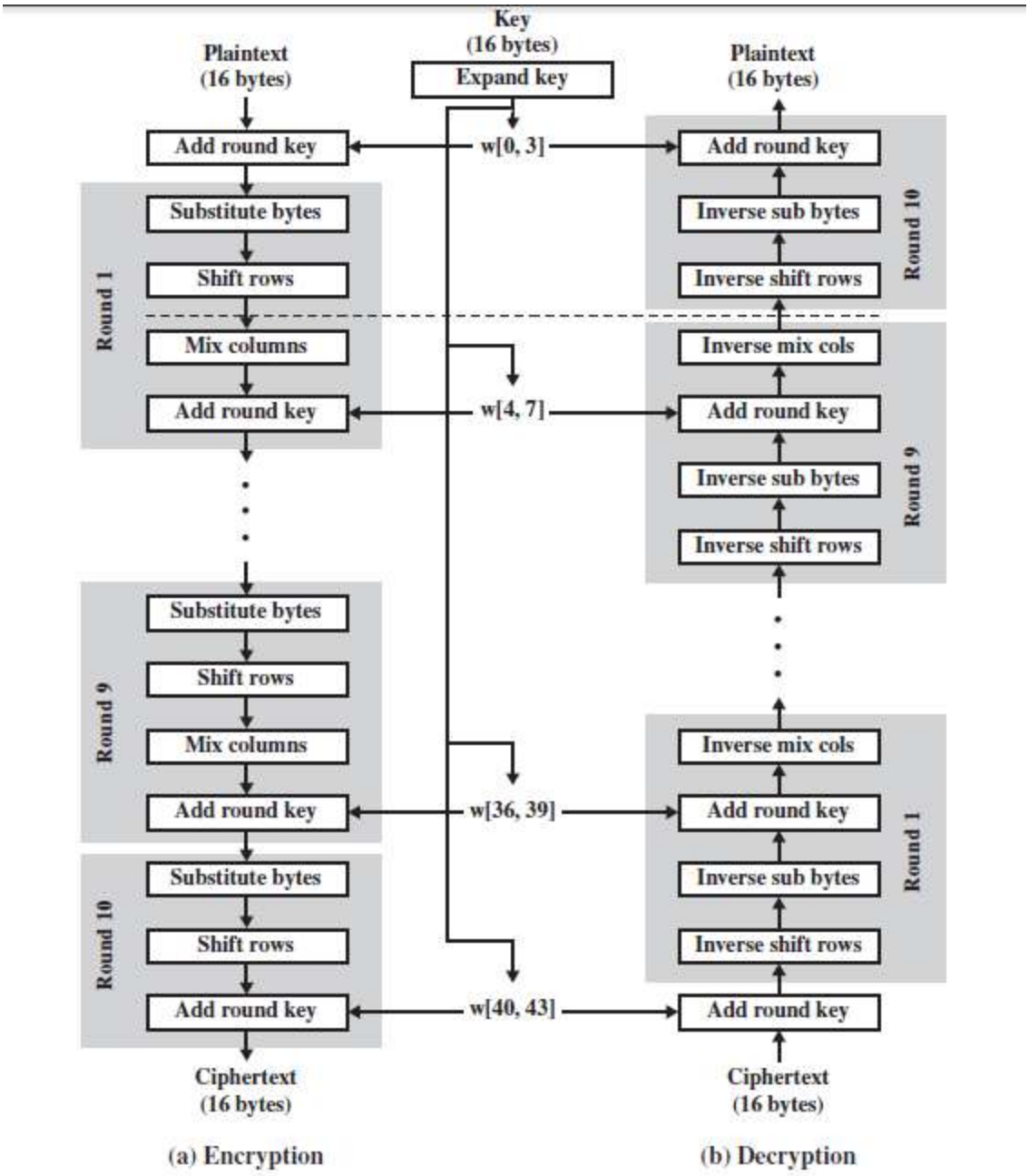


Figure: AES Encryption and Decryption

— As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

— Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure above lays out encryption and decryption going in opposite

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

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

AES Encryption process in single Round

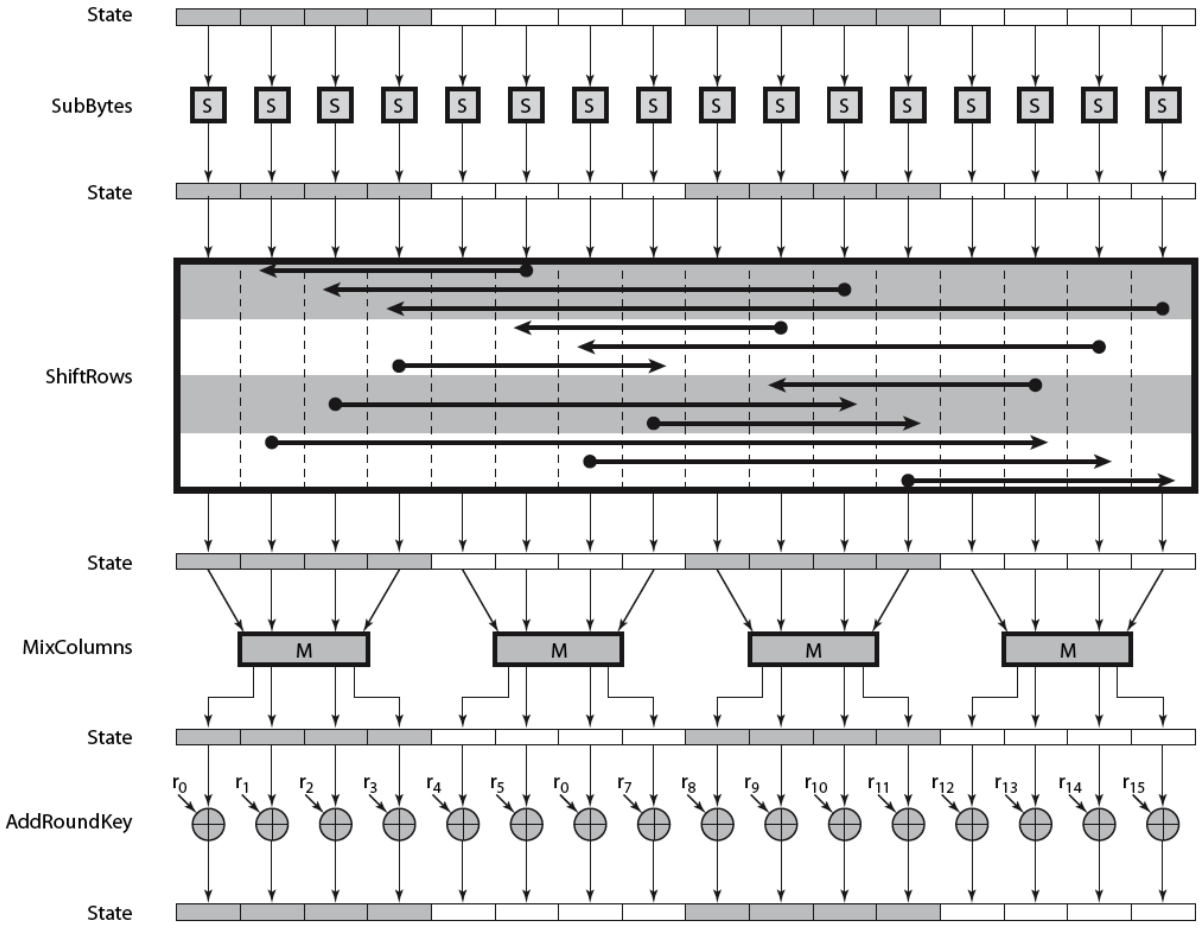


Figure: AES Encryption Round

Substitute Bytes Transformation

- The Byte Substitution layer can be viewed as a row of 16 parallel S-Boxes, each with 8 input and output bits. Note that all 16 S-Boxes are identical, unlike DES where eight different S-Boxes are used.
- Provides confusion.
- In the layer, each state byte Ai is replaced, i.e., substituted, by another byte Bi:

  S(Ai) = Bi.

- 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
  - For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.
    - {95}➔ {2A}.
- The S-Box is the only nonlinear element of AES, i.e., it holds that ByteSub(A)+ ByteSub(B) ≠ ByteSub(A+B) for two states A and B.
- The S-Box substitution is a bijective mapping, i.e., each of the 2^8 = 256 possible input elements is one-to-one mapped to one output element. This allows us to uniquely reverse the S-Box, which is needed for decryption.
- In software implementations the S-Box is usually realized as a 256-by-8 bit lookup table with fixed entries. (see table below)

  Let's assume the input byte to the S-Box is Ai = $(C2)_{hex}$, then the substituted value is

  $S((C2)_{hex}) = (25)_{hex}$.

  On a bit level—and remember, the only thing that is ultimate of interest in encryption is the manipulation of bits— this substitution can be described as:

$$S(11000010) = (00100101).$$

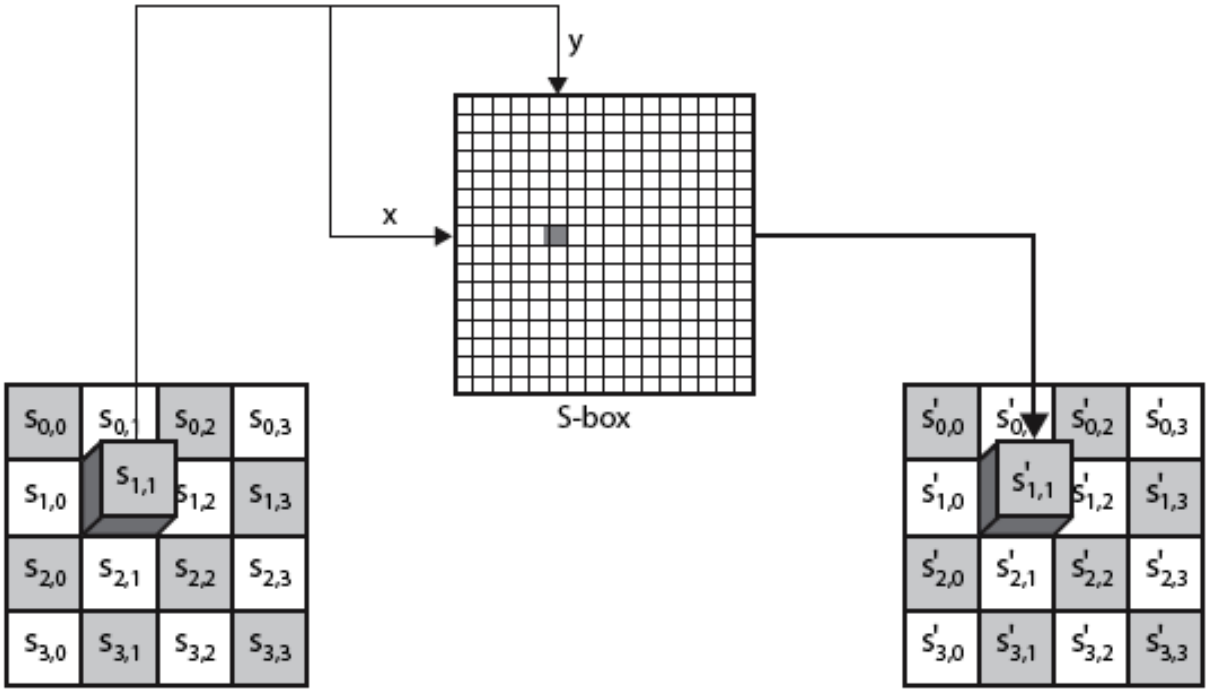


Figure: Substitute byte transformation

Example:

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

→

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

Note: Following tables show the S-Box and Inverse S-Box used in AES

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

(a) S-box

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

(b) Inverse S-box

Note: Following table shows the multiplicative inverse in GF(2^8) for bytes xy used within the AES S-Box

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

From the above table multiplicative inverse of $x^7+x^6+x = (11000010)_2 = (C2)_{16} = (xy)$ is given by the element in row C, column 2:

$$(2F)_{16} = (00101111)_2 = x^5+x^3+x^2+x+1.$$

This can be verified by multiplication:

$$(x^7+x^6+x) \cdot (x5+x3+x2+x+1) \equiv 1 \bmod P(x).$$

Here, $P(x) = x^8 +x^4 +x^3 +x+1$ is used as irreducible polynomial for AES

Mathematical description of the S-Box

→ Unlike the DES S-Boxes, which are essentially random tables that fulfill certain properties, the *AES S-Boxes have a strong algebraic structure*.

→ An AES S-Box can be viewed as *a twostep mathematical transformation*

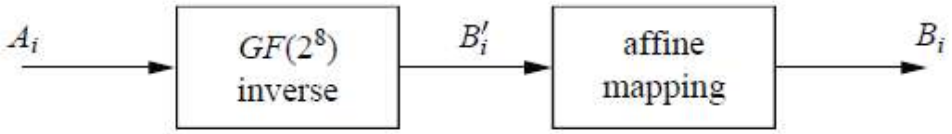


Figure : The two operations within the AES S-Box which computes the function $B_i = S(A_i)$

— The first part of the substitution is a Galois field inversion

— For each input element Ai, the inverse is computed:

$$\mathbf{B'_i = A^{-1}_i}$$

where both $\mathbf{A_i}$ and $\mathbf{B'_i}$ are considered elements in the field $\mathbf{GF(2^8)}$ with the fixed irreducible polynomial $\mathbf{P(x) = x^8 + x^4 + x^3 + x + 1}$

— Note that the inverse of the zero element does not exist. However, for AES it is defined that the zero element Ai = 0 is mapped to itself.

— In the second part of the substitution, each byte $\mathbf{B'_i}$ is multiplied by a constant bit matrix followed by the addition of a constant 8-bit vector. The operation is described by:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \bmod 2.$$

— Note that $\mathbf{B'_i = (b'_7, \ldots, b'_0)}$ is the bitwise vector representation of $\mathbf{B'_i(x) = A^{-1}_i(x)}$.

— This second step is referred to as affine mapping.

Example: We assume the S-Box input $A_i = (11000010)_2 = (C2)_{hex}$.

From inverse table (see above) we can see the inverse is:

$$A_i^{-1} = B_i' = (2F)_{hex} = (00101111)_2.$$

We now apply the $\mathbf{B'_i}$ bit vector as input to the affine transformation. Note that the least significant bit (lsb) $\mathbf{b'_0}$ of $\mathbf{B'_i}$ is at the rightmost position

$$B_i = (00100101) = (25)_{hex}$$

Hence ,

$$S((C2)_{hex}) = (25)_{hex}.$$

Note: The advantage of using inversion in GF(2^8) as the core function of the Byte Substitution layer is that it provides a high degree of nonlinearity, which in turn provides optimum protection against some of the strongest known analytical attacks. The affine step "destroys" the algebraic structure of the Galois field, which in turn is needed to prevent attacks that would exploit the finite field inversion.

ShiftRows Transformation

- Provides diffusion.

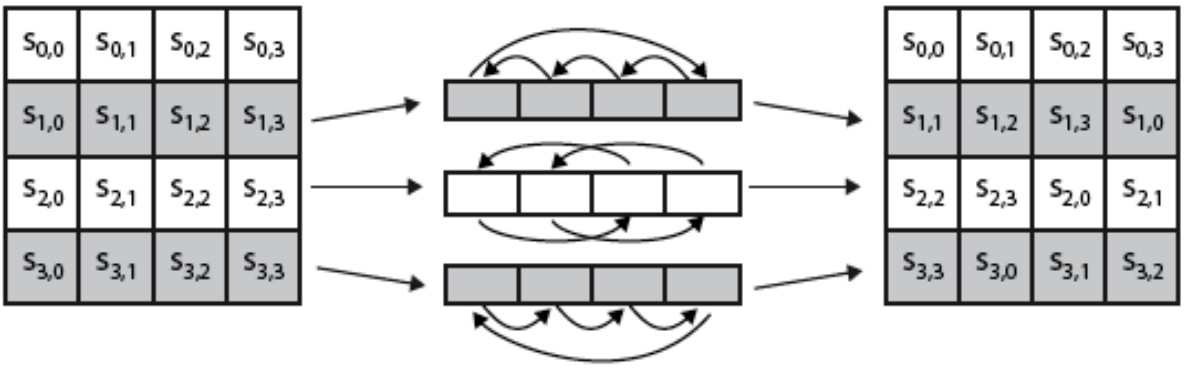


Figure: Pictorial illustration of forward shift row transformation in AES

- In **ShiftRows** transformation (**forward shift row 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.
- If the input of the **ShiftRows** is given as a state matrix $B = (B_0, B_1, \ldots, B_{15})$:

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

the output is the new state:

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	⟵ one position left shift
B_{10}	B_{14}	B_2	B_6	⟵ two positions left shift
B_{15}	B_3	B_7	B_{11}	⟵ three positions left shift

- The following is an example of **ShiftRows**.

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

→

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

- 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

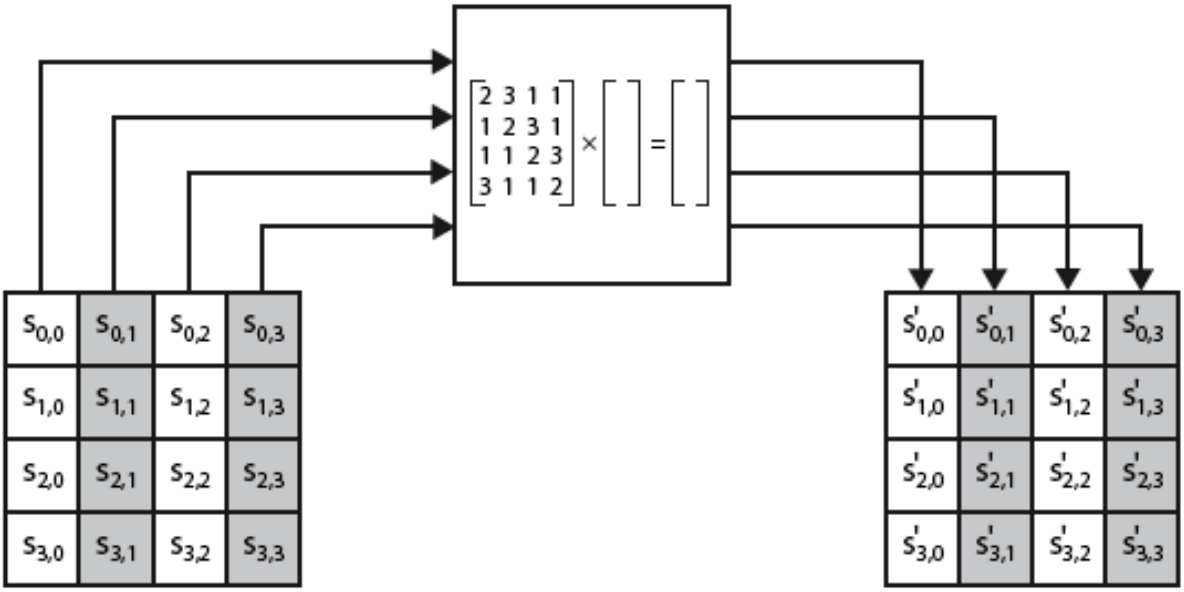


Figure: Pictorial illustration of MixColumn transformation

- 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 **MixColumn** step is a linear transformation which mixes each column of the state matrix. Since every input byte influences four output bytes, the **MixColumn** operation is the major diffusion element in AES.
- The combination of the **ShiftRows** and **MixColumn** layer makes it possible that after only three rounds every byte of the state matrix depends on all 16 plaintext bytes.
- In the following, we denote the 16-byte input state by B and the 16-byte output state by C:

  MixColumn(B) =C,

  where B is the state after the **ShiftRows** operation .

  Now, each 4-byte column is considered as a vector and multiplied by a fixed 4×4 matrix. The matrix contains constant entries. Multiplication and addition of the coefficients is done in GF(2^8). As an example, we show how the first four output bytes are computed:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

The second column of output bytes (C_4, C_5, C_6, C_7) is computed by multiplying the four input bytes (B_4, B_9, B_{14}, B_3) by the same constant matrix, and so on.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix} = \begin{bmatrix} C_0 & C_4 & C_8 & C_{12} \\ C_1 & C_5 & C_9 & C_{13} \\ C_2 & C_6 & C_{10} & C_{14} \\ C_3 & C_7 & C_{11} & C_{15} \end{bmatrix}$$

Note:

- Each state byte C_i and B_i is an 8-bit value representing an element from GF(2^8).
- All arithmetic involving the coefficients is done in this Galois field.
- For the constants in the matrix a hexadecimal notation is used: "01" refers to the GF(2^8) polynomial with the coefficients (00000001), i.e., it is the element 1 of the Galois field; "02" refers to the polynomial with the bit vector (00000010), i.e., to the polynomial x; and "03" refers to the polynomial with the bit vector (00000011), i.e., the Galois field element x+1.

Following is an example of **MixColumns** transformation

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

→

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

AddRoundKey Transformation

- 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 .

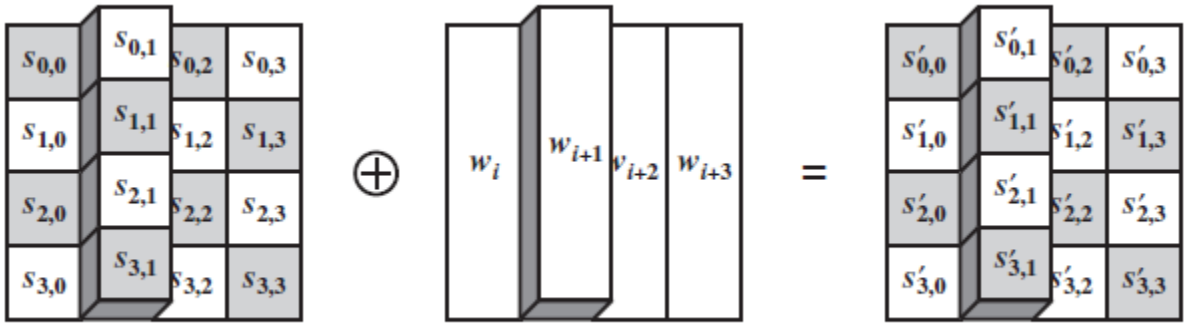


Figure: Add round key transformation

- The operation is viewed as a column wise 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**:

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

⊕

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

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

1. The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

AES Key Schedule

- The AES key schedule is word-oriented, where 1 word = 32 bits.
- Subkeys are stored in a key expansion array **W** that consists of words.
- There are different key schedules for the three different AES key sizes of 128, 192 and 256 bit, which are all fairly similar.

Key Expansion Algorithm (for 128 bit key)

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

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

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

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

- For a word whose position in the **W** array is a multiple of 4, a more complex function is used.
- Figure below illustrates the generation of the expanded key, using the symbol **g** to represent that complex function. The function g consists of the following subfunctions:
    1. **RotWord** performs a one-byte circular left shift on a word. This means that an input word $[B_0, B_1, B_2, B_3]$ is transformed into $[B_1, B_2, B_3, B_0]$.
    2. **SubWord** performs a byte substitution on each byte of its input word, using the S-box
    3. The result of steps 1 and 2 is **XORed** with a round constant, **Rcon[j].**
- The **round constant** is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with **Rcon** is to only perform an XOR on the leftmost byte of the word.
- The round constant is different for each round and is defined as

  Rcon[j] = (RC[j], 0, 0, 0), with **RC[1] = 1, RC[j] = 2 · RC[j-1]** and with multiplication defined over the field GF(2^8).

  The values of RC[j] in hexadecimal are:

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

For example, suppose that the round key for round 8 is

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

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

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

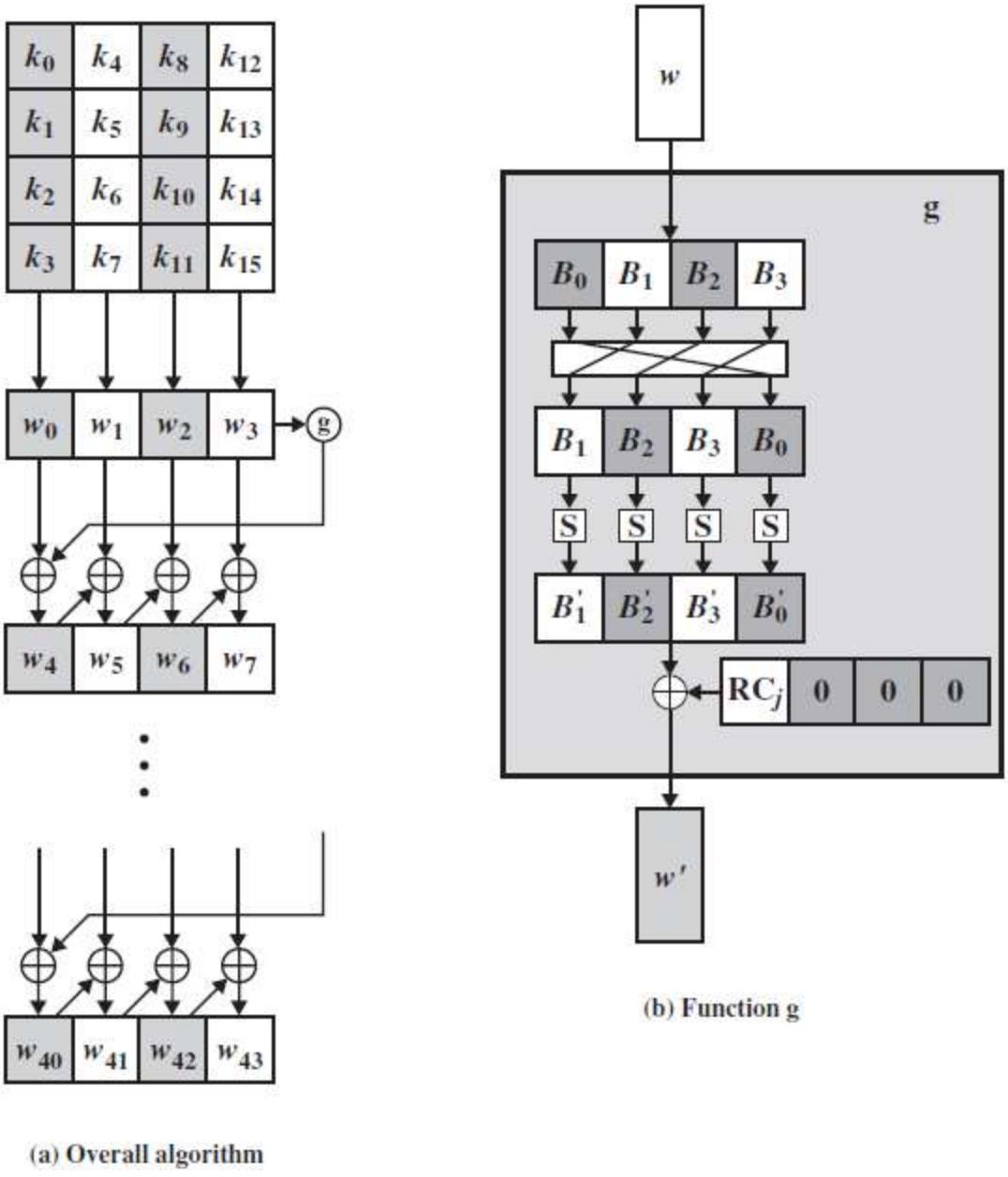


Figure: AES Key Expansion

International Data Encryption Algorithm (IDEA)

- The International Data Encryption Algorithm (IDEA), originally called Improved Proposed Encryption Standard (IPES), is a **symmetric-key block cipher** designed by James Massey and Xuejia Lai and was first described in 1991.
- The original algorithm went through few modifications and finally named as International Data Encryption Algorithm (IDEA)
- The algorithm was intended as a replacement for the Data Encryption Standard (DES).
- It entirely avoids the use of any lookup tables or S-boxes
- IDEA was used as the symmetric cipher in early versions of the Pretty Good Privacy (PGP) cryptosystem
- IDEA is not a Feistel cipher.

Detailed description of IDEA

→ IDEA operates with **64-bit plaintext** and **cipher text blocks** and is controlled by a **128-bit key**

→ The 64 bit plain text is divided into **4 sub-blocks**, **each of 16 bits in size**

→ The 128 bit key is used to generate **52 sub keys of length 16 bits**.

→ **Number of identical rounds is 8** and **in each round 6 keys are used.**

→ Like this 48 keys are used and at last round (also known as **half-round**) another 4 keys (8×6+4=52 in total) are used in both encryption and decryption process.

→ IDEA derives much of its security by mixing operations from different groups (incompatible)— **modular addition (Addition modulo 2^{16} denoted with $\boxplus$), modular multiplication (Multiplication modulo $2^{16}+1$ denoted $\odot$)** and **Bitwise XOR (exclusive OR denoted with $\oplus$).**

→ Completely *avoids substitution boxes* and table lookups used in the block ciphers

→ The algorithm structure has been chosen such that when different key sub-blocks are used, the *encryption process is identical to the decryption process*

→ The following figures shows a round (1-8) and final "half round".

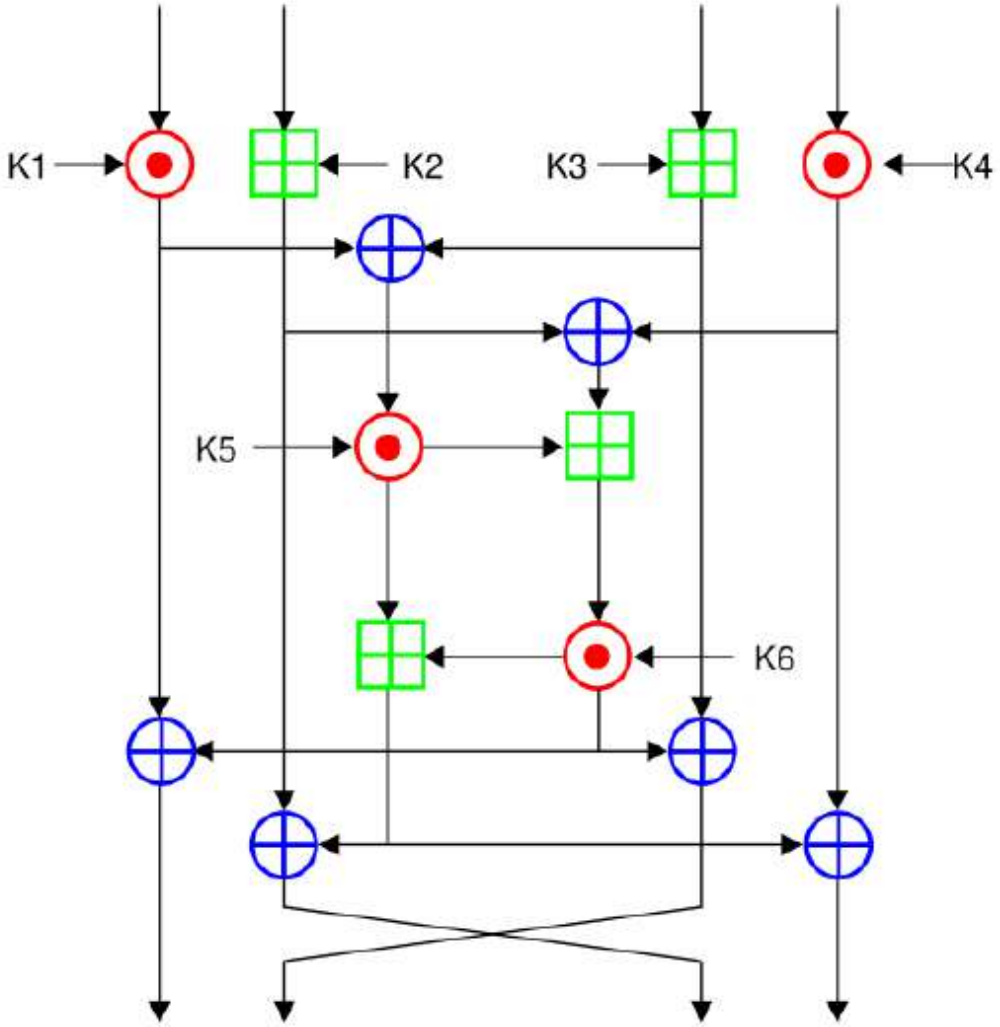


Figure: An identical round in IDEA

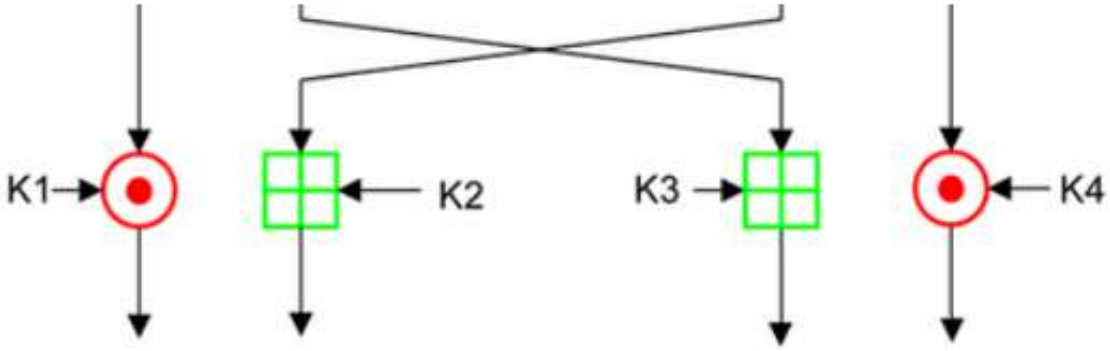


Figure: The half round in IDEA

Key schedule in IDEA

- First, the 128-bit key is partitioned into eight 16-bit sub-blocks which are then directly used as the first eight key sub-blocks
- 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

The 128-bit key of IDEA is taken as the first eight subkeys, K1 through K8. The next eight subkeys are obtained the same way, after a 25-bit circular left shift, and this is repeated until all encryption subkeys are derived.

Round Operations in IDEA

→ It has been mentioned above that IDEA uses 8 full rounds and 1 half round.

→ We now break the 8 full round and make it 16 rounds such that there are total 17 rounds. Whrere,

- **9 odd rounds** (1, 3, …., 17) are identical and
- **8 even rounds** (2, 4, ….,16) are identical.
- **each odd round takes 4 sub keys** and **each even round takes 2 sub keys.**

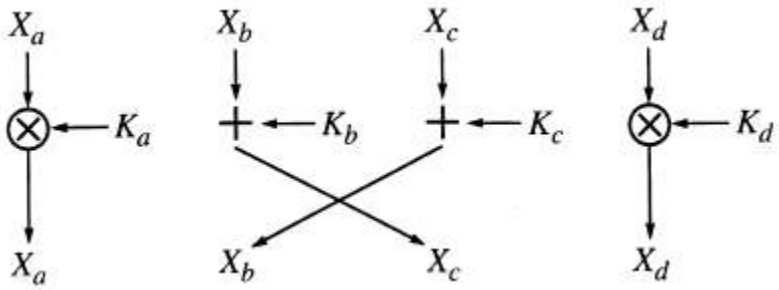
Odd Round:

→ Odd round is simple and **uses four keys** where **first and fourth sub keys** are used in **multiplication mod $2^{16}+1$** operation with first and fourth 16 bits fragment of 64 bits input block respectively.

→ The **second and third sub keys** are subjected to **addition mod 2^{16}** with second and third bits fragment of 64 bits input block respectively.

→ The output so obtained from **operations with first and fourth input sub-block will be first and fourth input for next round** and **output from operations with second and third input sub-block will be reversed i.e. becomes third and second input for next round.**

→ If $\mathbf{X_a, X_b, X_c}$, and $\mathbf{X_d}$ are input sub-blocks and $\mathbf{K_a, K_b, K_c}$, and $\mathbf{K_d}$ are sub keys then



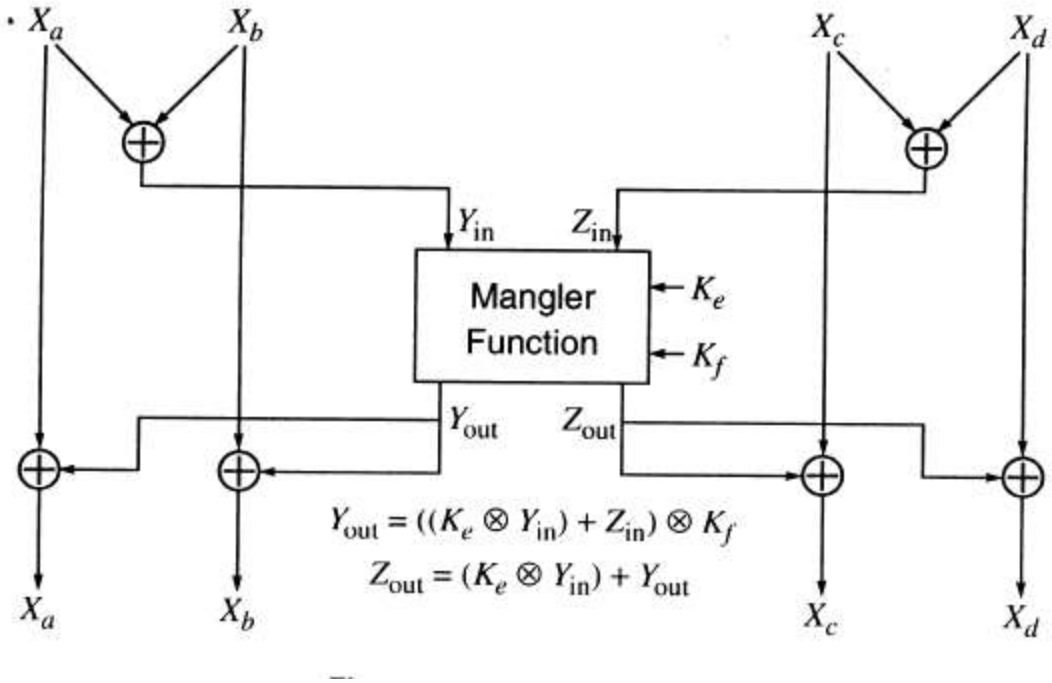
Even Round:

→ Even round is bit complicated then the odd round.

→ There are four input sub-blocks ($\mathbf{X_a, X_b, X_c}$, and $\mathbf{X_d}$) from the previous round and two sub keys ($\mathbf{K_e}$ and $\mathbf{K_f}$).

→ In even rounds **first and second input sub-blocks are subjected to XOR operation** to get single 16 bits output (say, $\mathbf{Y_{in}}$) and **third and fourth input sub-blocks are subjected to XOR operation** to get single 16 bits output (say, $\mathbf{Z_{in}}$).

→ $\mathbf{Y_{in}}$ and $\mathbf{Z_{in}}$ are fed to a function along with $\mathbf{K_e}$, and $\mathbf{K_f}$ to get outputs $\mathbf{Y_{out}}$ and $\mathbf{Z_{out}}$, where $\mathbf{Y_{out}}$ is **XORed** with $\mathbf{X_a}$ and $\mathbf{X_b}$,, to get first two input sub-blocks for next round and $\mathbf{Z_{out}}$ is **XORed** with $\mathbf{X_c}$ and $\mathbf{X_d}$, to get last two input sub-blocks for next round.



Decryption in IDEA

- The computational process used for decryption of the ciphertext is essentially the same as that used for encryption
- The only difference is that each of the 52 16-bit key sub-blocks used for decryption is the inverse of the key sub-block used during encryption
- In addition, the key sub-blocks must be used in the reverse order during decryption in order to reverse the encryption process

Modes of Block Cipher Encryptions

- 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.
- When multiple blocks of plaintext are encrypted using them same key, a number of security issues arise.

- To apply a block cipher in a variety of applications, **five modes of operation** have been defined by NIST (SP 800-38A).
- In essence, ***a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.***
- The five modes are intended to cover a wide variety of applications of encryption for which a block cipher could be used.
- These modes are intended for use with any symmetric block cipher, including triple DES and AES.
- In the previous chapters we introduced how DES, 2DES, 3DES, IDEA and AES encrypt a block of data. Of course, in practice one wants typically to encrypt more than one single 8-byte or 16-byte block of plaintext, e.g., when encrypting an e-mail or a computer file. There are several ways of encrypting long plaintexts with a block cipher.

The five modes of Operation in Block Cipher (defined by NIST) are :

1. Electronic Code Book mode (ECB)
2. Cipher Block Chaining mode (CBC)
3. Cipher Feedback mode (CFB)
4. Output Feedback mode (OFB)
5. Counter mode (CTR).

Electronic Codebook Mode (ECB)

- The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key.
- The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.
- For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary.
- Decryption is performed one block at a time, always using the same key.

- In the following figure, the plaintext (padded as necessary) consists of a sequence of b-bit blocks, $P_1, P_2, \ldots, P_N$; the corresponding sequence of ciphertext blocks is $C_1, C_2, \ldots, C_N$.
- 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. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use.
- The most significant characteristic of ECB is that if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext.
- 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.

  (Example : Substitution attack against electronic bank transfer (see page 126 of the book Understanding Cryptography by Christof Paar and Jan Pelzl)

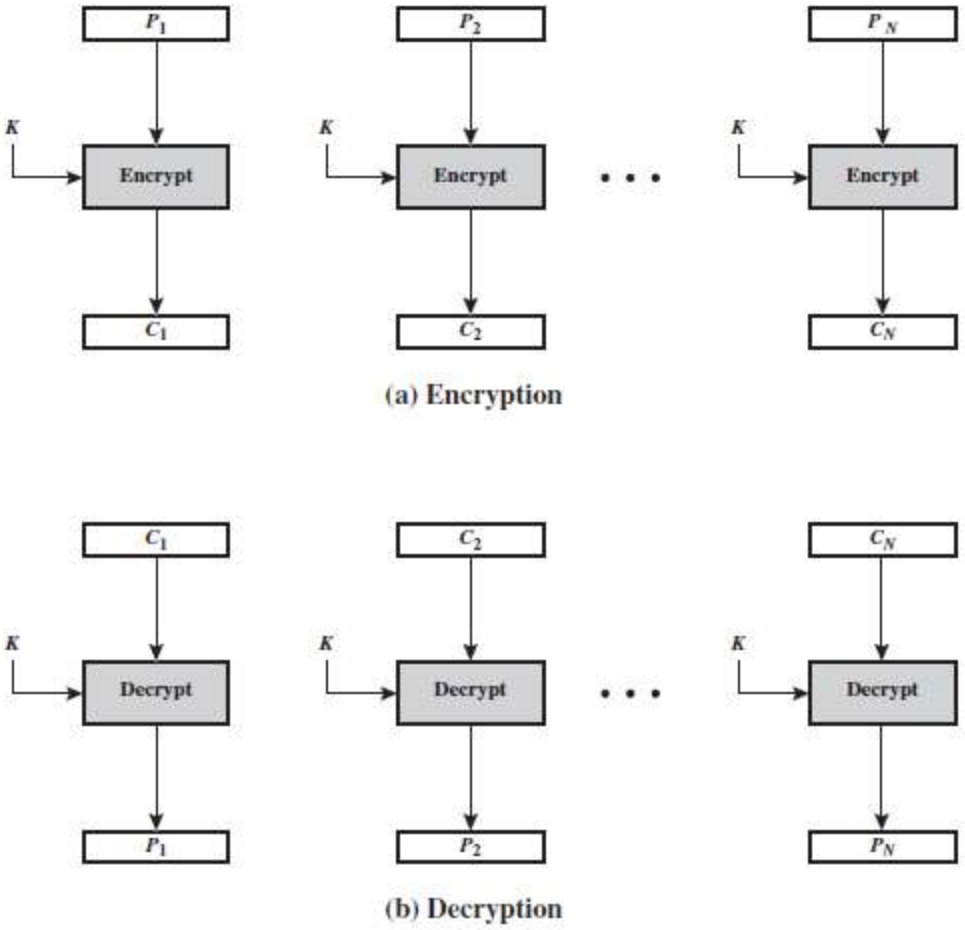


Figure: Electronic Codebook (ECB) Mode

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 (CBC)

- To overcome the problems of repetitions and order independence in ECB, there is a need some way of making the ciphertext dependent on all blocks before it. This is what CBC gives us, by combining the previous ciphertext block with the current message block before encrypting.
    - Message is broken into blocks
    - Linked together in encryption operation
    - Each previous cipher blocks is chained with current plaintext block, hence name is Cipher Block Chaining
- 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.
- To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is the same size as the cipher block. We can define CBC mode as

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

NOTE: Initialization Vector (IV) is usually well known (often all 0's)

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

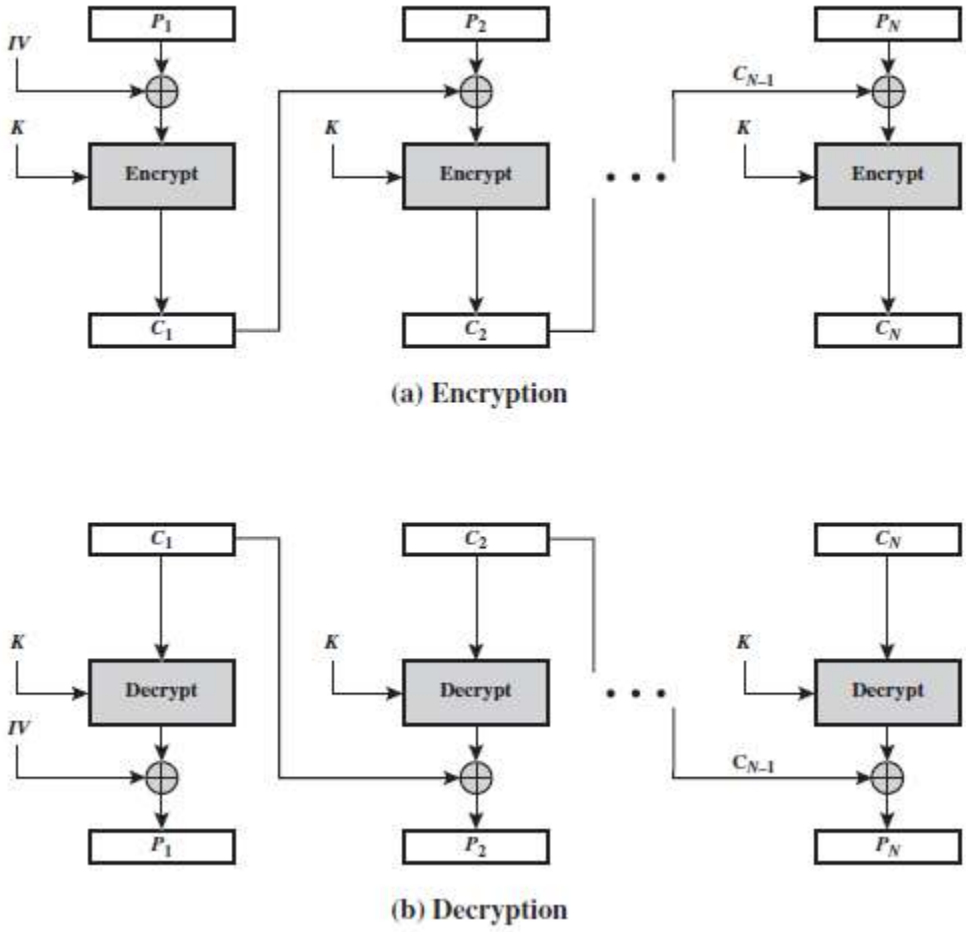


Figure: Cipher Block Chaining (CBC) Mode

- CBC is the block mode generally used. The chaining provides an avalanche effect, which means the encrypted message cannot be changed or rearranged without totally destroying the subsequent data. However there is the issue of ensuring that the IV is either fixed or sent encrypted in ECB mode to stop attacks on 1st block.

Note: For AES, DES, or any block cipher, encryption is performed on a block of b bits. In the case of DES, b = 64 and in the case of AES, b = 128. However, it is possible to convert a block cipher into a stream cipher, using one of the following three modes:

- Cipher Feedback (CFB) mode
- Output Feedback (OFB) mode
- Counter (CTR) mode.

Cipher Feedback Mode (OFB)

- 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 is Output Feedback Mode)
- 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)

✓ Figure below 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.

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 $\mathbf{P_1}$ to produce the first unit of ciphertext $\mathbf{C_1}$, which is then transmitted.

— In addition, the contents of the shift register are **shifted left by s bits**, and C_1 is placed in the rightmost (least significant) s bits of the shift register.

— This process continues until all plaintext units have been encrypted.

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.

Note that it is the encryption function that is used, not the decryption function.

We can define CFB mode as follows.

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

Note:

Although CFB can be viewed as a stream cipher, it does not conform to the typical construction of a stream cipher. In a typical stream cipher, the cipher takes as input some initial value and a key and generates a stream of bits, which is then XORed with the plaintext bits. In the case of CFB, the stream of bits that is XORed with the plaintext also depends on the plaintext.

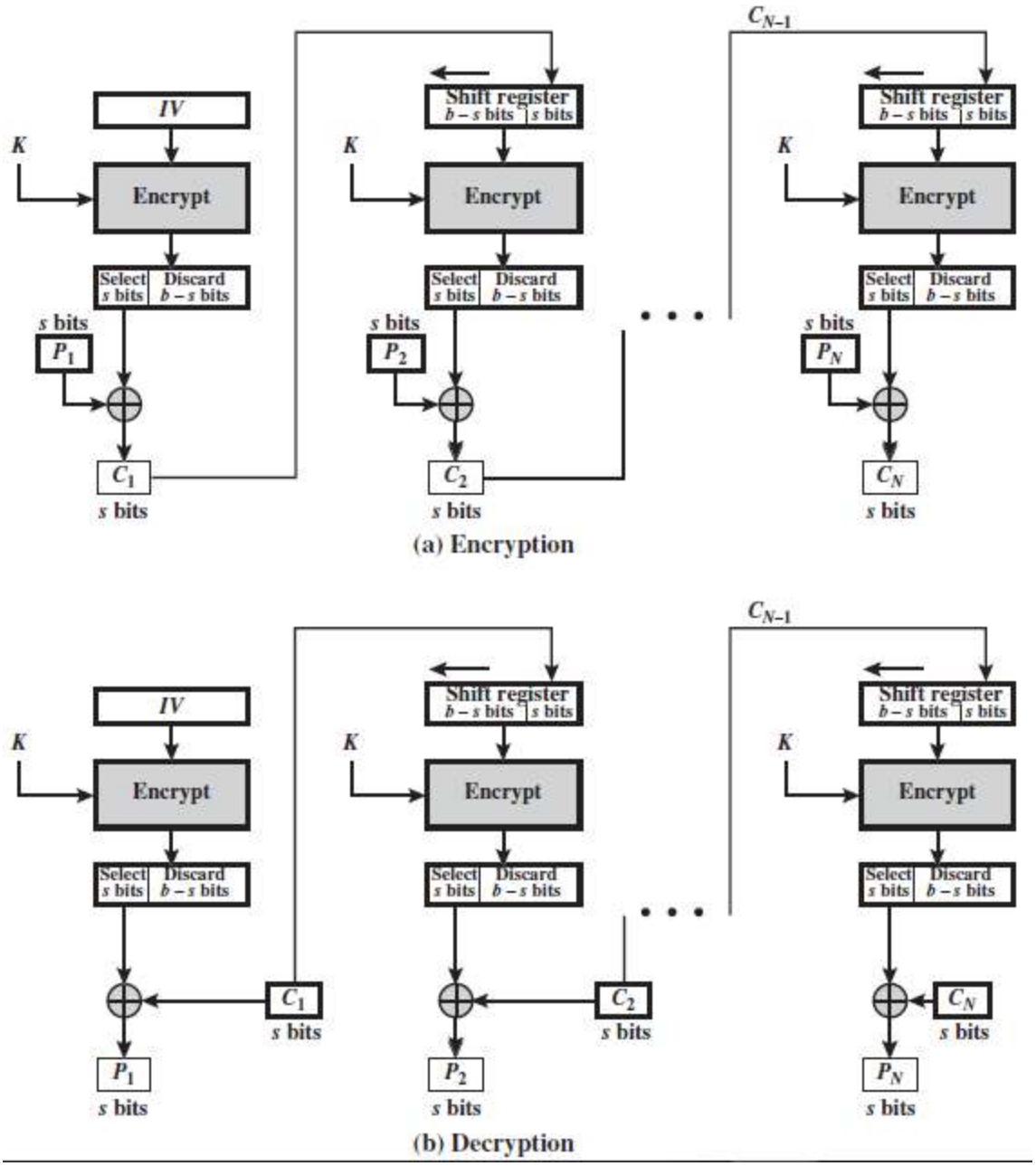


Figure: s-bit Cipher Feedback (CFB) Mode

Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- limitation is need to stall while do block encryption after every n-bits
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propogate for several blocks after the error

Output Feedback Mode (OFB)

- 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.

    - Message is treated as a stream of bits
    - Output of cipher is added to message
    - Output is then feed back (hence name is Output Feedback Mode)
    - Feedback is independent of message
        - can be computed in advance

We can define OFB mode as follows.

> Let the size of a block be **b**. If the last block of plaintext contains **u** bits (indicated by *****), with **u < b**, the most significant **u** bits of the last output block $\mathbf{O_N}$ are used for the XOR operation; the remaining **b - u** bits of the last output block are discarded.

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

Note: As with CBC and CFB, the OFB mode requires an initialization vector. In the case of OFB, the IV must be a *nonce*; that is, the IV must be unique to each execution of the encryption operation.

- One advantage of the OFB method is that bit errors in transmission do not propagate.
- The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

Advantages and Limitations of OFB

- Bit errors do not propagate
- More vulnerable to message stream modification
- 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

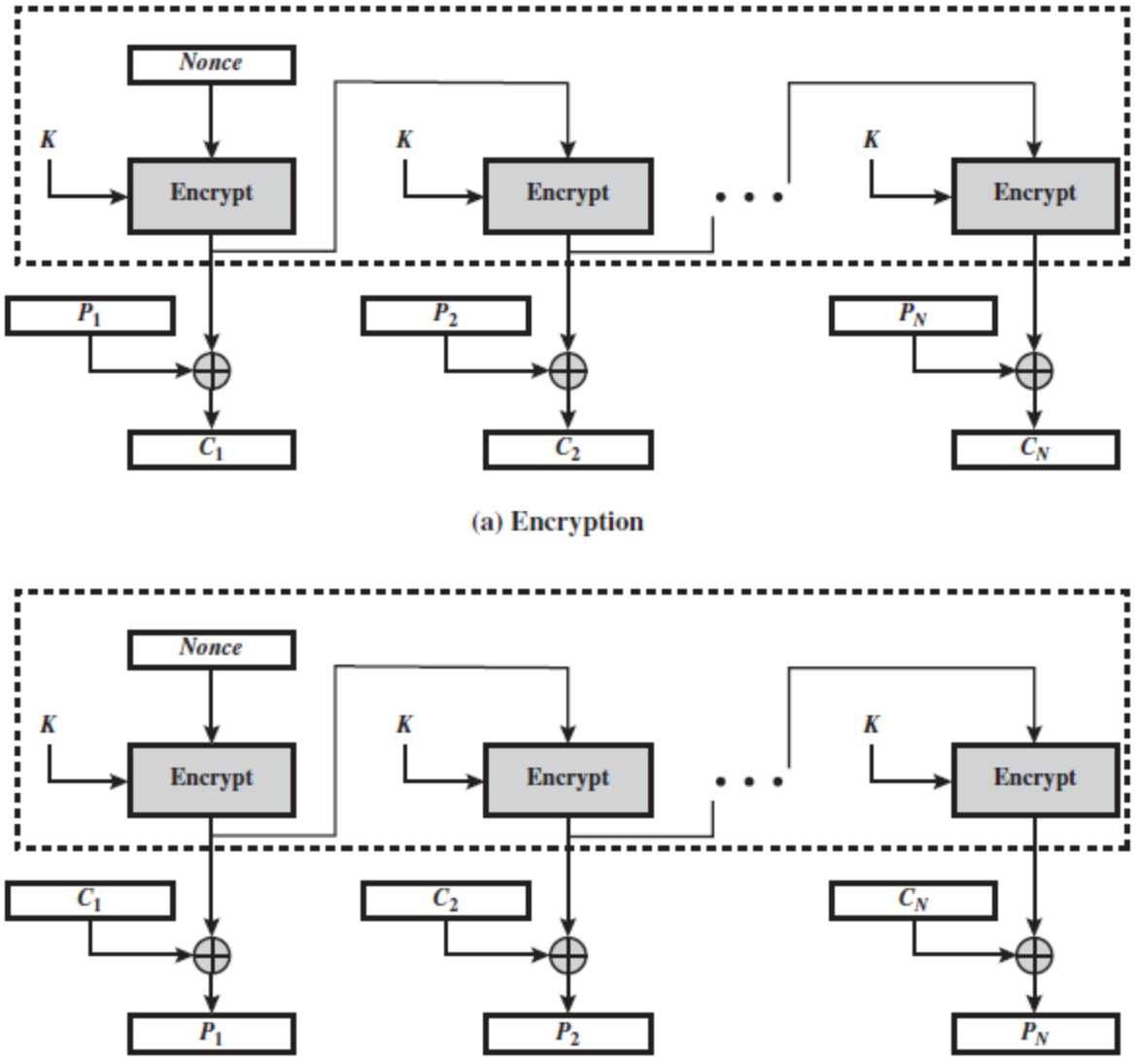


Figure: Output Feedback (OFB) Mode

Counter (CTR) mode

- Figure below depicts the CTR mode.
- A counter equal to the plaintext block size is used.
- The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted.
- Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (**modulo 2^b**, where **b** is the block size).
- For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; **there is no chaining**.
- For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.
    - Thus, the initial counter value must be made available for decryption.
- Given a sequence of counters $T_1, T_2, \ldots, T_N$, 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
- As with the OFB mode, the initial counter value must be a nonce; that is, T_1 must be different for all of the messages encrypted using the same key. Further, all T_i values across all messages must be unique.

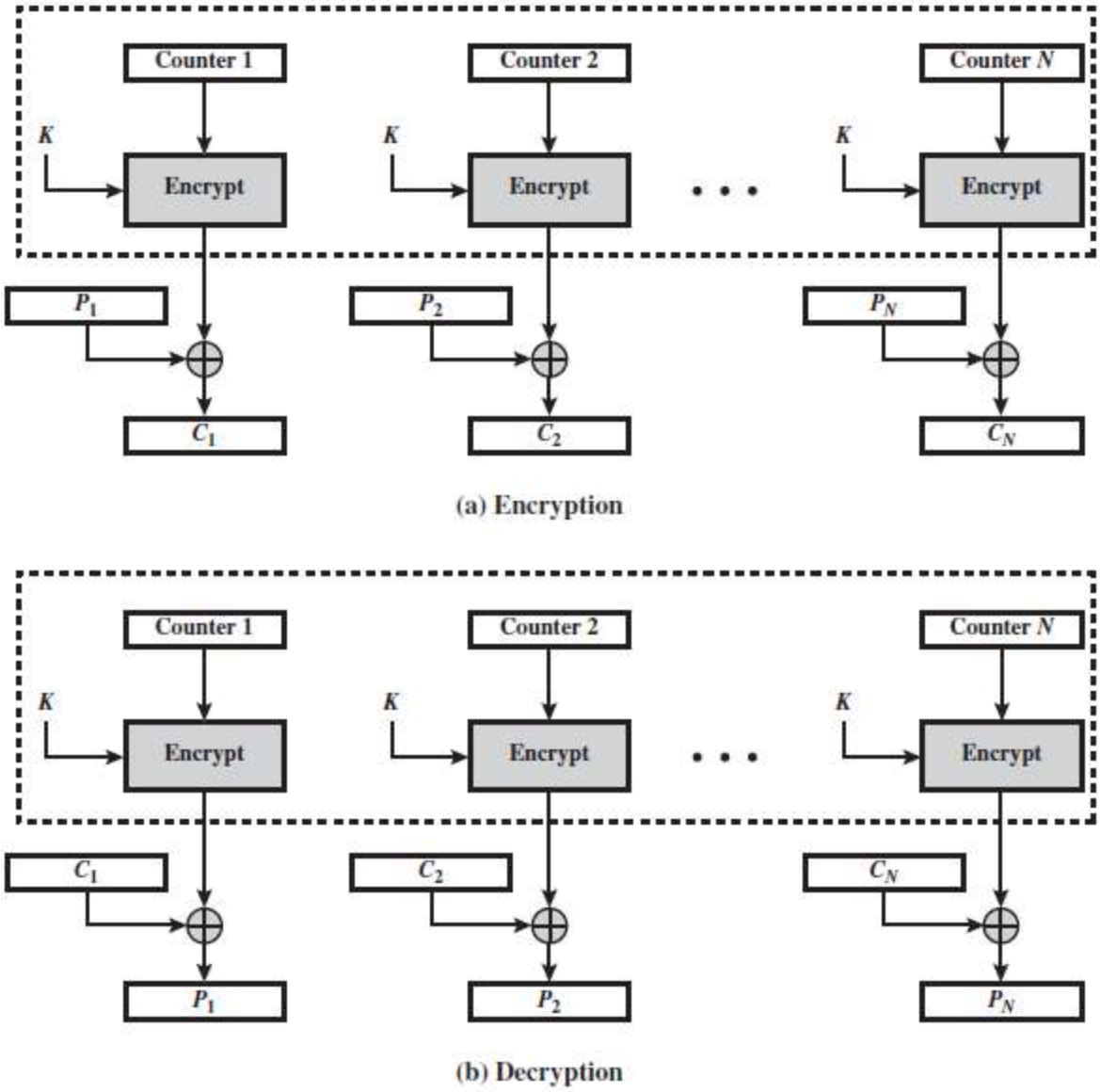


Figure: Counter (CTR) Mode

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