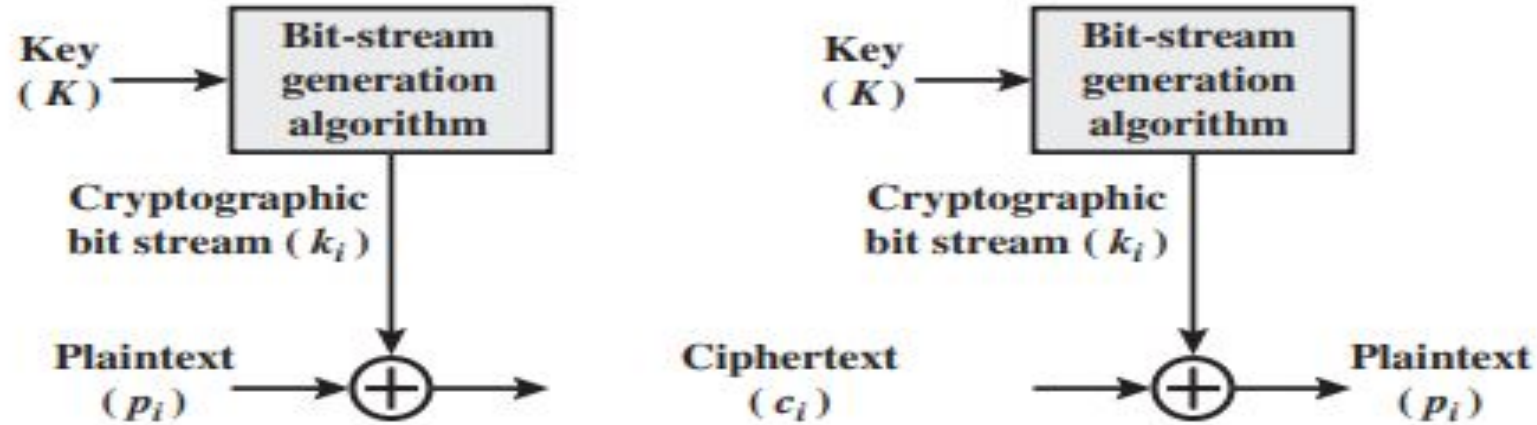
UNIT -2

UNIT II BLOCK CIPHERS & STREAM CIPHERS MECHANISMS

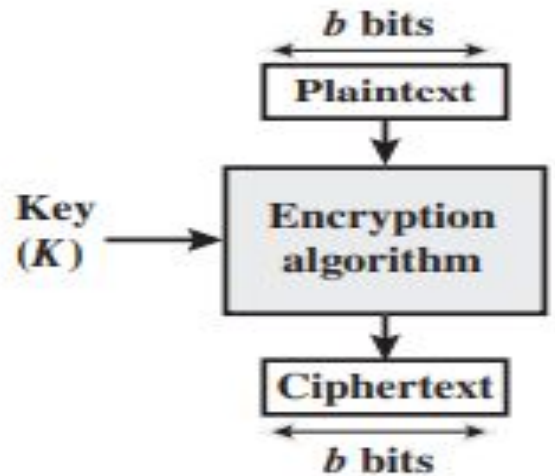
Block Cipher Mechanisms: DES, Block cipher modes of operation. Introduction to Finite Fields: Groups, Rings and Fields, Modular Arithmetic, Euclid's Algorithm, Finite Fields, Advanced Encryption Standard, Blowfish. Stream Cipher Mechanisms: RC4Stream Cipher, Key Distribution -Diffie Hellman Key Exchange, Psuedo Random Number Generation.

STREAM CIPHERS & BLOCK CIPHERS

- A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time.
- Examples of classical stream ciphers are the autokeyed Vigenere cipher and the Vernam cipher
- In this approach (Figure 2.1a), the bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong. Now, the two users need only share the generating key, and each can produce the keystream
- A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.
- Typically, a block size of 64 or 128 bits is used.
- As with a stream cipher, the two users share a symmetric encryption key (Figure 2.1b).



(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

Fig 2.1 a) & b) Stream Cipher and Block Cipher

Feistel Network

- Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers
- The essence of the approach is to develop a block cipher wit 2^k ey length of k bits and a block length of n bi $2^n!$ llowing a total of possible transformations, rather than the transformations available with the ideal block cipher
- Used in DES, IDEA, RC5 (Rivest's Cipher n. 5), and many other block ciphers
- Not used in AES

Feistel proposed the use of a cipher that alternates **substitutions and permutations,** where these terms are defined as follows:

- **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding 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

Feistel's is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates **confusion and diffusion** functions

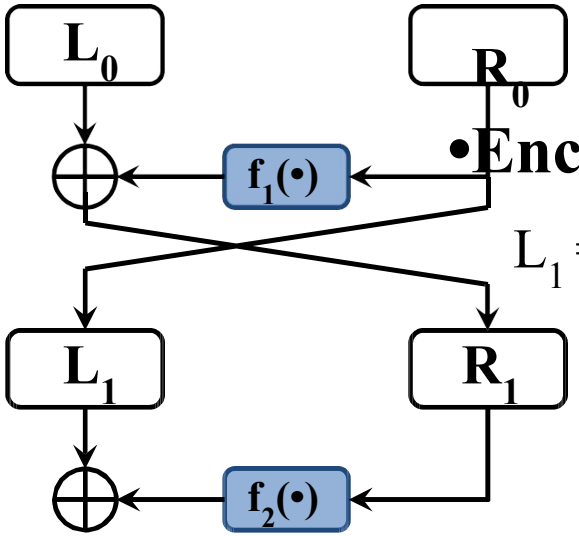
- In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext.
- This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits.

An example of diffusion is to encrypt a message $M = m_1, m_2, m_3, \ldots$ of characters with an averaging operation:

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

On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key

Feistel Network



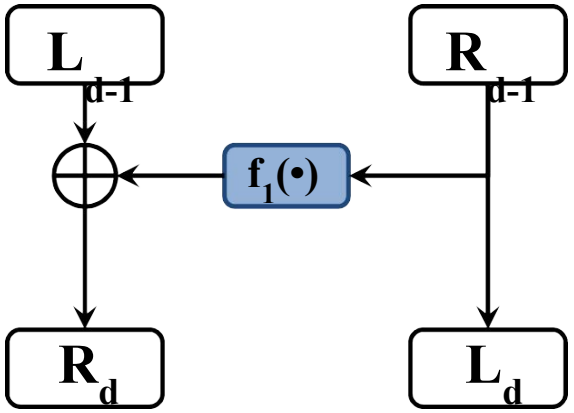
- **Encryption**:

$L_1 = R_0$ $\quad R_1 = L_0 \oplus f_1(R_0)$

$L_2 = R_1$ $\quad R_2 = L_1 \oplus f_2(R_1)$

……

$L_d = R_{d-1}$ $\quad R_d = L_{d-1} \oplus f_d(R_{d-1})$



- **Decryption**:

$R_{d-1} = L_d$ $\quad L_{d-1} = R_d \oplus f_d(L_d)$

……

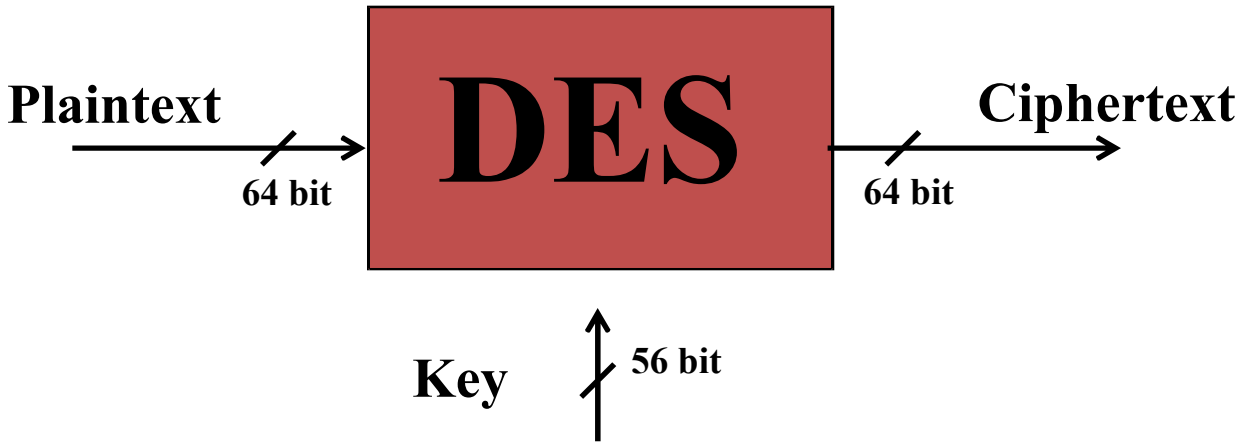
$R_0 = L_1$; $\quad L_0 = R_1 \oplus f_1(L_1)$

Data Encryption Standard (DES)

- The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46)
- The algorithm itself is referred to as the Data Encryption Algorithm (DEA)
- For DES, data are encrypted in **64-bit blocks using a 56-bit key**
- The algorithm transforms 64-bit input in a series of steps into a 64-bit output
- The same steps, with the same key, are used to reverse the encryption

DES Features

- Features:
  - Block size = 64 bits
  - Key size = 56 bits (in reality, 64 bits, but 8 are used as parity-check bits for error control, see next slide)
  - Number of rounds = 16
  - 16 intermediary keys, each 48 bits



DES Encryption

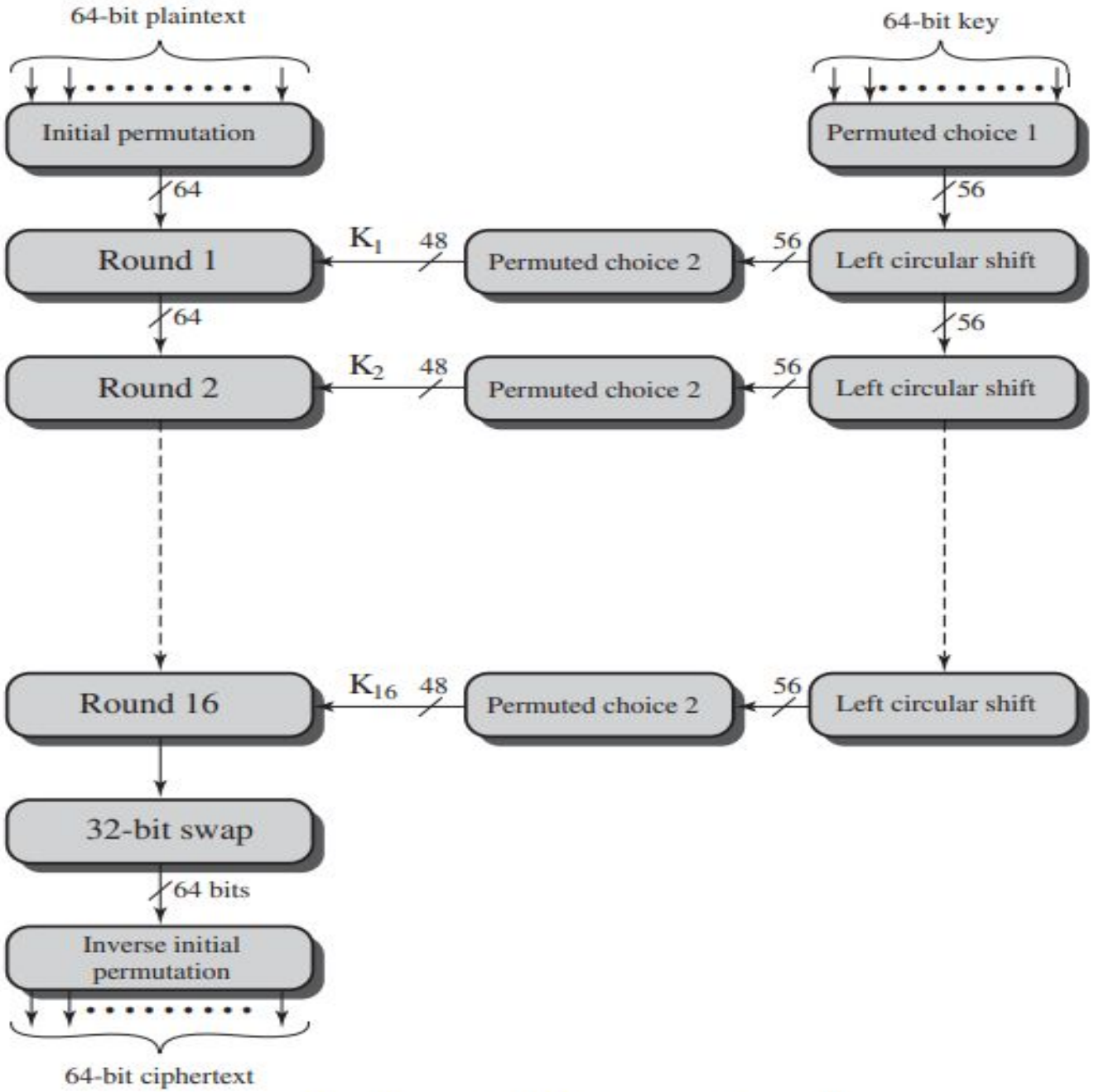
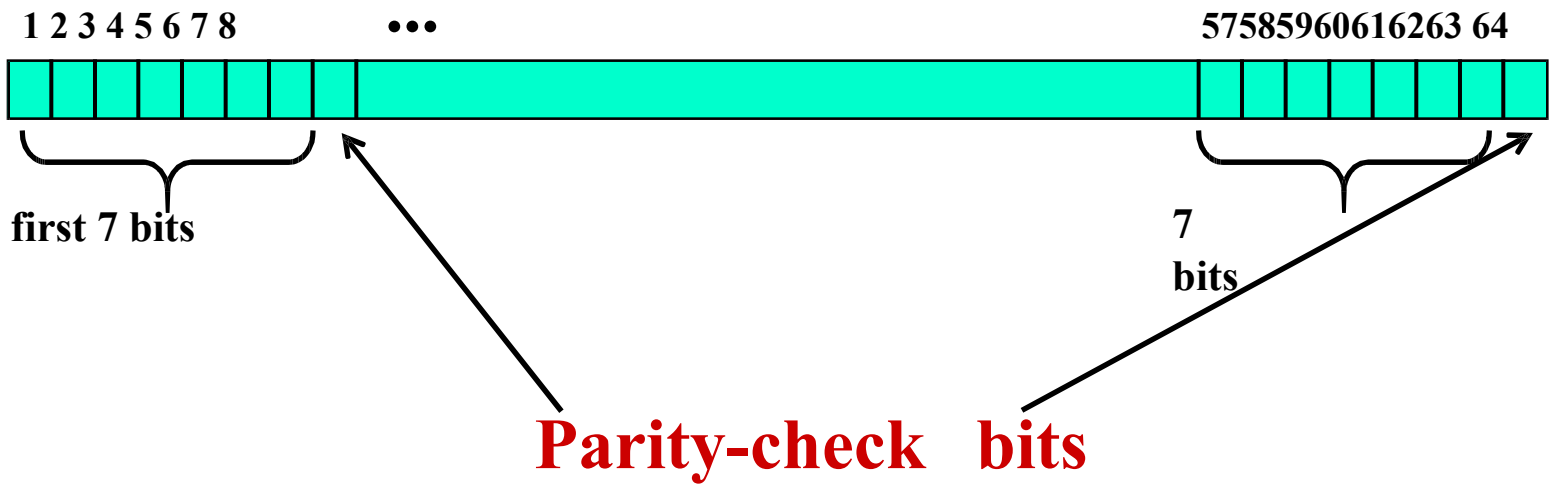


Figure General Depiction of DES Encryption Algorithm

Key length in DES

- In the DES specification, the key length is 64 bit:
- 8 bytes; in each byte, the 8th bit is a parity-check bit



Each parity-check bit is the XOR of the previous 7 bits

(a) Initial Permutation (IP)

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

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

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

(c) Expansion Permutation (E)

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

(d) Permutation 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

DETAILS OF SINGLE ROUND

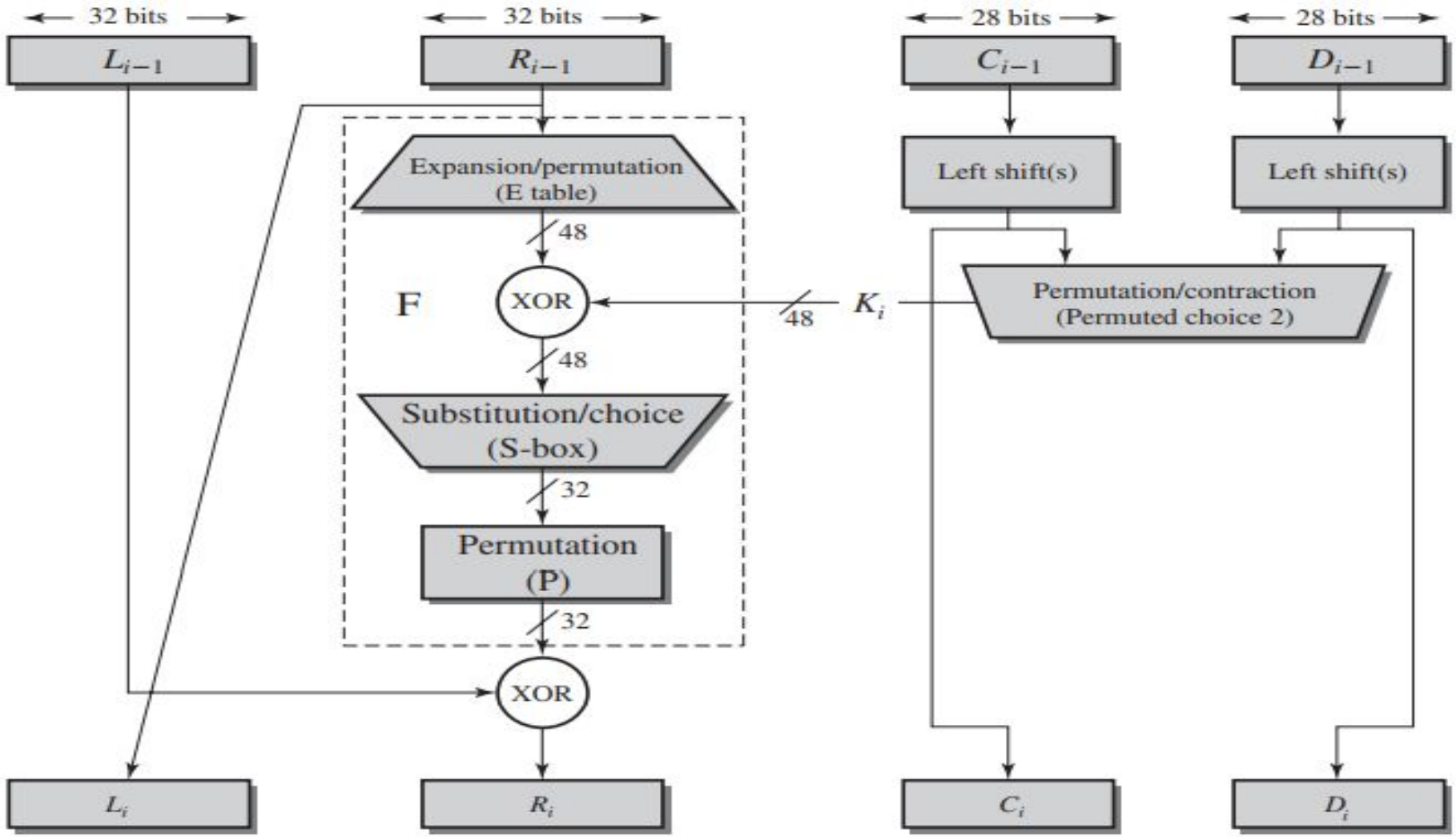


Figure Single Round of DES Algorithm

- Figure shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram.
- The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right)
- As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

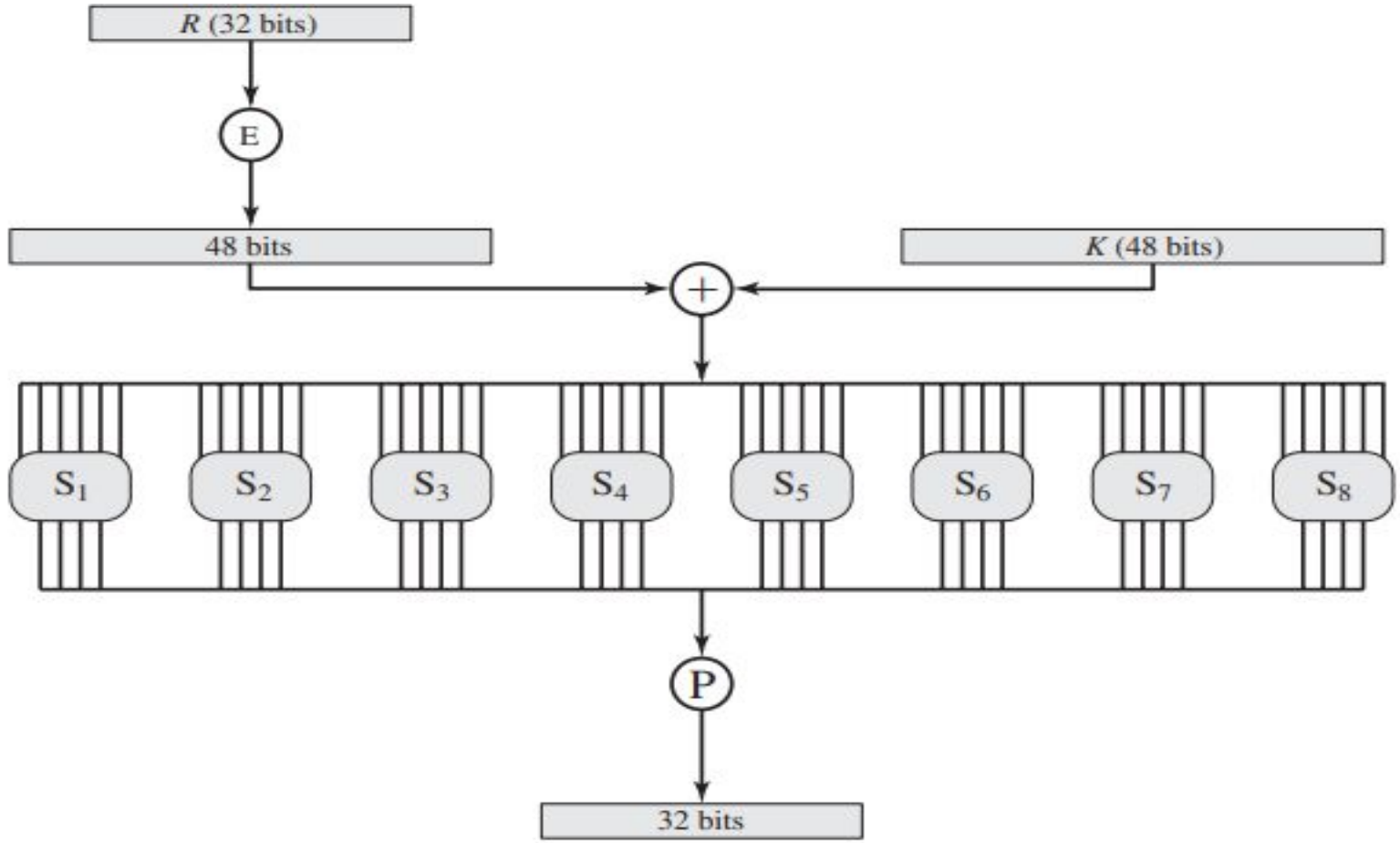


Figure Calculation of F(R, K)

Table 3.3 Definition of DES S-Boxes

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3.4 DES Key Schedule Calculation

(a) Input Key

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

(b) Permuted Choice One (PC-1)

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

(c) Permuted Choice Two (PC-2)

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

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES Decryption

- A change in one bit of the plaintext or one bit of the key should produce a change in many bits of the cipher text is known as the **Avalanche Effect**

- The statistical structure of the plaintext is dissipated into long-range statistics of the cipher text is called **Diffusion**

Strength of DES

- **The Use of 56-Bit Keys - With a key length of 56 bits, there are 256** possible keys, which is approximately 7.2 * 1016 keys. Thus a brute-force attack appears impractical.
- DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose “DES cracker” machine that was built for less than $250,000.

The attack took less than three days.

- The Nature of the DES Algorithm - cryptanalysis is possible by exploiting the characteristics of the DES algorithm.
- A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts.
- A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

Block Cipher Modes of operation

- Electronic Codebook (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

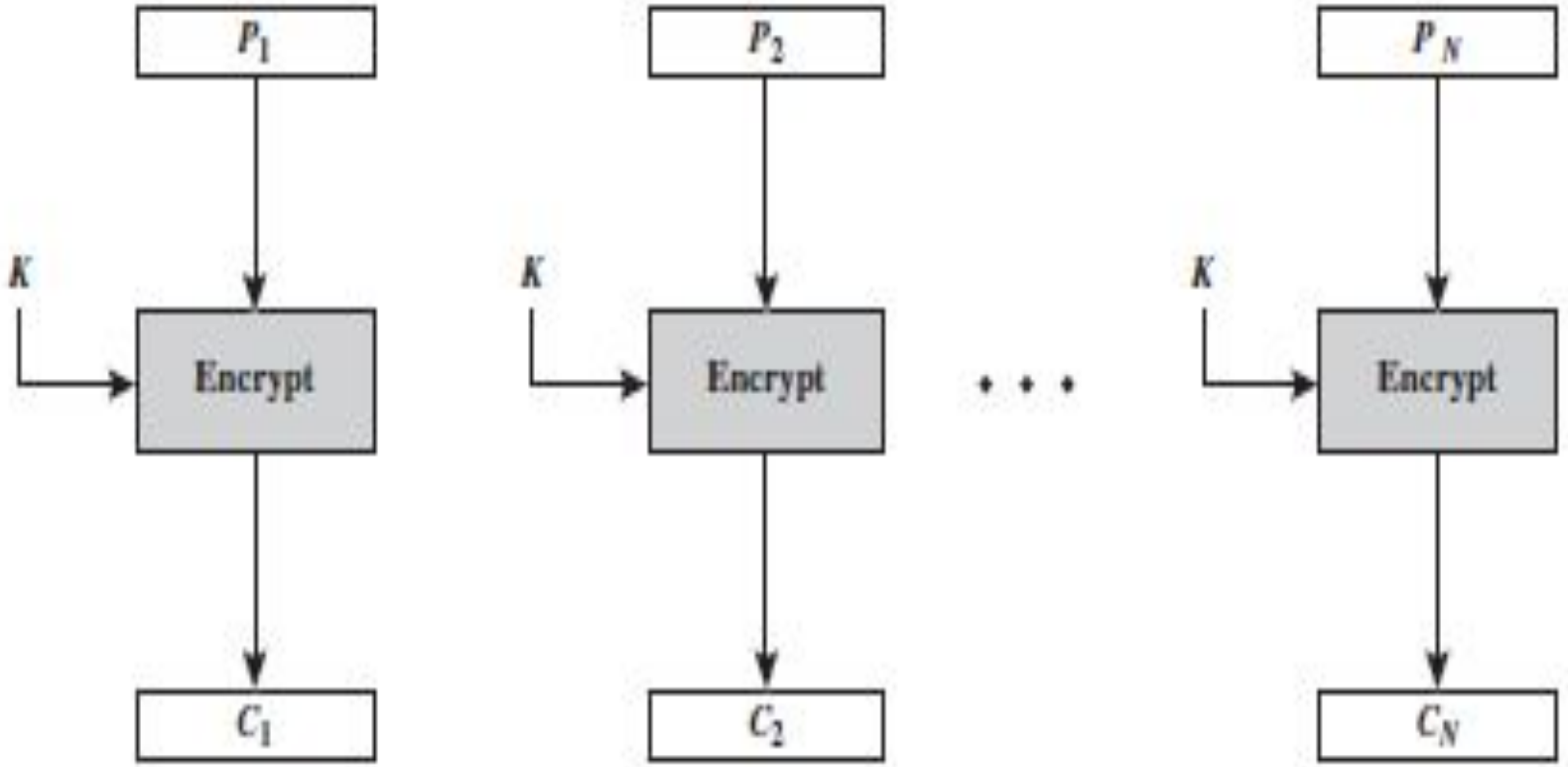
When we use block cipher modes of operation?

- Block cipher only allow to encrypt entire blocks.
- What if our message is longer/shorter than the block size?
- When message is longer/shorter than the block size, we use modes of operations.
- Algorithms that exploit a block cipher to provide a service (e.g. confidentiality).

Block cipher mode of operation – Electronic Codebook (ECB)

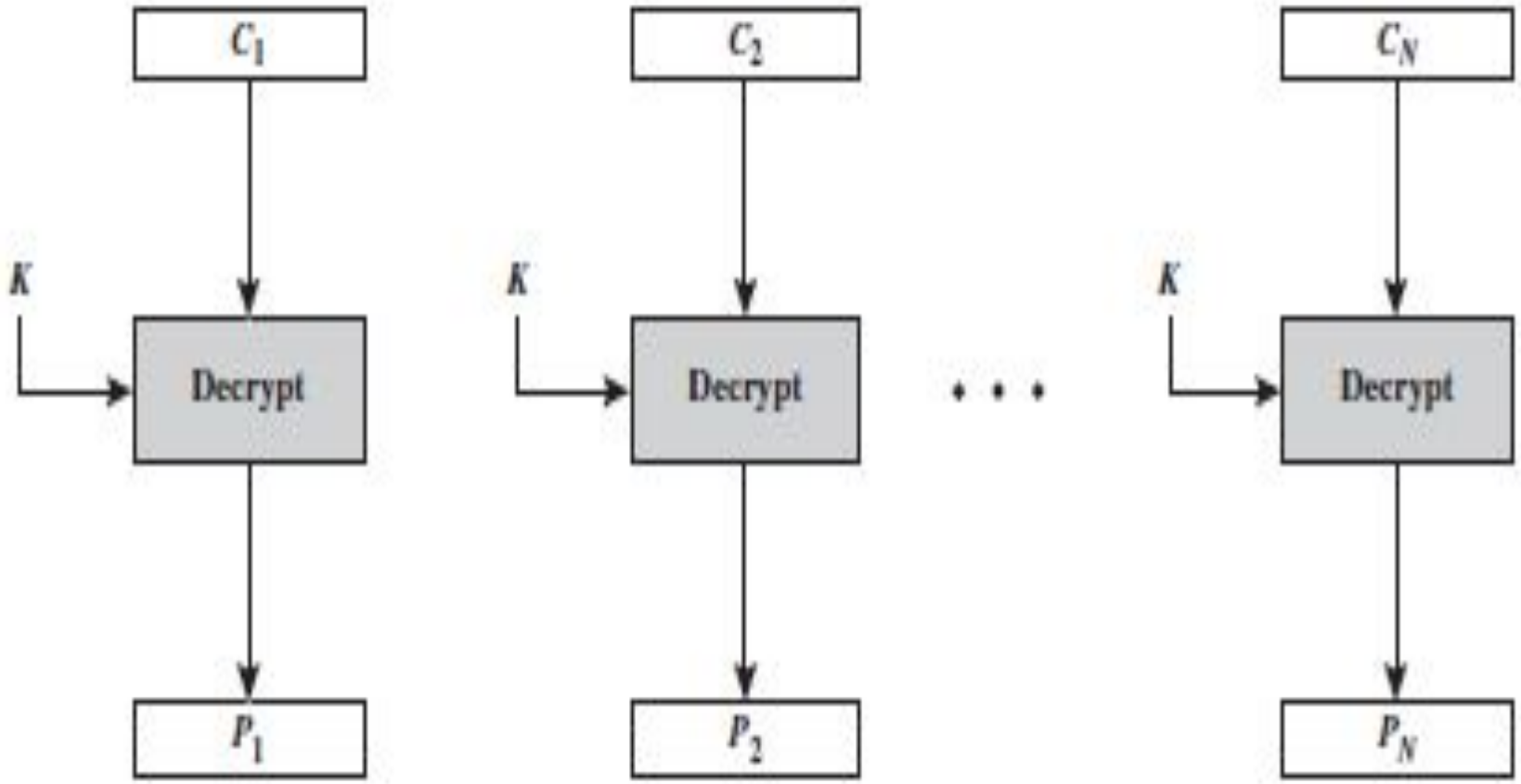
- The simplest mode is the **electronic codebook (ECB) mode, in which plaintext** is handled one block at a time and each block of plaintext is encrypted using the same key (Figure)
- The term *codebook is used because, for a given key, there is* a unique ciphertext for every *b*-bit block of plaintext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

Electronic Codebook (ECB) - Encryption



(a) Encryption

Electronic Codebook (ECB) - Decryption



(b) Decryption

- 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.
- For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext–ciphertext pairs to work with.

Cipher Block Chaining Mode(CBC)

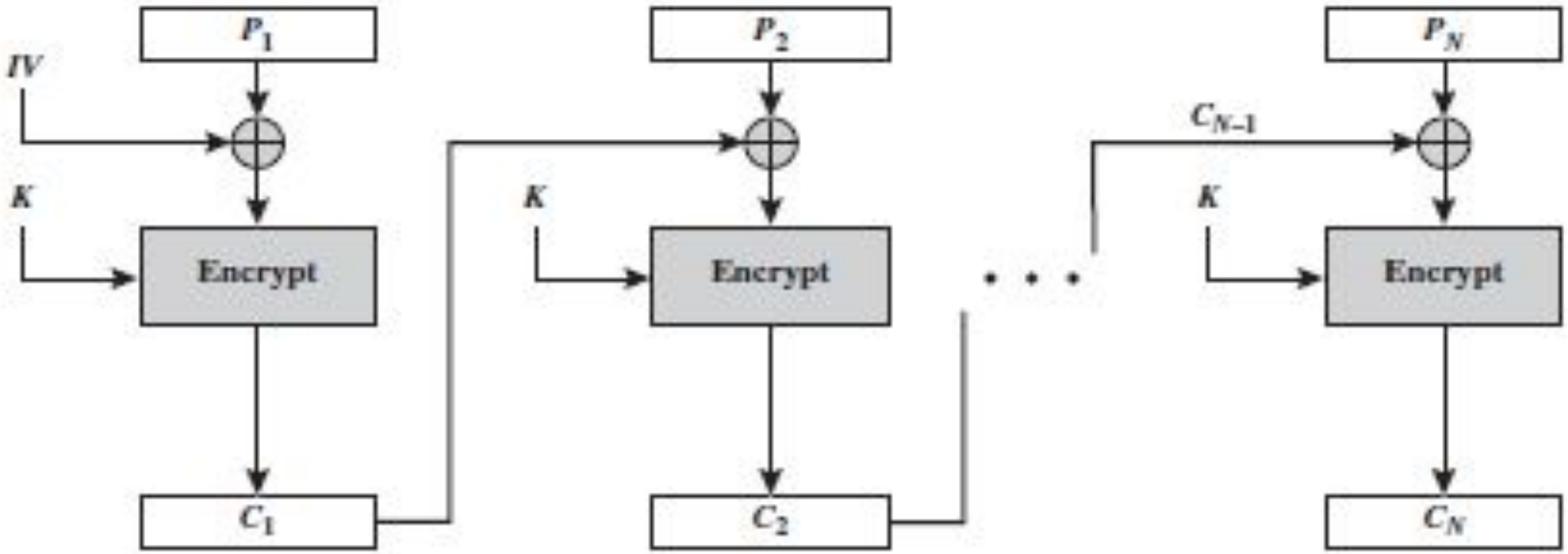
- To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks
- A simple way to satisfy this requirement is the **cipher block chaining (CBC) mode** (Figure)
- In this scheme, **the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block**
- In effect, we have chained together the processing of the sequence of plaintext blocks.
- The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block.
- Therefore, repeating patterns of bits are not exposed.
- As with the ECB mode, the CBC mode requires that the last block be padded to a full bits if it is a partial block.

- **To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext.**
- To see that this works, we can write

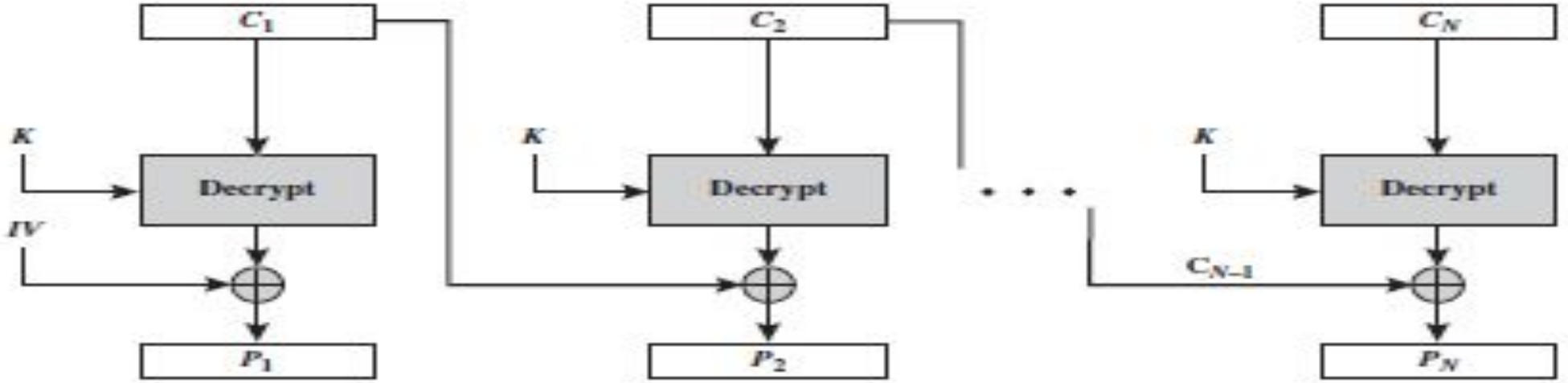
$$C_j = \mathrm{E}(K, [C_{j-1} \oplus P_j])$$

Then

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



(a) Encryption



(b) Decryption

- For decryption, each cipher block is passed through the decryption algorithm.
- The result is XORed with the preceding ciphertext block to produce the plaintext block.
- **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 that same size as the cipher block.** We can define CBC mode as

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

- The IV must be known to both the sender and receiver but be unpredictable by a third party.
- In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV.
- For maximum security, the IV should be protected against unauthorized changes.
- This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext

What is initialization vector?

- An initialization vector (IV) or starting variable is a block of bits that is used by several modes to randomize the encryption and hence to produce distinct cipher texts even if the same plain text is encrypted multiple times, without the need for a slower re-keying process
- An initialization vector has different security requirements than a key, so the IV usually does not need to be secret
- However, in most cases, it is important that an initialization vector is never reused under the same key

What is initialization vector? (continue…)

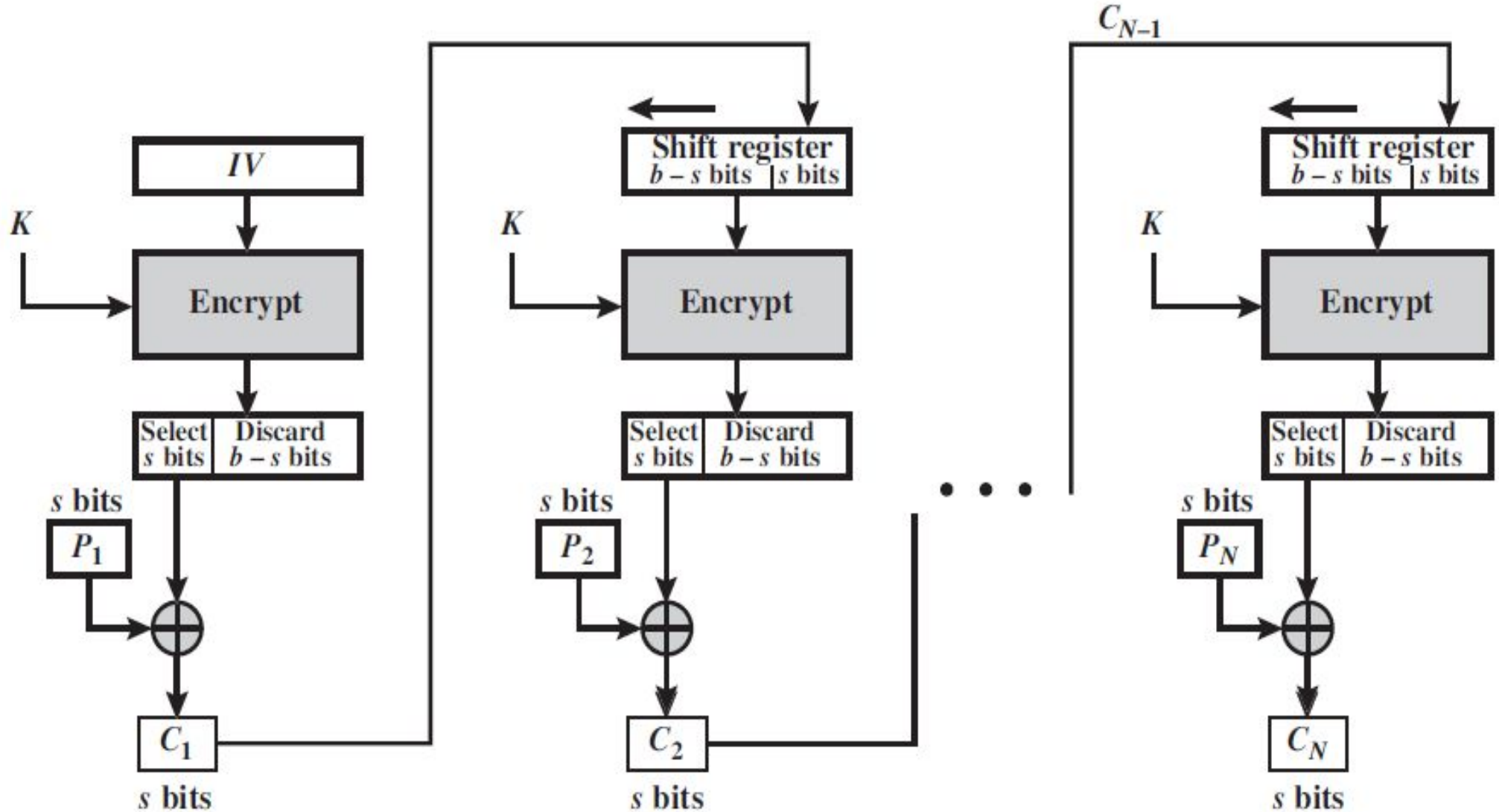
- For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages
- For OFB and CTR, reusing an IV completely destroys security. This can be seen because both modes effectively create a bit stream that is XORed with the plaintext, and this bit stream is dependent on the password and IV only. Reusing a bit stream destroys security.
- In CBC mode, the IV must, in addition, be unpredictable at encryption time; in particular, the (previously) common practice of re-using the last cipher text block of a message as the IV for the next message is insecure.

- So long as it is unpredictable, the specific choice of IV is unimportant
- Sp800-38a recommends two possible methods: The first method is to apply the encryption function, under the same key that is used for the encryption of the plaintext, to a **nonce**
- **The nonce** must be a data block that is unique to each execution of the encryption operation. For example, the nonce may be a counter, a timestamp, or a message number.
- The second method is to generate a random data block using a random number generator
- In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than bits.

Cipher Feedback Mode(CFB)

- ECB and CBC modes encrypt and decrypt blocks of the message
- Block size n is predetermine by the underlying cipher ;

  for example, for DES n = 64 for AES n =128
- In some situations, we need use DES or AES as secure cipher , but the plain text or cipher text block size are to be smaller
- For example, to encrypt and decrypt 8-bit characters , you would not want to use one of the traditional cipher like Caesar cipher
- The solution is to use DES or AES in cipher feedback mode



(a) Encryption

- Figure depicts the CFB scheme for Encryption. 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*
- First, consider **encryption**. The input to the encryption function is a *b*-bit shift register that is initially set to some initialization vector (IV)
- The leftmost (most significant) *s* bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C1, which is then transmitted
- In addition, the contents of the shift register are shifted left by *s* bits, and C1 is placed in the rightmost (least significant) *s* bits of the shift register.
- This process continues until all plaintext units have been encrypted

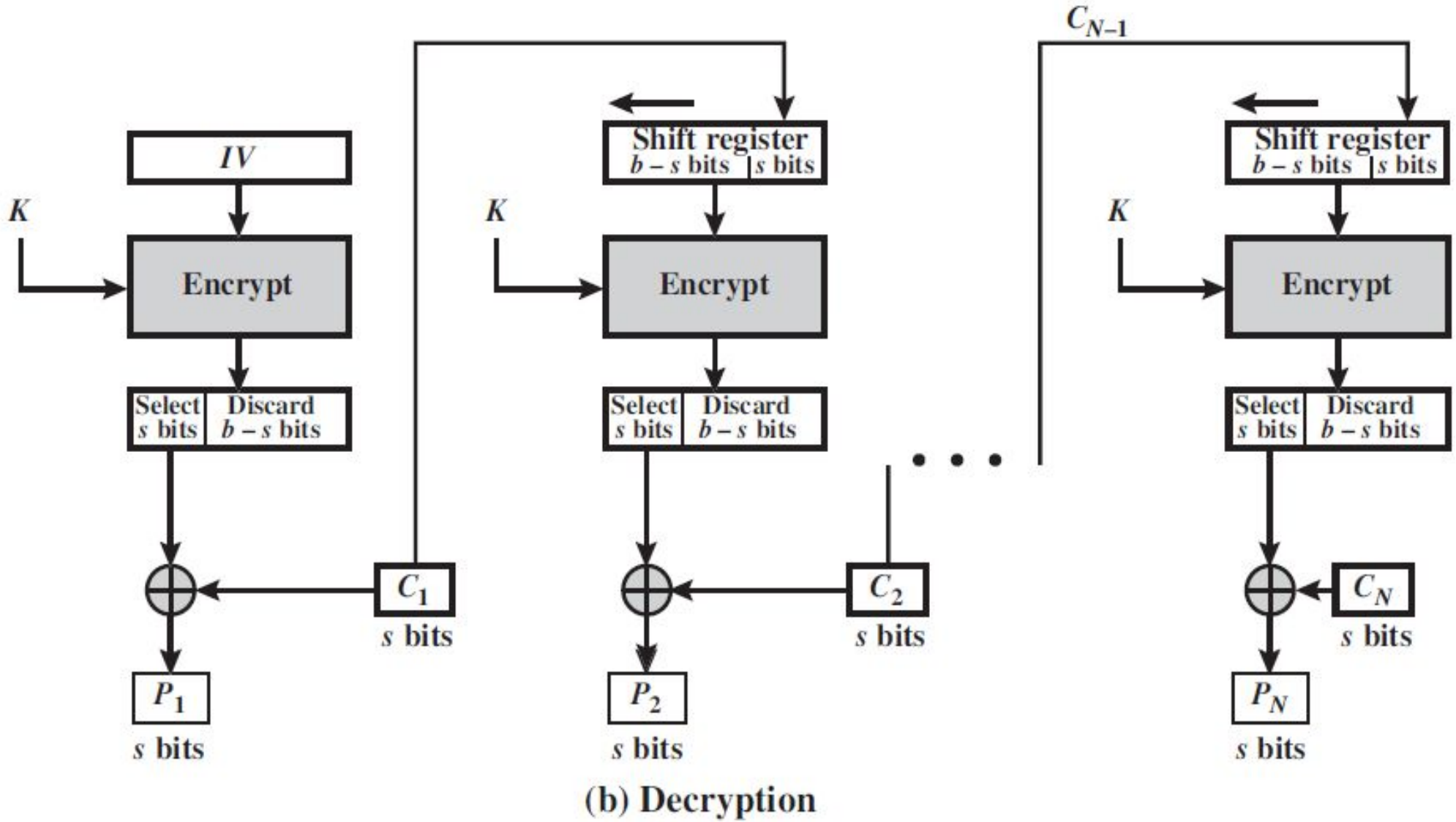


Figure *s*-bit Cipher Feedback (CFB) Mode

- For **decryption**, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit
- **Note that it is the *encryption function that is used, not the decryption function*.**
- Let MSB*s(X)* be defined as the most significant *s* bits of *X* .
- Then

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

Therefore, by rearranging terms:

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

The same reasoning holds for subsequent steps in the process.

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

- 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

Cipher feedback mode Security issues

- Just like CBC , patterns at the block level are not preserved
- More than one message can be encrypted with the same key , but the value of the IV should be changed for each message.
- This means that sender needs to use a different IV each time sender sends a message
- Attacker can add some cipher text block to the end of the cipher text stream.

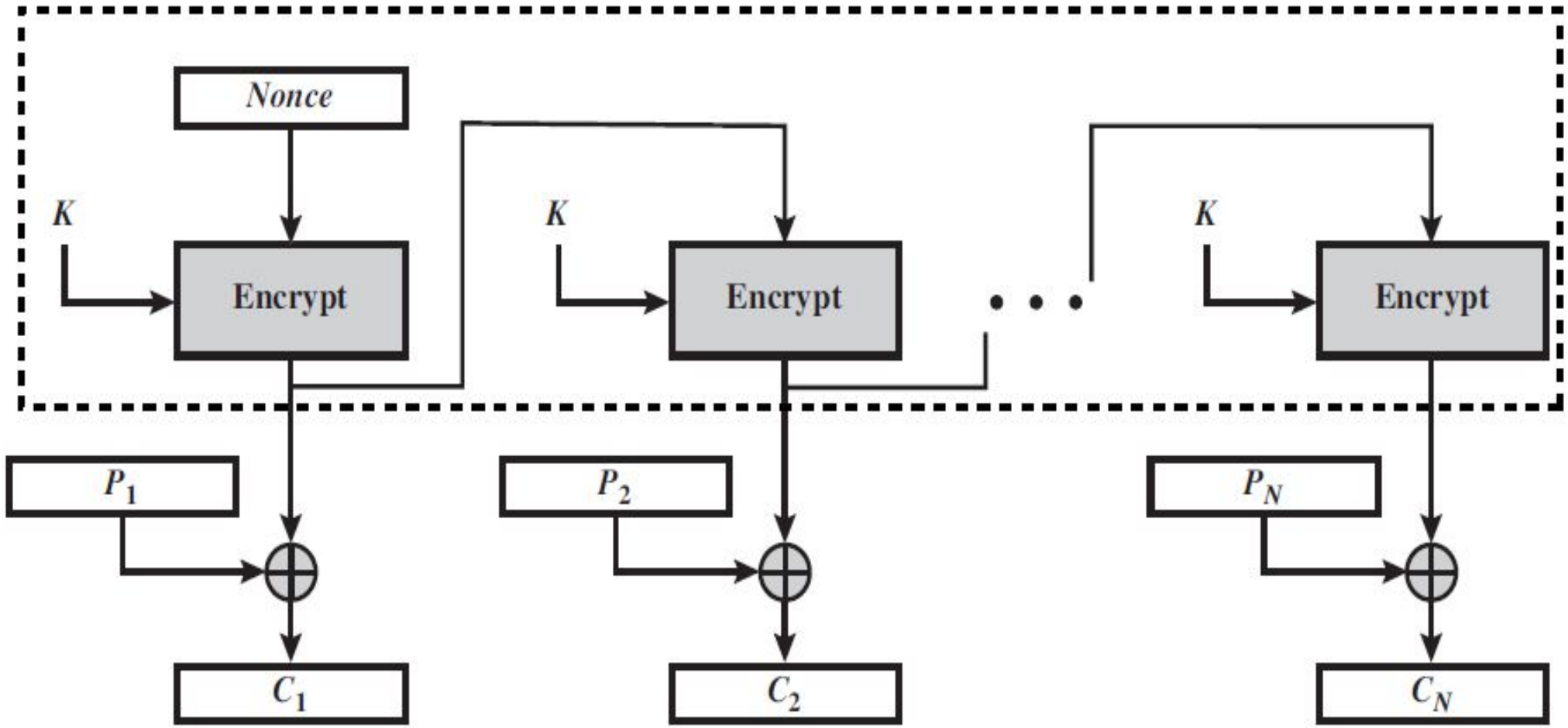
Output Feedback Mode (OFB)

- The **output feedback (OFB) mode is similar in structure to that of CFB.**
- **As can be** seen in Figure, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB, the ciphertext unit is fed back to the shift register
- The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, not on an *s* -bit subset
- Encryption can be expressed as

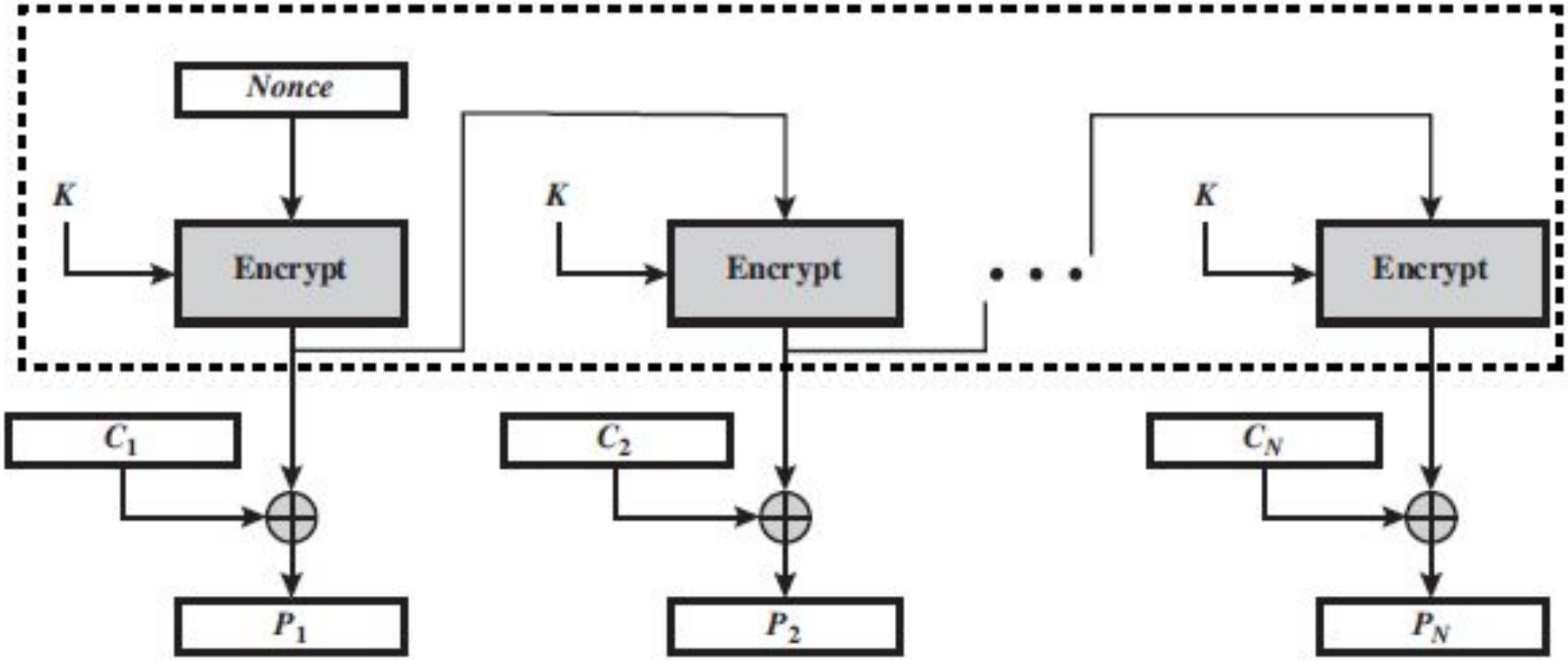
$$C_j = P_j \oplus \mathrm{E}(K, [C_{j-i} \oplus P_{j-1}])$$

By rearranging terms, we can demonstrate that decryption works.

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



(a) Encryption



(b) Decryption

Figure Output Feedback (OFB) Mode

- 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
- For a given key and IV, the stream of output bits used to XOR with the stream of plaintext bits is fixed
- 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
- Consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintex
- Thus, controlled changes to the recovered plaintext can be made
- This may make it possible for an opponent, by making the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code.

- OFB has the structure of a typical stream cipher, because the cipher generates a stream of bits as a function of an initial value and a key, and that stream of bits is XORed with the plaintext bits
- The generated stream that is XORed with the plaintext is itself independent of the plaintext; this is highlighted by dashed boxes in Figure
- One distinction from the stream ciphers is that OFB encrypts plaintext a full block at a time, where typically a block is 64 or 128 bits
- Many stream ciphers encrypt one byte at a time

Cipher output feedback mode

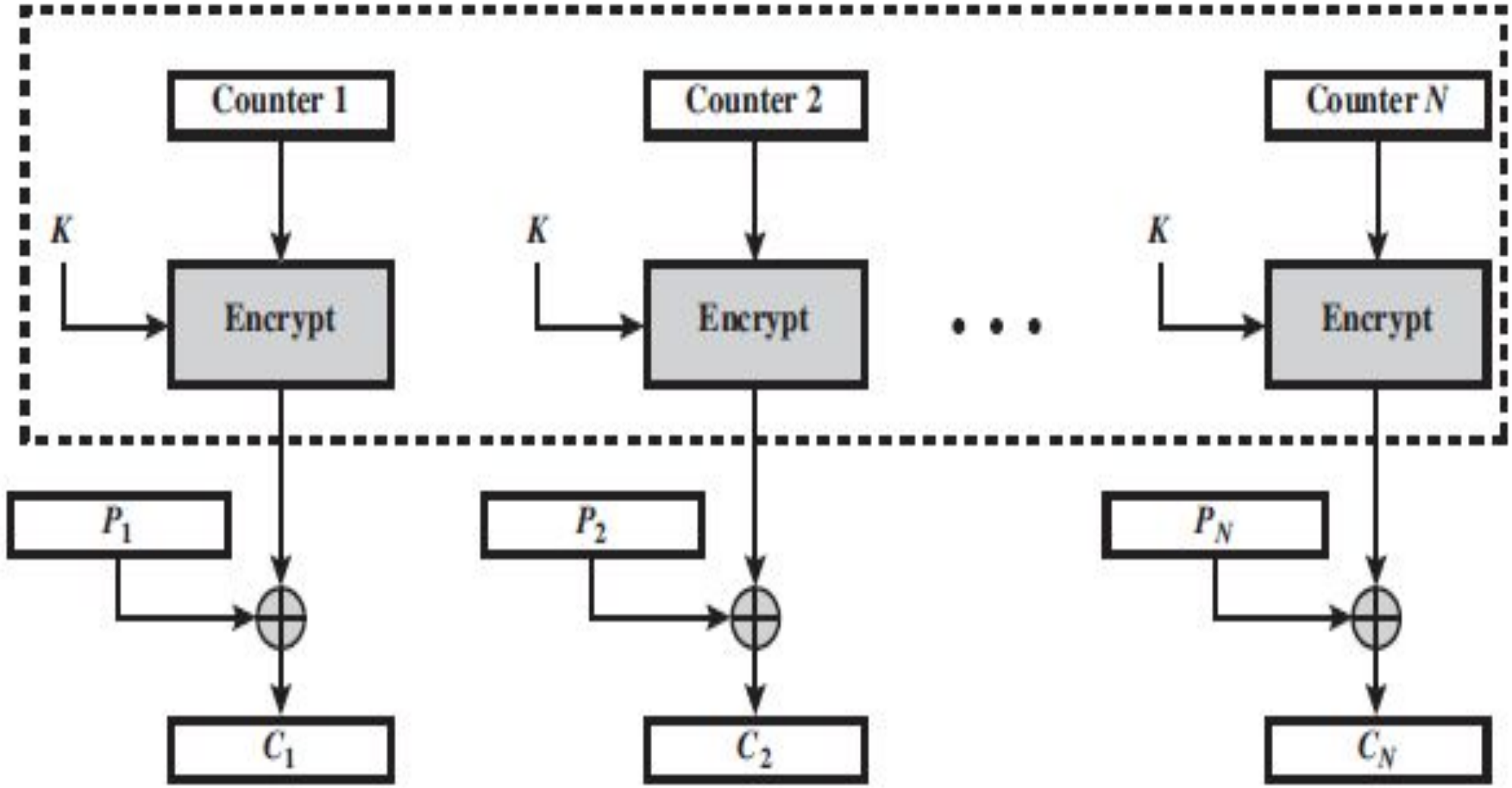
- Output feedback mode is very similar to CFB mode , with one difference: each bit in the cipher text is independent of the previous bit or bits
- This avoids error propagation
- If an error occur in transmission , it does not affect the bits that follow
- Note that , like cipher feedback mode , both the sender and the receiver use the encryption algorithm

Cipher output feedback mode Security issues

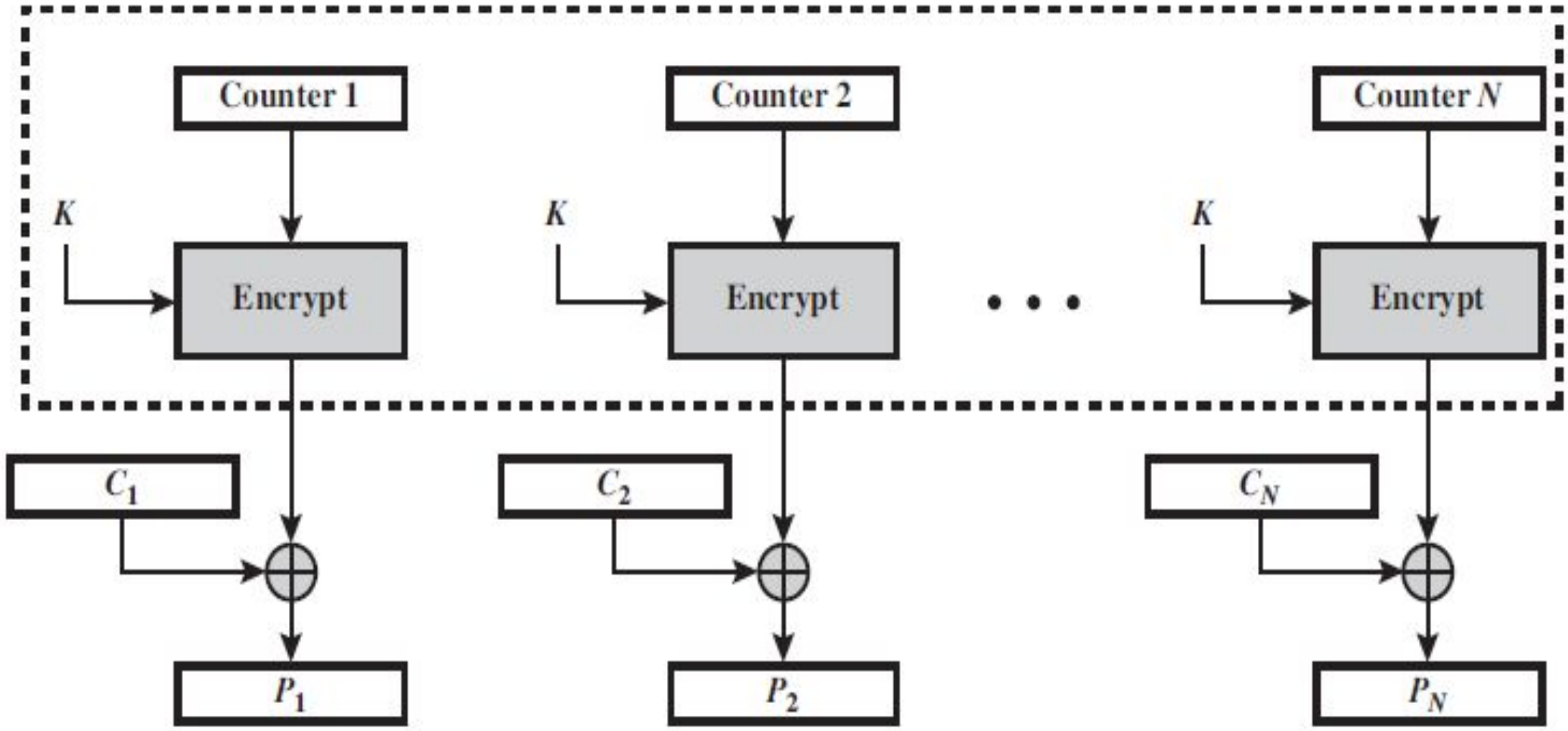
- Just like CBC , patterns at the block level are not preserved.
- Any change in the cipher text affects the plain text encrypted at the receiver side.

Counter Mode(CTR)

- Although interest in the **counter (CTR) mode has increased recently with applications** to ATM (asynchronous transfer mode) network security and IP sec (IP security), this mode was proposed early on (e.g., [DIFF79])
- Figure 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



(a) Encryption



(b) Decryption

Figure Counter (CTR) Mode

- 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
- Unlike the ECB, CBC, and CFB modes, we do not need to use padding because of the structure of the CTR mode
- A counter value is used multiple times, then the confidentiality of all of the plaintext blocks corresponding to that counter value may be compromised
- In particular, if any plaintext block that is encrypted using a given counter value is known, then the output of the encryption function can be determined easily from the associated ciphertext block

- This output allows any other plaintext blocks that are encrypted using the same counter value to be easily recovered from their associated ciphertext blocks
- One way to ensure the uniqueness of counter values is to continue to increment the counter value by 1 across messages
- That is, the first counter value of the each message is one more than the last counter value of the preceding message
- **Advantages of CTR mode:**

  Hardware efficiency, Software efficiency, Preprocessing, Random access, Provable security and Simplicity

Counter Mode Extra points

- In the counter mode , there is no feedbck
- The pseudo randomness in the key streams achieved using a counter
- An n bit counter is initialized to a predetermined value(IV) and incremented based on a predefined rule(mod 2^n)
- To provide a better randomness , the increment value can depend on the block numbers to be incremented
- The plain text and cipher block text block have same block size as the underlying cipher.
- Both encryption and decryption can be performed fully in parallel on multiple blocks ,provides true random access to cipher text blocks

Introduction to Finite Fields: Groups, Rings and Fields

Groups

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

Groups

- (A4) Inverse element: For each a in G, there is an element a′ in G such that $a \cdot a' = a' \cdot a = e$.
- A group is said to be abelian if it satisfies the following additional condition:

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

Rings

- A ring R, sometimes denoted by {R, +, * }, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in R the following axioms are obeyed.
- (M1) Closure under multiplication: If a and b belong to R, then ab is also in R.
- (M2) Associativity of multiplication: a(bc) = (ab)c for all a, b, c in R.
- (M3) Distributive laws: a(b + c) = ab + ac for all a, b, c in R.
  (a + b)c = ac + bc for all a, b, c in R.

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

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

Rings

An **integral domain, which is a commutative ring that obeys the following axioms:**

(M5) Multiplicative identity: There is an element 1 in R such that

$a1 = 1a = a$ for all a in R.

(M6) No zero divisors: If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Fields

A field F denoted by {F, +, * }, is a set of elements with two binary operations, called addition and multiplication, such that the following conditions hold:

- A **field F, sometimes denoted by {F, +, x}, is a set of elements with two binary operations, called** addition and multiplication, such that for all a, b, c in F the following axioms are obeyed:
- **(A1 - M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.**
- **(M7) Multiplicative inverse: For each a in F, except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.**

Modular Arithmetic

The Modulus

If a is an integer and n is a positive integer, we define a mod n to be the remainder when a is divided by n. The integer n is called the **modulus**. Thus, for any integer a,

we can rewrite Equation as follows:

$$a = qn + r \qquad 0 < r < n;\ q = [a/n] \qquad a = [a/n] * n + (a \bmod n)$$

11 mod 7 = 4; - 11 mod 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 as $a = b \pmod{n}$.[2]

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

Properties of Congruences

Congruences have the following properties:

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

To demonstrate the first point, if $n \mid (a - b)$, 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) = (b mod n).

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

Modular Arithmetic Operations

Modular arithmetic exhibits the following properties:

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

We demonstrate the first property. Define $(a \bmod n) = r_a$ and $(b \bmod n) = r_b$. Then we can write $a = r_a + jn$ for some integer j and $b = r_b + kn$ for some integer k. Then

$$\begin{aligned}(a + b) \bmod n &= (r_a + jn + r_b + kn) \bmod n \\ &= (r_a + r_b + (k + j)n) \bmod n \\ &= (r_a + r_b) \bmod n \\ &= [(a \bmod n) + (b \bmod n)] \bmod n\end{aligned}$$

The remaining properties are proven as easily. Here are examples of the three properties:

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

Properties of Modular Arithmetic

Define the set Z_n as the set of nonnegative integers less than:

$Z_n = \{0, 1, 2, \ldots\ldots (n - 1)\}$

This is referred to as the **set of residues**, or **residue classes** (mod n).

To be more precise, each integer in Z_n represents a residue class.

The residue classes (mod n) are labelled as [0], [1], [2],……, [n - 1]

where

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

Table 4.2 Arithmetic 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

(a) Addition 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

(b) Multiplication 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

(c) Additive and multiplicative inverses modulo 8

The residue classes (mod 4) are
[0] = {c, - 16, - 12, - 8, - 4, 0, 4, 8, 12, 16, c}
[1] = {c, - 15, - 11, - 7, - 3, 1, 5, 9, 13, 17, c}
[2] = {c, - 14, - 10, - 6, - 2, 2, 6, 10, 14, 18, c}
[3] = {c, - 13, - 9, - 5, - 1, 3, 7, 11, 15, 19, c}

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

Table 4.3 Properties of Modular Arithmetic for Integers in Z_n

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

If we perform modular arithmetic within Z_n, the properties shown in Table 4.3 hold for integers in Z_n. We show in the next section that this implies that Z_n is a commutative ring with a multiplicative identity element.

There is one peculiarity of modular arithmetic that sets it apart from ordinary arithmetic. First, observe that (as in ordinary arithmetic) we can write the following:

$$\textbf{if } (a + b) = (a + c) \pmod{n} \quad \textbf{then} \quad b = c \pmod{n} \tag{4.4}$$

$$(5 + 23) = (5 + 7) \pmod{8};\ 23 = 7 \pmod{8}$$

Equation (4.4) is consistent with the existence of an additive inverse. Adding the additive inverse of a to both sides of Equation (4.4), we have

$$\begin{aligned} ((-a) + a + b) &= ((-a) + a + c) \pmod{n} \\ b &= c \pmod{n} \end{aligned}$$

However, the following statement is true only with the attached condition:

$$\textbf{if } (a \times b) = (a \times c) \pmod{n} \textbf{ then } b = c \pmod{n} \quad \textbf{if } a \text{ is relatively prime to } n \tag{4.5}$$

Recall that two integers are **relatively prime** if their only common positive integer factor is 1. Similar to the case of Equation (4.4), we can say that Equation (4.5) is

consistent with the existence of a multiplicative inverse. Applying the multiplicative inverse of a to both sides of Equation (4.5), we have

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

To see this, consider an example in which the condition of Equation (4.5) does not hold. The integers 6 and 8 are not relatively prime, since they have the common factor 2. We have the following:

$$6 \times 3 = 18 = 2 \pmod{8}$$
$$6 \times 7 = 42 = 2 \pmod{8}$$

Yet $3 \neq 7 \pmod{8}$.

The reason for this strange result is that for any general modulus n, a multiplier a that is applied in turn to the integers 0 through $(n - 1)$ will fail to produce a complete set of residues if a and n have any factors in common.

With $a = 6$ and $n = 8$,

Z_8	0	1	2	3	4	5	6	7
Multiply by 6	0	6	12	18	24	30	36	42
Residues	0	6	4	2	0	6	4	2

Because we do not have a complete set of residues when multiplying by 6, more than one integer in Z_8 maps into the same residue. Specifically, $6 \times 0 \bmod 8 = 6 \times 4 \bmod 8$; $6 \times 1 \bmod 8 = 6 \times 5 \bmod 8$; and so on. Because this is a many-to-one mapping, there is not a unique inverse to the multiply operation.

However, if we take $a = 5$ and $n = 8$, whose only common factor is 1,

Z_8	0	1	2	3	4	5	6	7
Multiply by 5	0	5	10	15	20	25	30	35
Residues	0	5	2	7	4	1	6	3

The line of residues contains all the integers in Z_8, in a different order.

In general, an integer has a multiplicative inverse in Z_n if that integer is relatively prime to n. Table 4.2c shows that the integers 1, 3, 5, and 7 have a multiplicative inverse in Z_8; but 2, 4, and 6 do not.

Two numbers are called Additive inverses if their sum is zero.
Two numbers are called Multiplicative inverses if their product is one.

Euclid's algorithm

- One of the basic techniques of number theory is the Euclidean algorithm, which is a simple procedure for determining the **greatest common divisor** of two positive integers
- **Two integers are relatively prime if and only if their only common positive integer factor is 1**
- gcd(a, b) denotes the greatest common divisor of a and b
- The greatest common divisor of a and b is the largest integer that divides both a and b

Greatest Common Divisor

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

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

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

An equivalent definition is the following:

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

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

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

Also, because all nonzero integers divide 0, we have $\gcd(a, 0) = |a|$.

We stated that two integers a and b are relatively prime if their only common positive integer factor is 1. This is equivalent to saying that a and b are relatively prime if $\gcd(a, b) = 1$.

8 and 15 are relatively prime because the positive divisors of 8 are 1, 2, 4, and 8, and the positive divisors of 15 are 1, 3, 5, and 15. So 1 is the only integer on both lists.

Euclid's algorithm

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

gcd(30,7) =gcd (7 , 30 mod 7)

=gcd(7,2) = gcd(2,7 mod 2)

= gcd(2,1) =gcd(1,2 mod 1)

= gcd(1,0) = 1

gcd(18,12) = gcd (12,18 mod 12) =gcd(12,6)

=gcd(6,12mod6)=gcd(6,0)=6

gcd(11,10) = gcd(10,11 mod 10) = gcd(10,1),

= gcd(1,10 mod 1)=gcd(1,0)=1

Find gcd(1970,1066) using Euclid's algorithm

Ans: GCD(1970,1066) = 2

gcd(a, b)

- Things to keep in mind:

1. gcd(a,b) = gcd(b,a)
2. gcd(a,b) = gcd(-a,b) = gcd(a,-b) = gcd(-a,-b)
3. gcd(a,b) = gcd(b, *remainder of* (a/b))
4. gcd(a,**0**) = **|**a**|**

Finite fields

- Finite Fields, also known as Galois Fields, are cornerstones for understanding any cryptography
- A field can be defined as a set of numbers that we can add, subtract, multiply and divide together and only ever end up with a result that exists in our set of numbers
- Galois showed that for a field to be finite, the number of elements should be $p^{n,}$ where p is a prime and n is a positive integer
- A Galois field, $GF(p^n)$, is a finite field with p^n elements.

Finite Fields of Order p

- For a given prime, p, the finite field of order p, GF(p) is defined as the set Zp of integers {0, 1,..., p- 1}, together with the arithmetic operations modulo p.
- Define GF(5) on the set Z5 (5 is a prime) with addition and multiplication operators

  GF(5) = {0,1,2,3,4}[+,*]

Finite Fields of Order p

Modular Addition

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

Finite Fields of Order p

Modular Multiplication

*	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Finite Fields of Order p

Additive inverse (w-) and Multiplicative inverse (w^{-1})

w	w-	w^{-1}
0	0	-
1	4	1
2	3	3
3	2	2
4	1	4

Advanced Encryption Standard

- The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits
- The AES specification uses the same three key size alternatives but limits the block length to 128 bits
- A number of AES parameters depend on the key length (Table)
- Assume a key length of **128 bits**, which is likely to be the one most commonly implemented

Table. AES Parameters

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

The AES Cipher - Rijndael

- Rijndael was selected as the AES in Oct-2000
  - Designed by Vincent Rijmen and Joan Daemen in Belgium
  - Issued as FIPS PUB 197 standard in Nov-2001

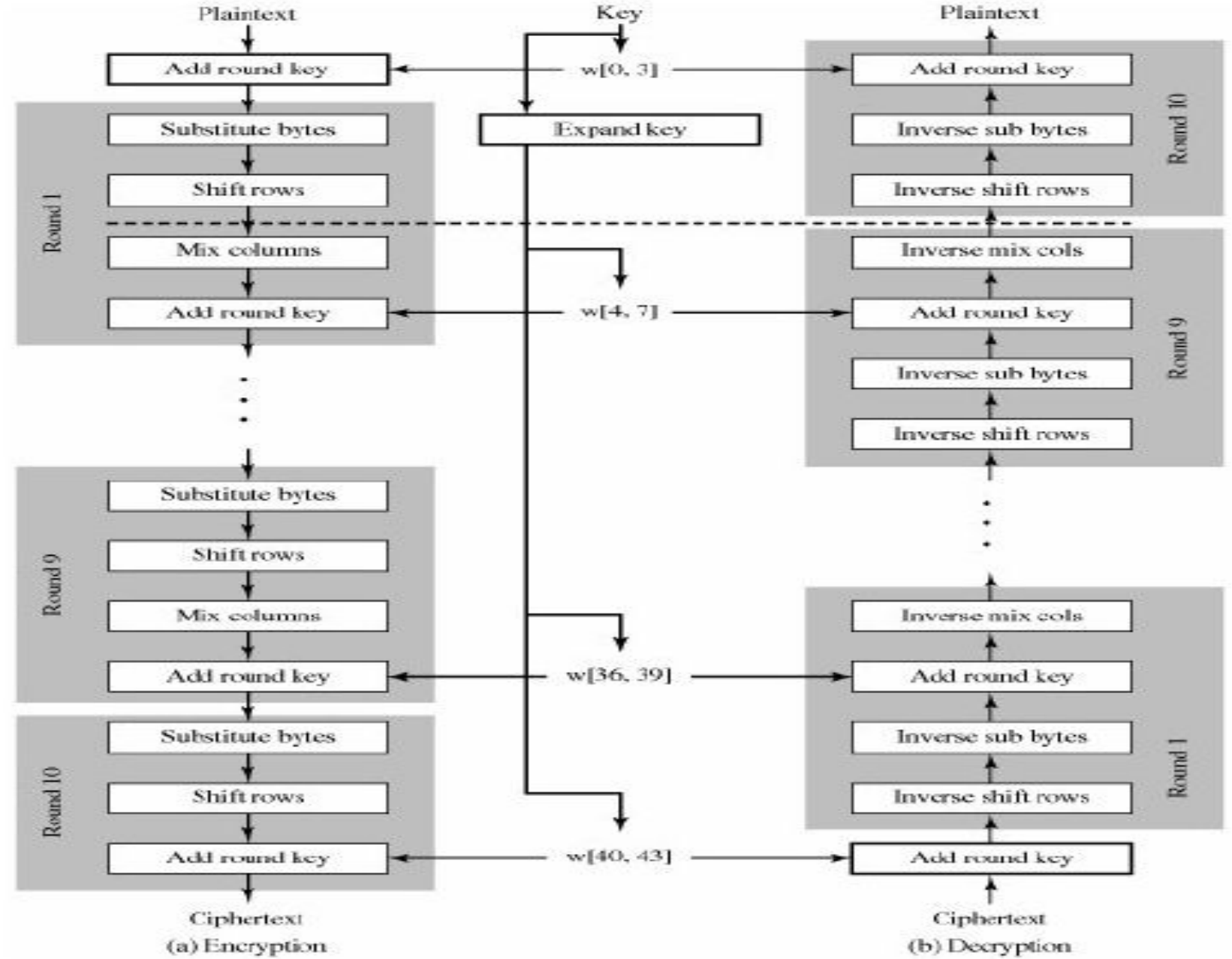
- An **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes (128 bits)
  - operates on entire data block in every round

- Rijndael design:
  - simplicity
  - has 128/192/256 bit keys, 128 bits data
  - resistant against known attacks
  - speed and code compactness on many CPUs

V. Rijmen

J. Daemen

AES Data Structures



Overall AES structure

- AES structure is not a **Feistel structure.**
- Two of the AES finalists, including Rijndael, do not use a
- Feistel structure but process the entire data block in parallel during each round using
- substitutions and permutation.

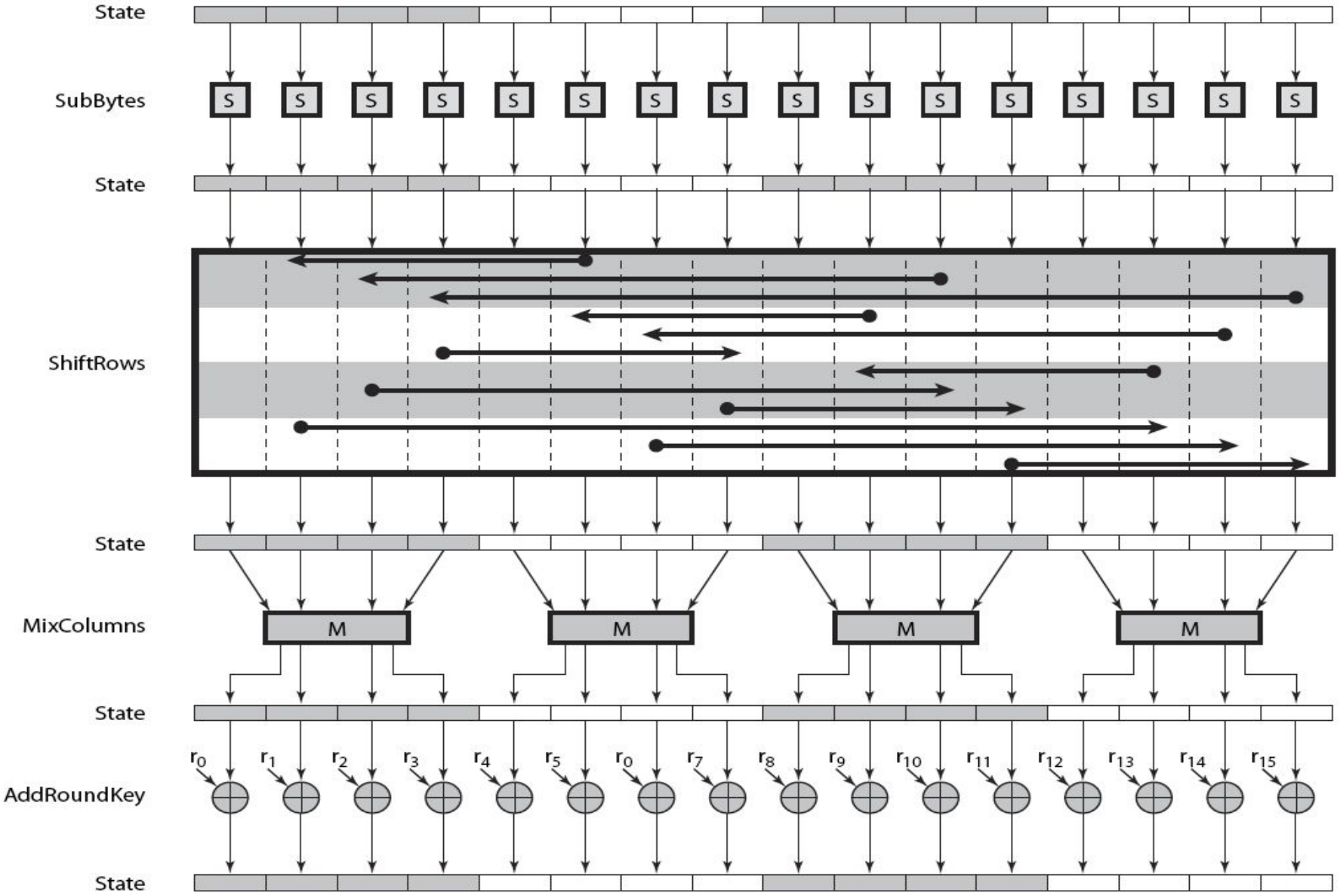
Four different stages are used, one of permutation and three of substitution:

- **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

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

- The Figure shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes.
- This block is copied into the **State array, which is modified at each stage of encryption or decryption.**
- **After the final** stage, **State is copied to an output matrix.**
- **These operations are depicted in Figure a. Similarly, the** 128-bit key is depicted as a square matrix of bytes.
- This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key (Figure 5.2b). Note that the ordering of bytes within a matrix is by column.
- So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in matrix,** the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w matrix.**

AES Encryption Round



- 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

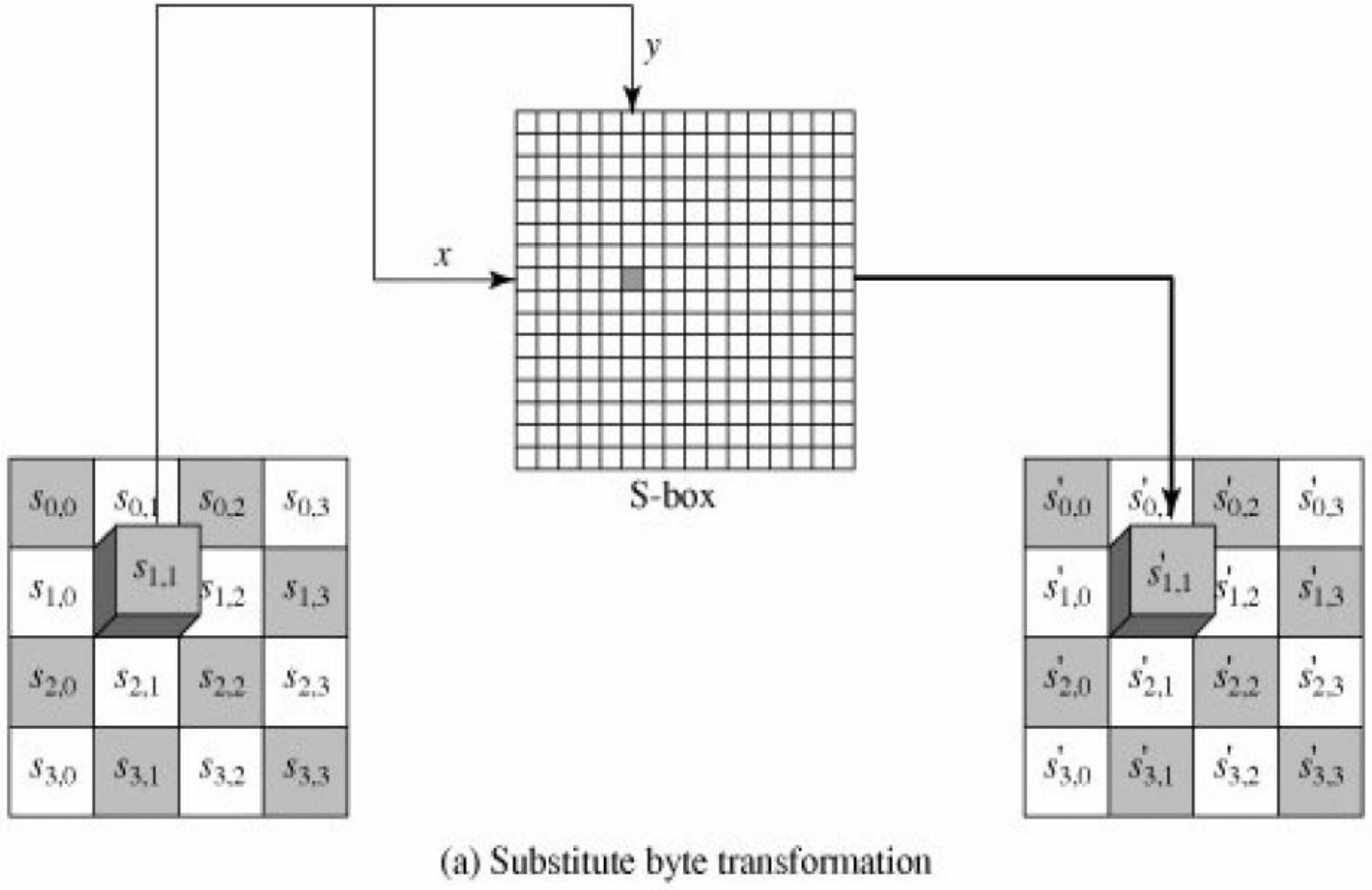
- For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm
- For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus A \oplus B = B$.
- As with most block ciphers, the decryption 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 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

Substitute Bytes Transformation

Forward and Inverse Transformations

- The **forward substitute byte transformation, called SubBytes, is a simple table lookup (Figure .a)**
- AES defines a 16 x 16 matrix of byte values, called an S-box (Table a), that contains a permutation of all possible 256 8-bit values
- Each individual byte of **State is mapped into a new byte in the following** way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.
- These row and column values serve as indexes into the S-box to select a unique 8-bit output value.

Figure. Substitute Bytes Transformation



(a) Substitute byte transformation

(a) S-box

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

(b) Inverse S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

Example of the SubBytes transformation

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

→

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

The S-box is constructed in the following fashion:

1.

   Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02},.... {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row x, column y is $\{xy\}$.

2.

   Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8); the value {00} is mapped to itself.

3.

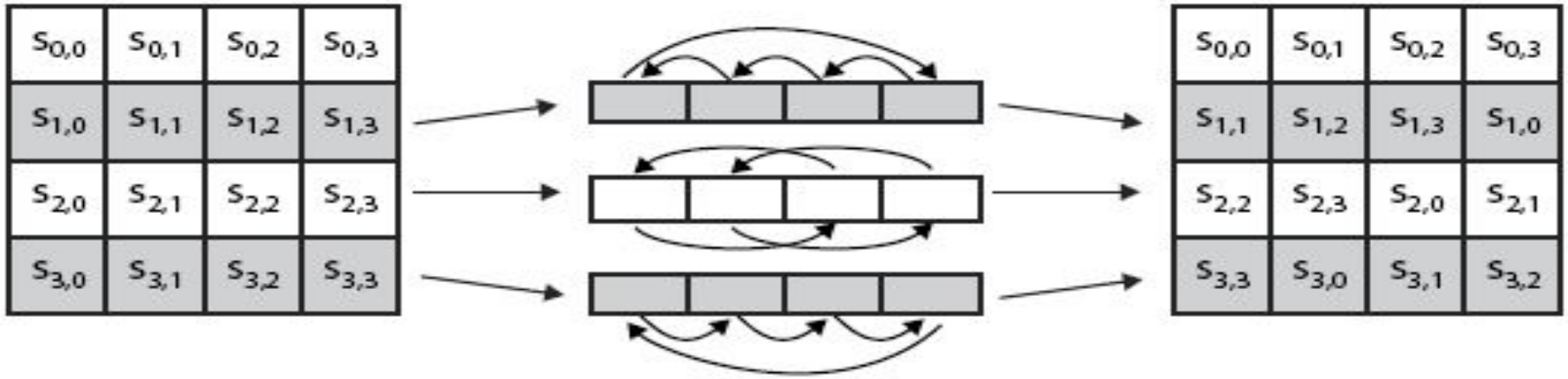
   Consider that each byte in the S-box consists of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0).

   Apply the following transformation to each bit of each byte in the S-box:

ShiftRows Transformation

Forward and Inverse Transformations

- The **forward shift row transformation, called ShiftRows, is depicted in Figure a**
- **The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-**byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed



The following is an example of ShiftRows:

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

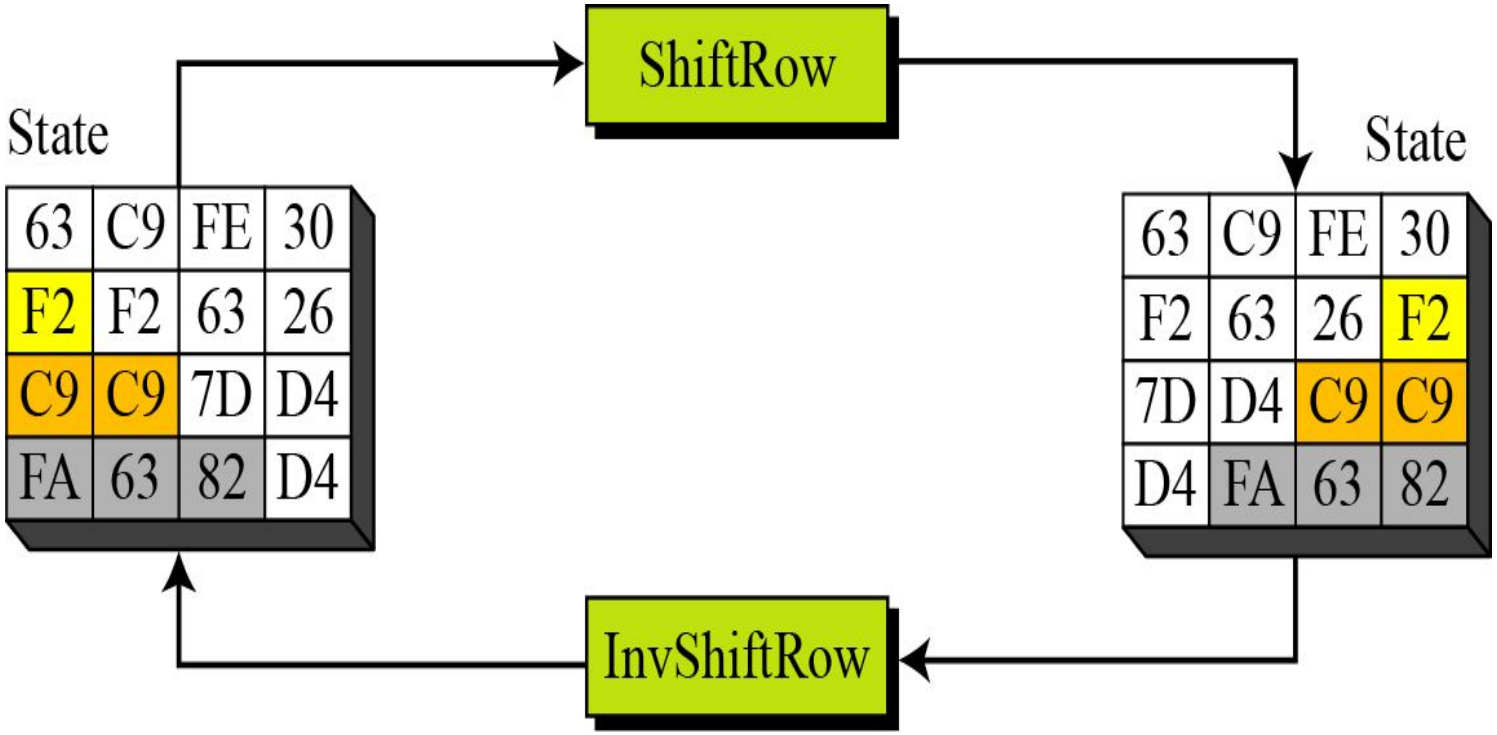
→

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

Figure AES Row and Column Operations

- The **inverse shift row transformation, called InvShiftRows, performs the circular shifts in the** opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on
- The shift row transformation is more substantial than it may first appear**.** This is because the **State, as** well as the cipher input and output, is treated as an array of four 4**-**byte columns
- Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State, and so on. Further, as will be** seen, the round key is applied to **State column by column.**
- **Thus, a row shift moves an individual byte** from one column to another, which is a linear distance of a multiple of 4 bytes

ShiftRows and InvShiftRows



MixColumns Transformation

Forward and Inverse Transformations

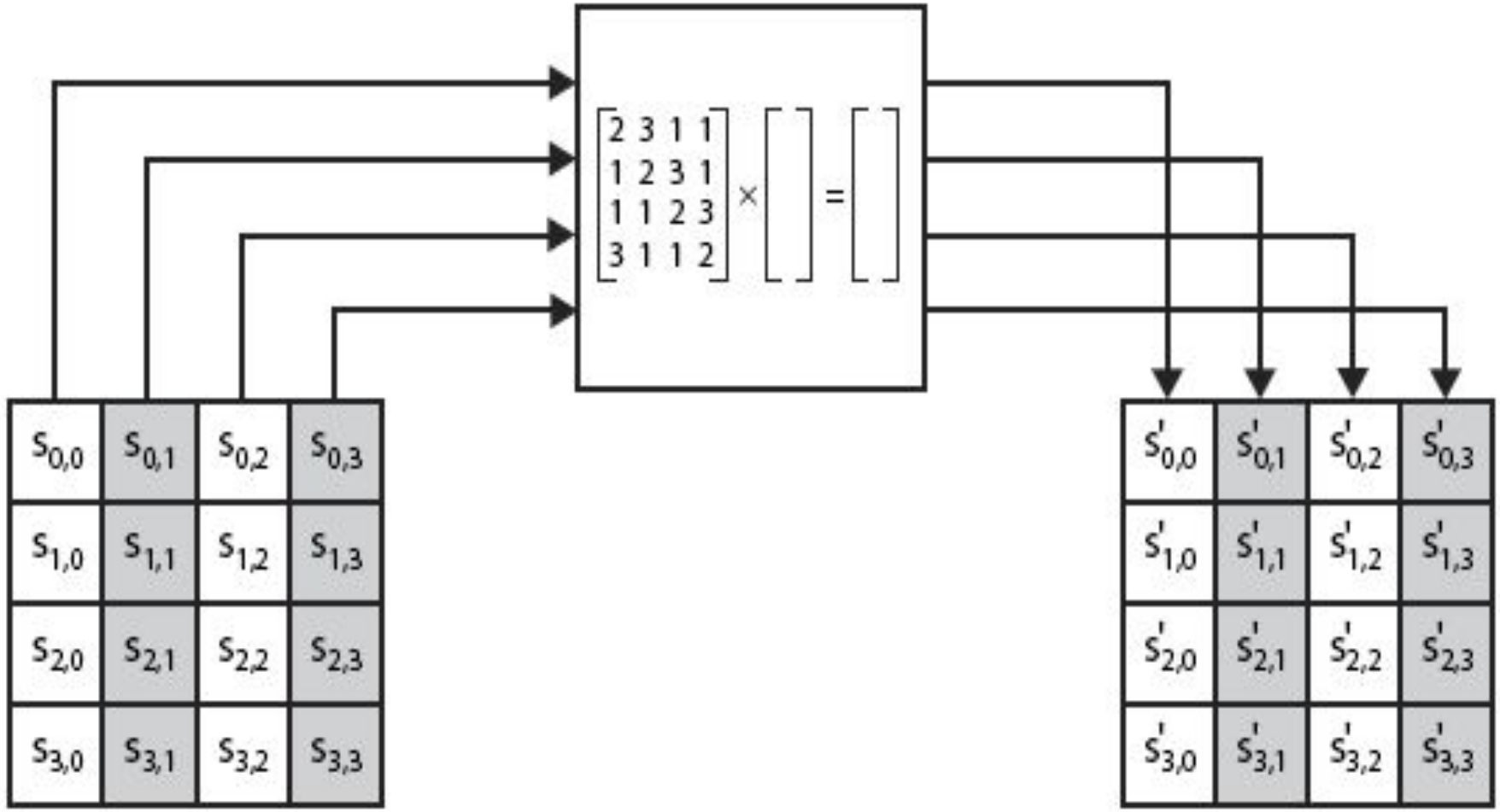
- The **forward mix column transformation, called MixColumns, operates on each column individually**
- Each byte of a column is mapped into a new value that is a function of all four bytes in that column.
- The transformation can be defined by the following matrix multiplication on **State**

The following is an example of MixColumns:

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

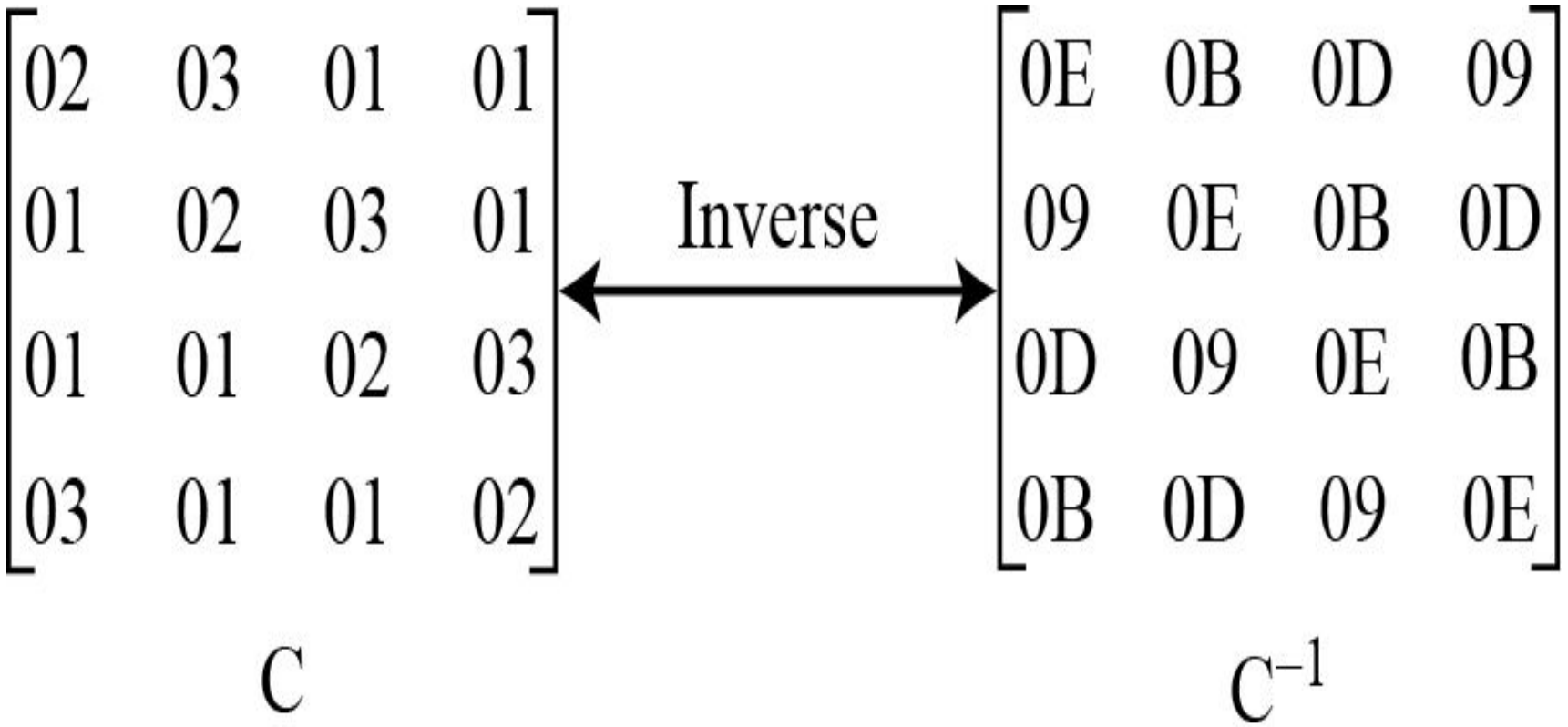
→

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



The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

MixColumn and InvMixColumn



AddRoundKey Transformation

Forward and Inverse Transformations

- In the **forward add round key transformation, called AddRoundKey, the 128 bits of State are** bitwise XORed with the 128 bits of the round key
- As shown in Figure, the operation is viewed as a columnwise operation between the 4 bytes of a **State column and one word of the round key; it can also** be viewed as a byte-level operation.
- The following is an example of AddRoundKey:

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

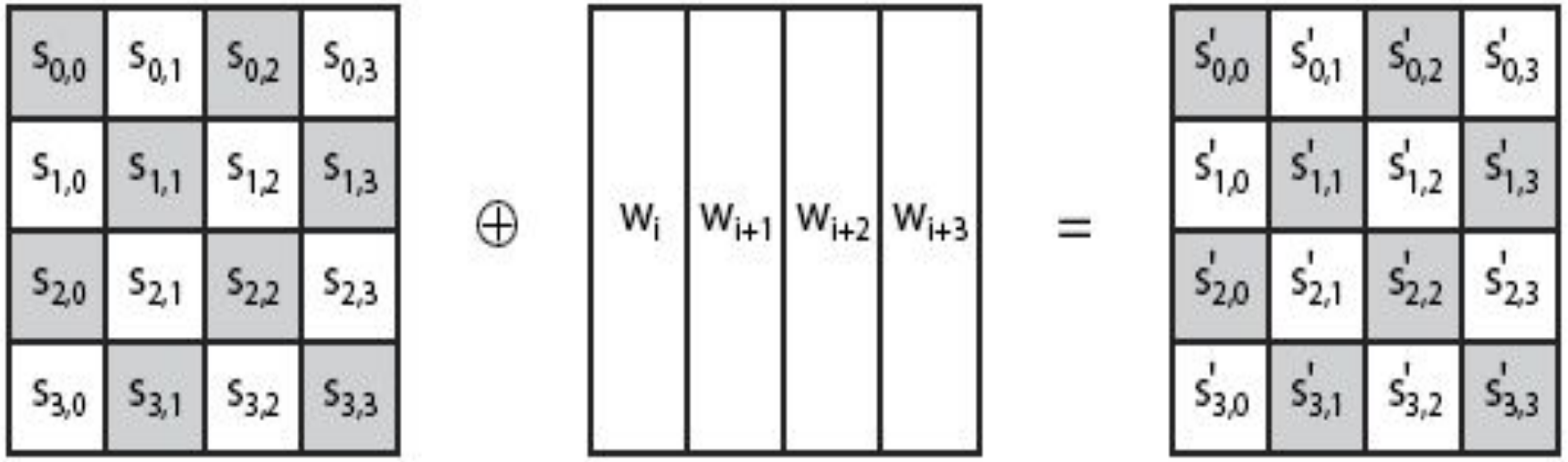
$\oplus$

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

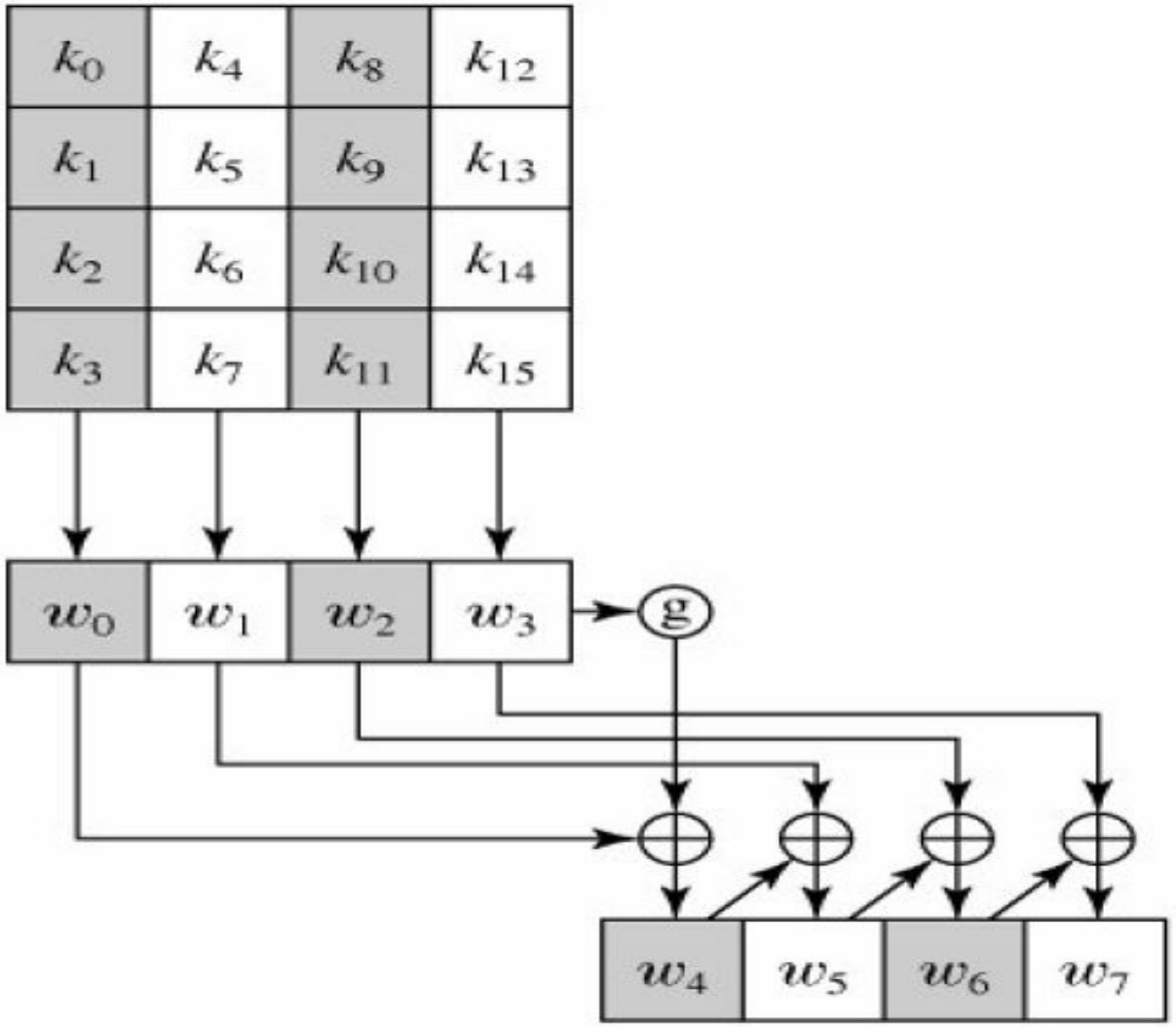
=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

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



AES Key Expansion



AES Key Scheduling

- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words

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

AddRoundKey

- XOR state with 128-bits of the round key
- AddRoundKey proceeds one column at a time.
  - adds a round key word with each state column matrix
  - the operation is matrix addition
- Inverse for decryption identical
  - since XOR own inverse, with reversed keys
- Designed to be as simple as possible

Key Expansion submodule

- **RotWord** performs a one byte circular left shift on a word For example:

  RotWord[b0,b1,b2,b3] = [b1,b2,b3,b0]

- **SubWord** performs a byte substitution on each byte of input word using the S-box

- **SubWord(RotWord(temp))** is XORed with RCon[j] – the round constant

AES Security

- AES was designed after DES.
- Most of the known attacks on DES were already tested on AES.
- Brute-Force Attack
  - AES is definitely more secure than DES due to the larger-size key.
- Statistical Attacks
  - Numerous tests have failed to do statistical analysis of the ciphertext
- Differential and Linear Attacks
  - There are no differential and linear attacks on AES as yet.

RC4

- Rivest Cipher 4 (RC4) is a stream cipher designed in 1987 by Ron Rivest for RSA Security.
- It is a variable key size stream cipher with byte-oriented operations.
- The algorithm is based on the use of a random permutation.
- For encryption and decryption, a byte *k* is generated from S by selecting one of the 255 entries in a systematic fashion.
- To encrypt, XOR the value k with the next byte of plaintext.
- To decrypt, XOR the value k with the next byte of ciphertext.

RC4 – Key Scheduling Algorithm (KSA)

- A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0],S[1], ..., S[255].
- If the length of the key k is 256 bytes, then k is assigned to T.
- Otherwise, for a key with length(k-len) bytes, the first k-len elements of T as copied from K and then K is repeated as many times as necessary to fill T.
- Use T to produce the initial permutation of S. Starting with S[0] to S[255], and for each S[i] algorithm swap it with another byte in S according to a scheme dictated by T[i], but S will still contain values from 0 to 255

RC4 – Key Scheduling Algorithm (KSA)

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

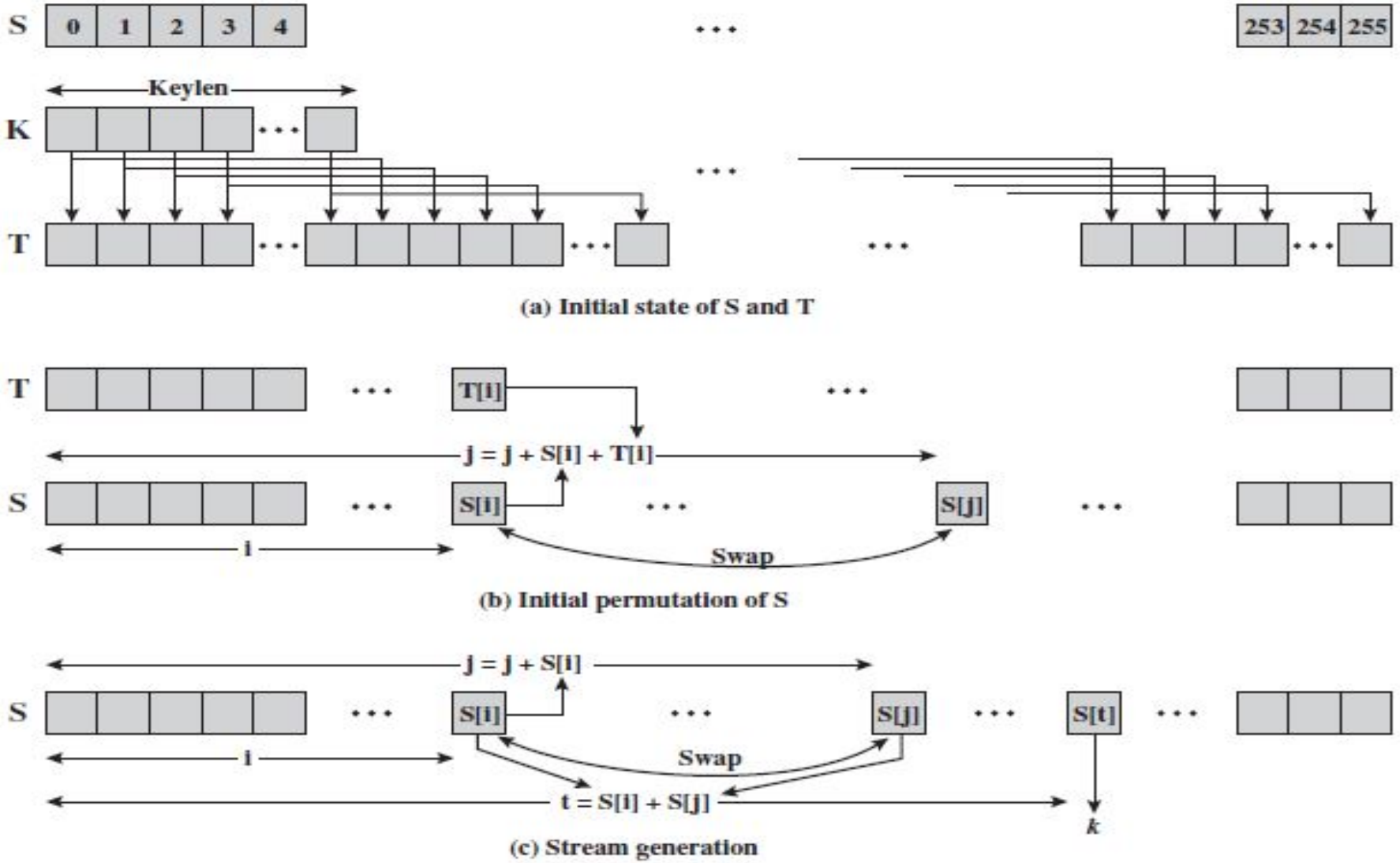
```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
```

RC4 – Pseudo Random Generation Algorithm (PRGA)

- For each S[i] algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S.
- After reaching S[255] the process continues, starting from S[0] again.

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

RC4 – Stream generation



KEY DISTRIBUTION

- A term that refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key.
- For two parties A and B, key distribution can be achieved in a number of ways, as follows:
  - A can select a key and physically deliver it to B.
  - A third party can select the key and physically deliver it to A and B.
  - If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
  - If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

DIFFIE-HELLMAN KEY EXCHANGE

- The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages
- The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms

- A primitive root of a prime number whose powers modulo p generate all the integers from 1 to p -1
- That is, if a is a primitive root of the prime number p , then the numbers

$$a \bmod p, a^2 \bmod p, \ldots, a^{p-1} \bmod p$$

  are distinct and consist of the integers from 1 through p-1 in some permutation
- For any integer b and a primitive root a of prime number p , a unique exponent can be found such that the exponent is referred to as the **discrete logarithm** of b for the base a , mod p .This value can be expressed as $\text{dlog}_{a,p}(b)$

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{XA} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{XB} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{XA} \bmod q$

Calculation of Secret Key by User B

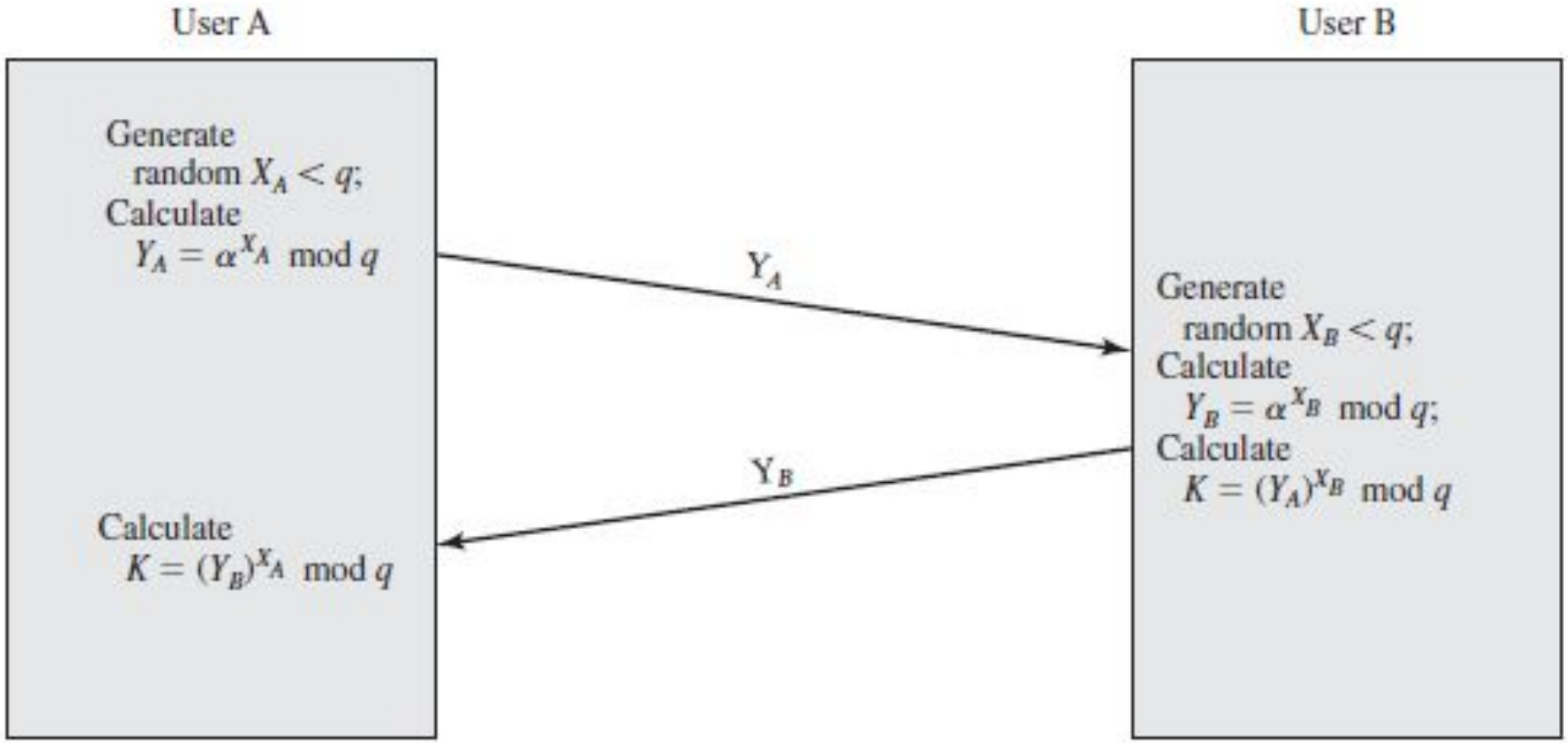
$K = (Y_A)^{XB} \bmod q$

Alice

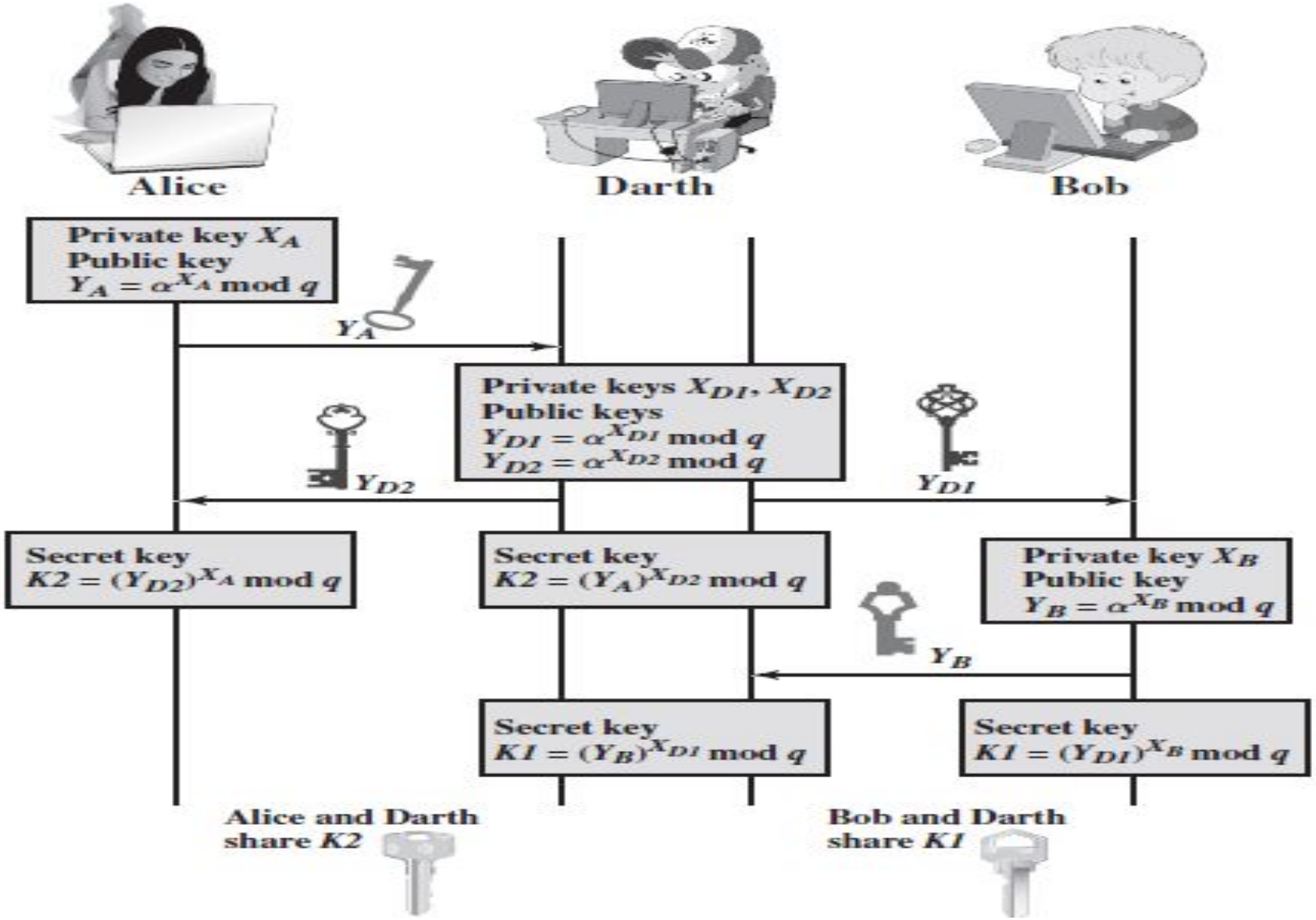
Bob

Alice		Bob
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q		Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$		Bob generates a private key X_B such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$	Y_A →	Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
Alice receives Bob's public key Y_B in plaintext	← Y_B	Bob receives Alice's public key Y_A in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$		Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Key Exchange Protocols



Man-in-the-Middle Attack



- Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}
- Alice transmits Y_A to Bob
- Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates
- Bob receives Y_{D1} and calculates
- Bob transmits Y_B to Alice
- Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates
- Alice receives Y_{D2} and calculates