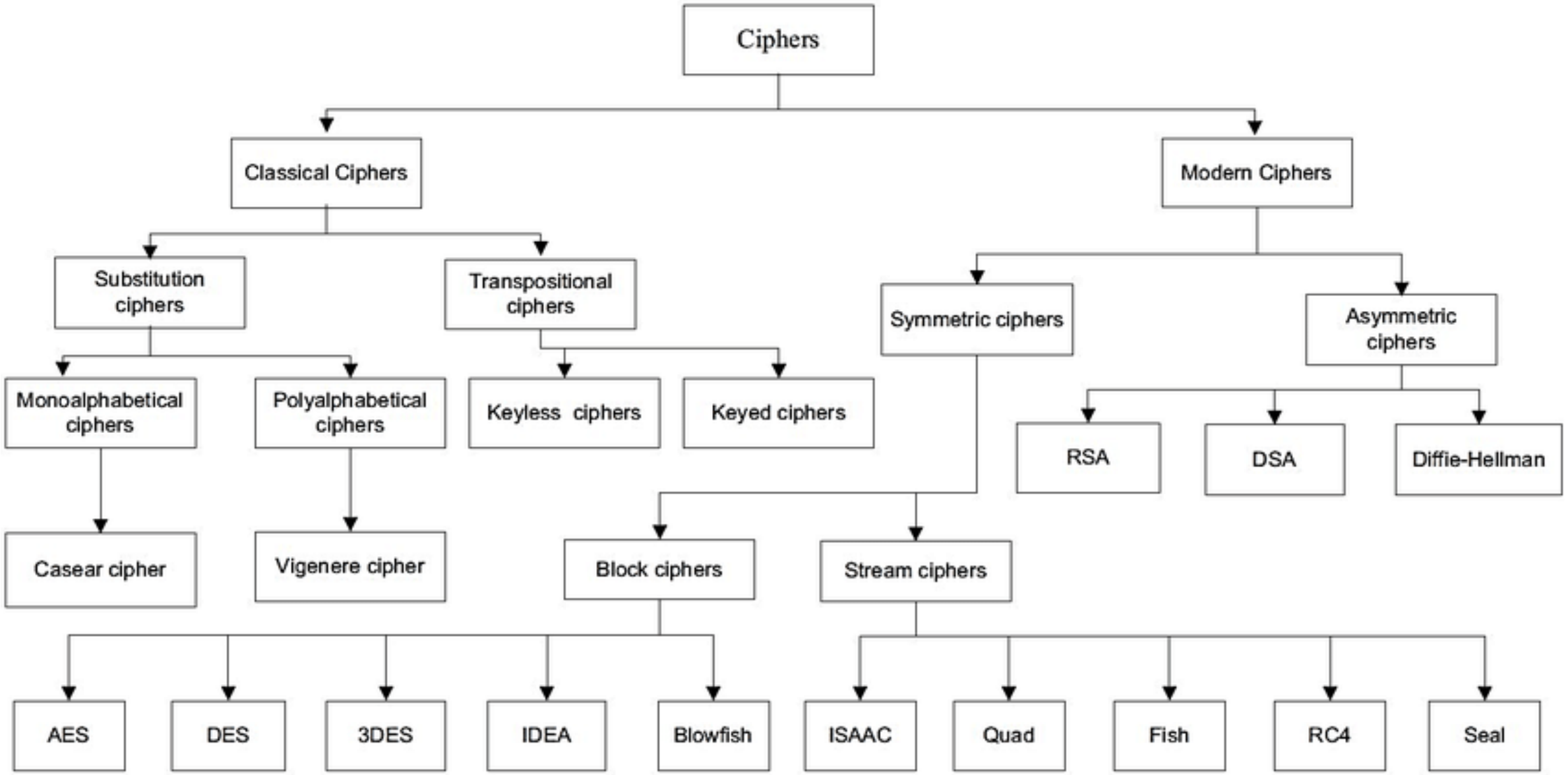
Modern Cryptography

Prepared By-ANOOJA JOY

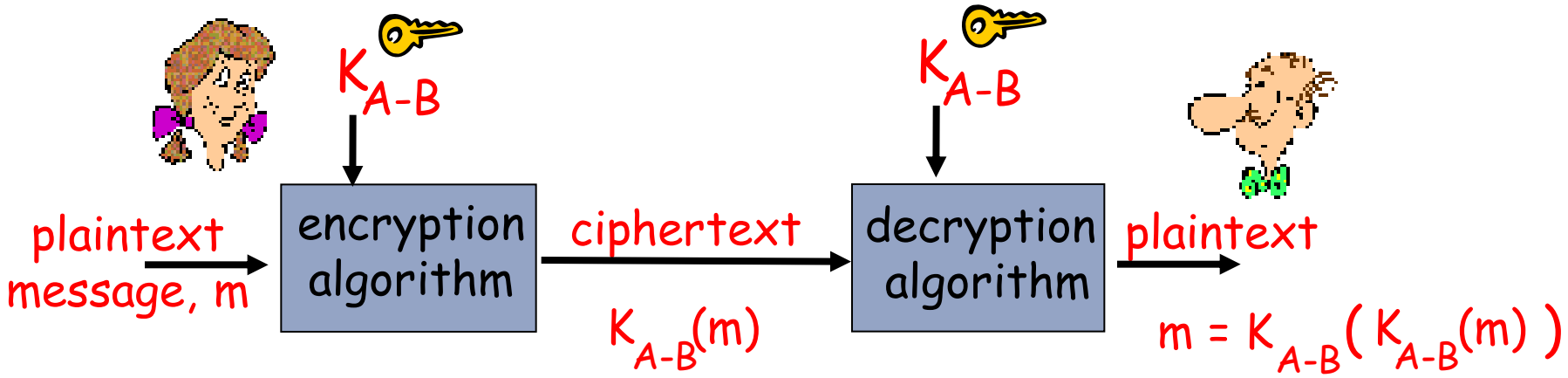
CLASSIFICATION OF CIPHER ALGORITHMS



MODERN CRYPTOGRAPHY

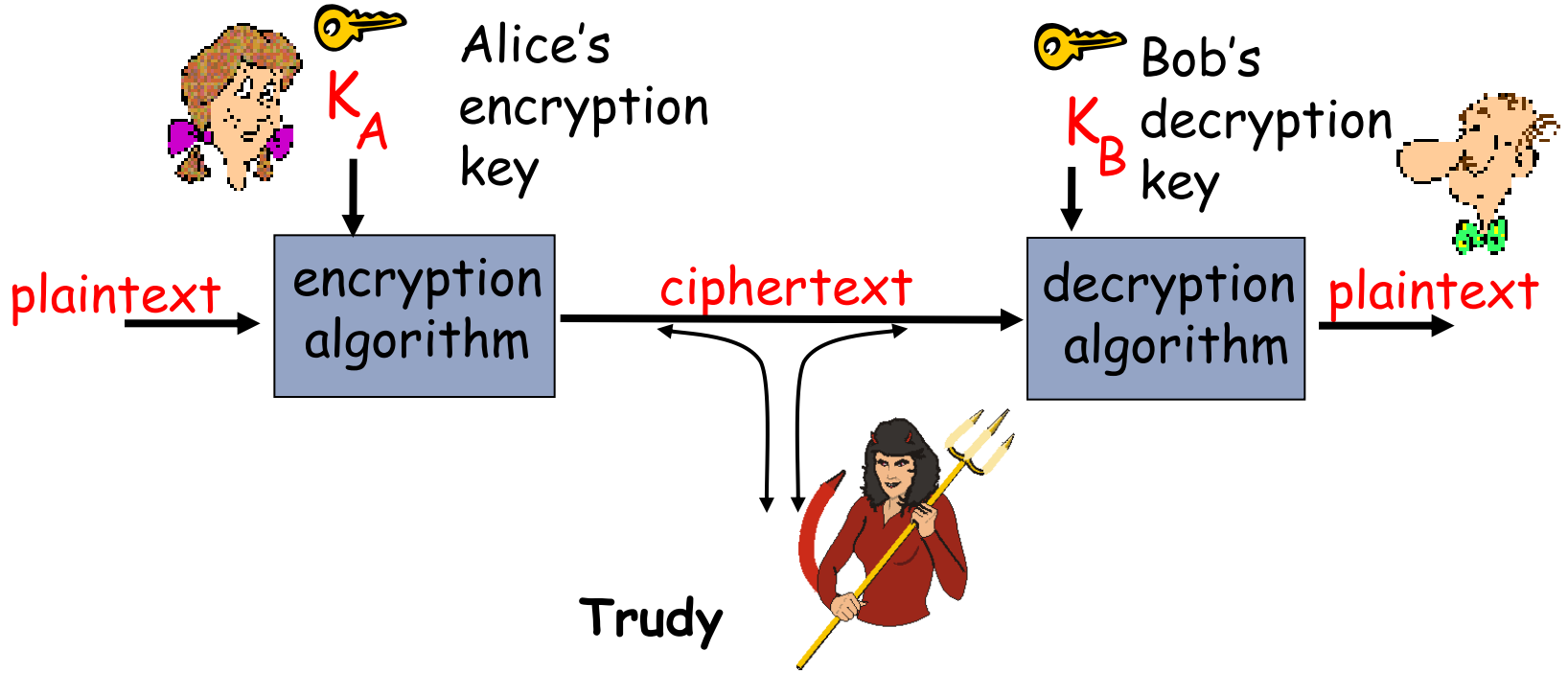
- ***Modern cryptography*** mainly acts on **binary bits** rather than characters.
- **Symmetric Modern Cryptographic Algorithms** for securing message is **public** and **key** needs to be a **secret.**
- ***Stream Ciphers*** and ***Block ciphers*** are two categories of ciphers used in **symmetric modern cryptography.**
- Stream and Block Ciphers differ in how large a piece of the message is processed in each encryption operation.

SYMMETRIC-KEY(PRIVATE-KEY) CRYPTOGRAPHY



- If **same key(K_{A-B})**is used by the sender(for encryption) and the receiver (for decryption) then its known as symmetric key cryptography. The key is shared by sender and receiver through a **secret channel** and is not known to public.
- The **plaintext** and the **ciphertext** having the **same size.**
- **Eg:** AES, DES

ASYMMETRIC-KEY(PUBLIC-KEY) CRYPTOGRAPHY



- If **different keys** are used by the sender (KA for encryption) and the receiver (KB for decryption) then its known as asymmetric key cryptography. The key is not shared publicly instead its been calculated as per the cipher.
- **Eg:** RSA, Elgamal

Public Cryptography

1. Public key is asymmetrical. Public key is used to encrypt message and private key is used to decrypt message.
2. No sharing of key and no problem in key sharing
3. In software implementation public key is faster
4. Often used for securely exchanging secret keys. Used for encryption and decryption as well for digital signatures.

- **Eg:** Diffie-Hellman, RSA, Elgamal

Private Cryptography

1. Private key is symmetrical . There is only one key and decryption key is a copy of it.
2. Both communicating parties shares the secret key and key sharing is a major challenge.
3. In hardware symmetric-key algorithms are faster
4. Used mainly for bulk data transmission and mainly for encryption and decryption an not for digital signatures.
5. **Eg:** DES, 3DES, AES, and RC4.

STREAM CIPHER

- Stream Ciphers encrypt plaintext **one byte or one bit at a time.**
- Eg: RC4, A5/1
- Stream Ciphers are **Faster**
- Stream cipher uses **only confusion.**
- Stream cipher **uses 8 bits.**
- Stream cipher is more complex.
- One key is used only one time
- Used in hardware implementation

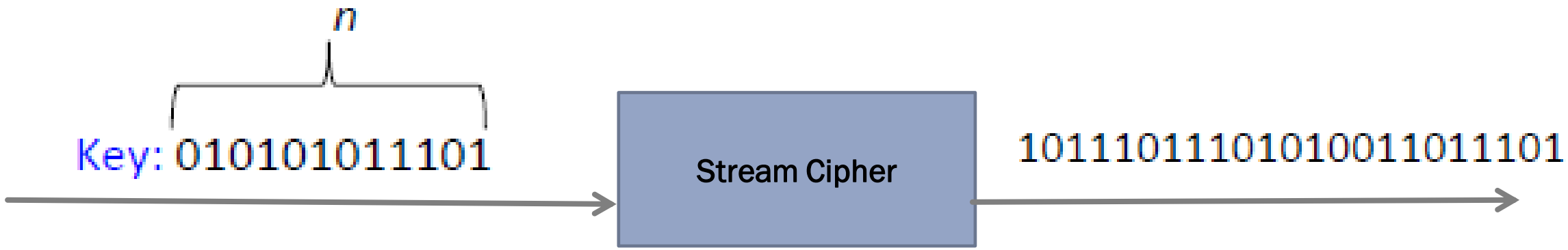
BLOCK CIPHER

- Block Ciphers encrypt plaintext in **chunks.** Common block sizes are **64 and 128 bits.**
- Eg: AES,DES
- Block cipher is **Slower**
- Block cipher Uses **confusion** as well as **diffusion**
- Block cipher uses blocks of size **64 or 128 bits**
- Complexity of block cipher is less.
- One key can be used in multiple times.
- Used in software implementation

STREAM CIPHER

STREAM CIPHER

- **Stream cipher:** Takes a key K of n bits in length and stretches it into a long key Stream



Encryption:

- Plaintext P = p 0 , p 1 , p 2 , p 3 , . . ., p n
- KeyStream S = s 0 , s 1 , s 2 , s 3 , . . ., s n
- Ciphertext C = c 0 = p0 ⊕ s 0 , c 1 = p1 ⊕ s 1 , c 2 = p2 ⊕ s 2 , . . ., c n = p n ⊕ s n

- Important property of **XOR** is that **it is reversible**, when applied twice it produces original data.

XOR OPERATION

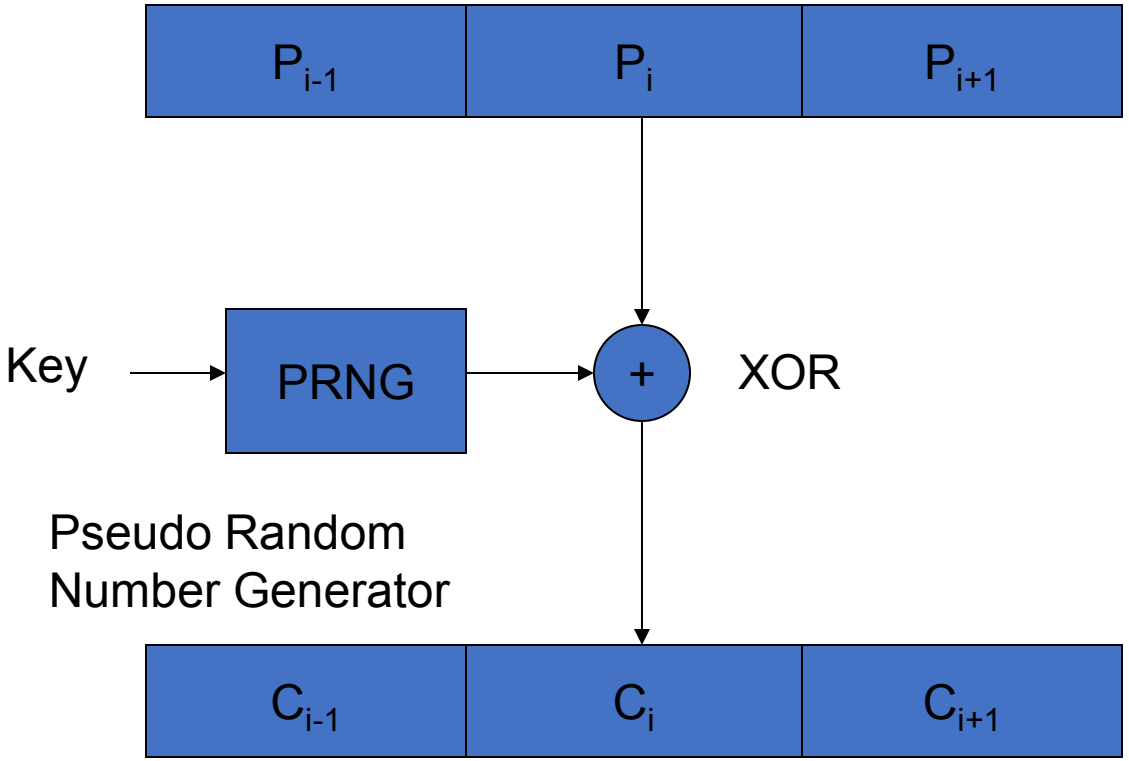
Input	output
$0 \oplus 0$	0
$0 \oplus 1$	1
$1 \oplus 0$	1
$1 \oplus 1$	0

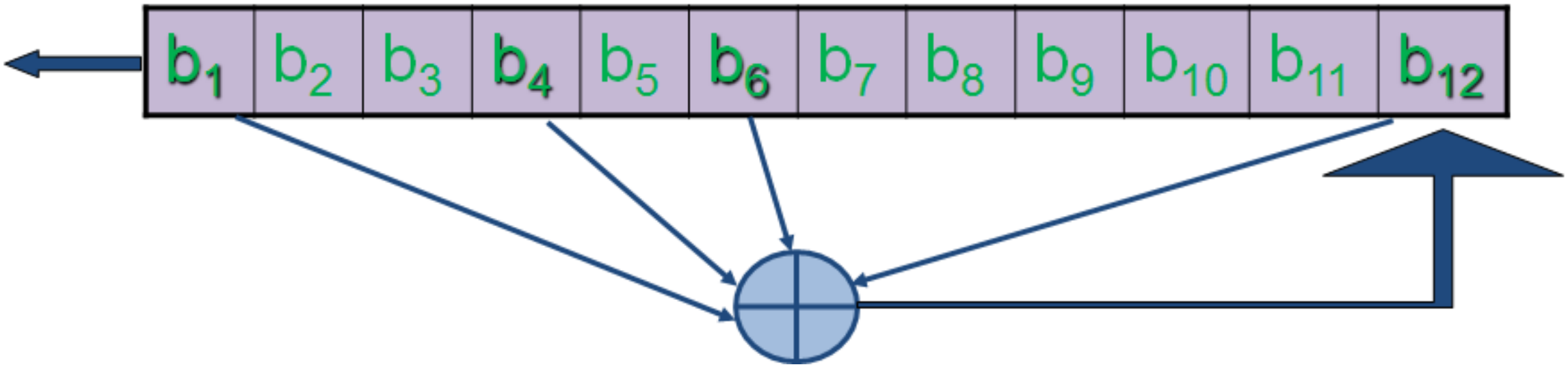
Truth table of the XOR operation

STREAM CIPHER

- A ***stream cipher*** is a method of encrypting **plain text data stream** using a **cryptographic key** which is a **pseudorandom digit stream** and an **algorithm,** by applying it to each binary digit at a time.
- **Pseudorandom keystream** is typically generated serially from a random seed value using **digital shift registers**.
- It's a **symmetric key** cipher both sender and receiver has same stream cipher algorithm, and both know key K.
- A stream cipher takes a **key K** of **n bits** which is **stretched** to a long keystream (equivalent to plain text)and which is **XORed with plain text** to produce cipher text.
- StreamCipher(k)=S where S is the resulting keystream represented as s_0 ,s_1 ,s_2...
- **ENCRYPTION**: CipherText $c_0 = p_0 \oplus s_0$,$c_1 = p_1 \oplus s_1$
- **DECRYPTION:** PlainText $p_0=c_0 \oplus S_0$,$p_1=c_1 \oplus S_1$
- Key stream should never be reused and should be random too else messages can be easily recovered.

Stream Ciphers

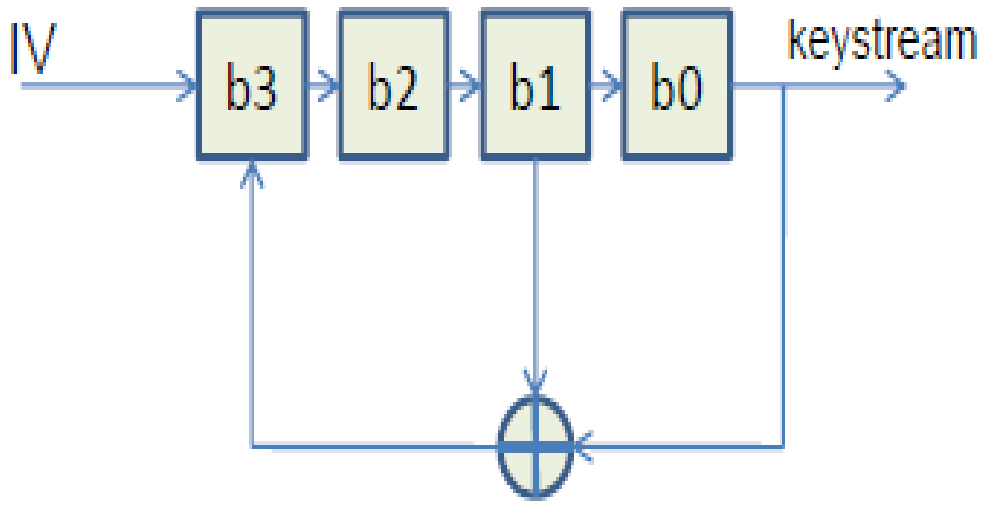




Linear feedback shift registers (LFSR)

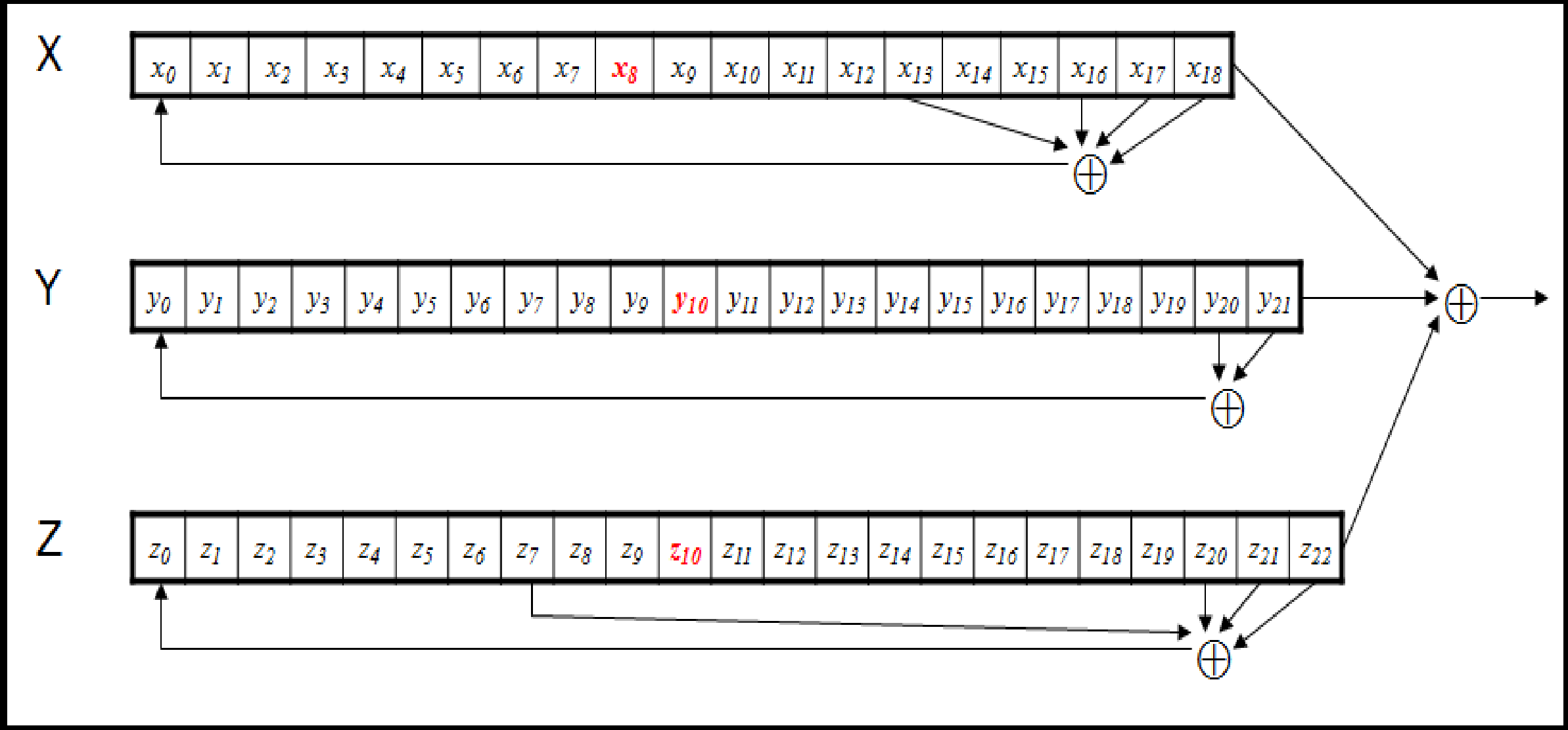
- $x^{12}+x^6+x^4+x+1$ corresponds to LFSR of length 12

LFSR



Initialization Vector

b3	b2	b1	b0
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	1
1	1	0	0
0	1	1	0
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1
1	0	0	0


X
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17} x_{18}
Y
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12} y_{13} y_{14} y_{15} y_{16} y_{17} y_{18} y_{19} y_{20} y_{21}
Z
z_0 z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10} z_{11} z_{12} z_{13} z_{14} z_{15} z_{16} z_{17} z_{18} z_{19} z_{20} z_{21} z_{22}

A5/1 Cipher

A5/1 Stream Cipher

- **A5/1** uses **three linear feedback shift registers(LFSR),** which hold total **64 bits.**

Initialization

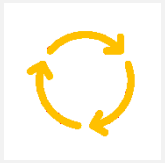
1. **Register X** hold 19 bits of the key (x_0, x_1, x_2, …, x_{18})
2. **Register Y** hold **22 bits** of the **key (y0, y1, y2, …, y21)**
3. **Register Z** hold **23 bits** of the **key (z0, z1, z2, …, z22)**

- Each value is a **single bit**
- Key K is 64bits and is used as t**he initial fill for 3 shift registers** in order t**o generate KeyStream.**
- Keystream bit is **XOR of rightmost bits of registers**
- Each register **steps (or not)** based on **maj(x_8, y_{10}, z_{10})**
- **APPLICATION:** Used in confidentiality for **GSM phones.**

Majority Function of A5/1

Majority function **maj(x,y,z)** maps multiple inputs to one output.

Its value is **false** when **n/2 or more values are false** else **true**.



At each iteration: $m = maj(x_8, y_{10}, z_{10})$

Examples:

maj(0,1,0) =0

maj**(1,1,0) = 1**

A5/1: Keystream

For Each Key stream Bit we generate

- At each iteration: $m = \text{maj}(x_8, y_{10}, z_{10})$
- If $x_8 = m$ then X *steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y *steps*
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ and $y_0 = t$
- If $z_{10} = m$ then Z *steps*
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$
- Keystream **bit** is $x_{18} \oplus y_{21} \oplus z_{22.}$ It can be XORed with plaintext for encryption
- Repeat the entire process as many key stream bits are required.

A5/1 Shift Registers

Register Number	Length In bits	Primitive Polynomial	Clock-Controlling Bit (LSB is 0)	Bits that Are XORed
X	19	$x^{19} + x^{18} + x^{17} + x^{16} + x^{13} + 1$	8	18,17,16,13
Y	22	$x^{22} + x^{21} + x^{20} + 1$	10	21,20
Z	23	$x^{23} + x^{22} + x^{21} + x^{20} + x^{7} + 1$	10	22,21,20,7

A5/1 Exercise

- **Implement the A5*/1 algorithm. Suppose that, after a particular step, the values in* the registers are**

X = (x0, x1, . . . , x18) = (1010101010101010101)

Y = (y0, y1, . . . , y21) = (1100110011001100110011)

Z = (z0, z1, . . . , z22) = (11100001111000011110000)

- Generate and print the next 32 keystream bits. Print the contents of *X, Y and Z after* the 32 keystream bits have been generated

SOLUTION

- *X = (x0, x1, . . . , x18) = (10 10 10 10 10 10 10 10 10 1)*
- *Y = (y0, y1, . . . , y21) = (11 00 11 00 11 00 11 00 11 00 11)*
- *Z = (z0, z1, . . . , z22) = (11 10 00 01 11 10 00 01 11 10 00 0)*
- M= major vote (maj(x8,y10,z10))
- M=maj(1,0,1)=1

For register x

- Compare x_8=1 with m =1 they are equal then do step
- $T_x = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
- $T_x = 0 \oplus 1 \oplus 0 \oplus 1$, Tx=0

X = (tx,,x0, x1, . . . , x17) = (0 10 10 10 10 10 10 10 10 10)

For register y

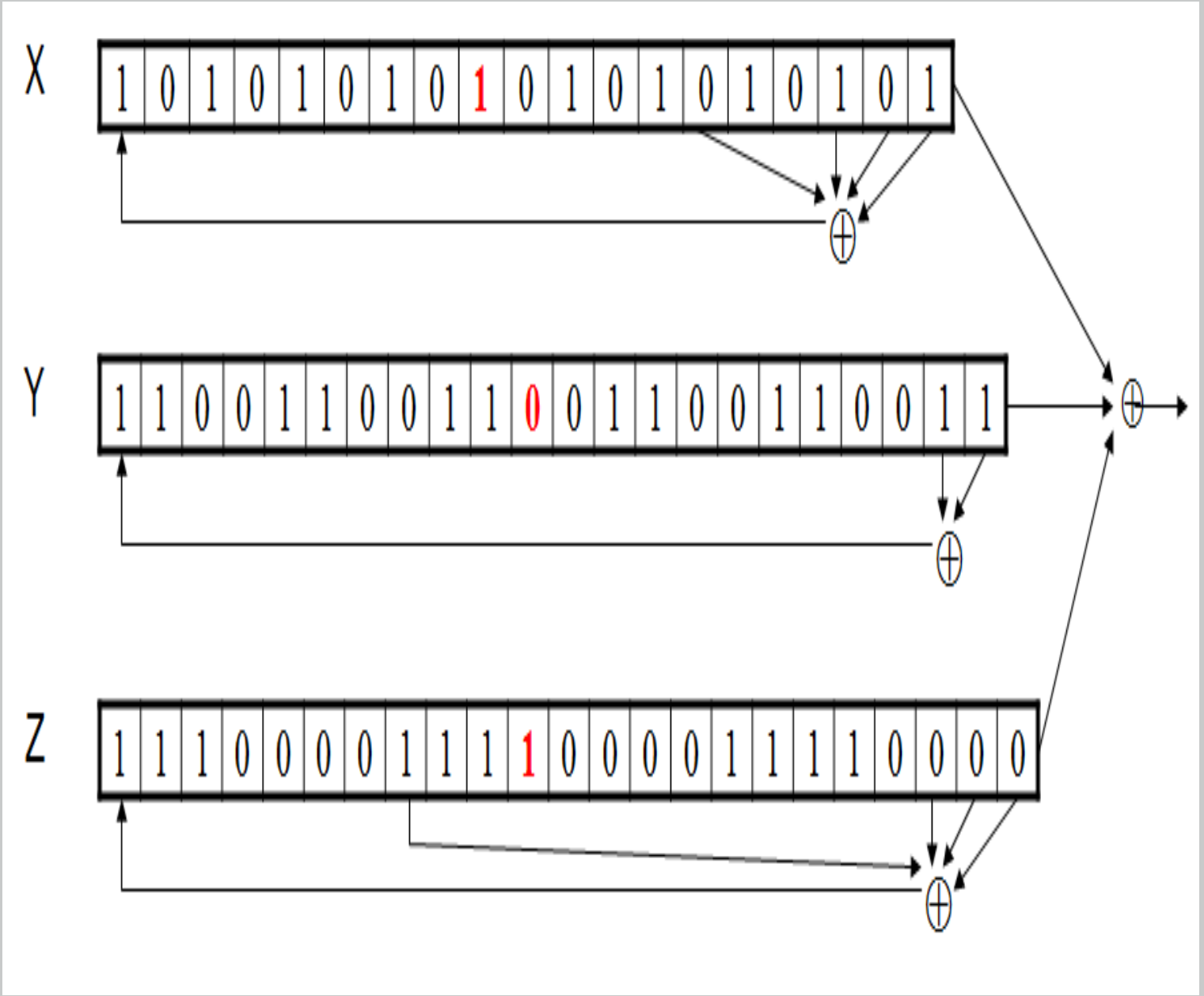
- Compare y_{10}=0 with m =1 they are not equal then do not do step
- *Y = (y0, y1, . . . , y21) = (11 00 11 00 11 00 11 00 11 00 11)*

For register z

- Compare z_{10}=1 with m =1 they are equal then do step
- $T_z = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
- $T_z = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
- *Z = (T_z, z0, z1, . . . , z21) = (1 11 10 00 01 11 10 00 01 11 10 00)*
- **Keystream bit is $x_{18} \oplus y_{21} \oplus z_{22} = 0 \oplus 1 \oplus 0 = 1$**

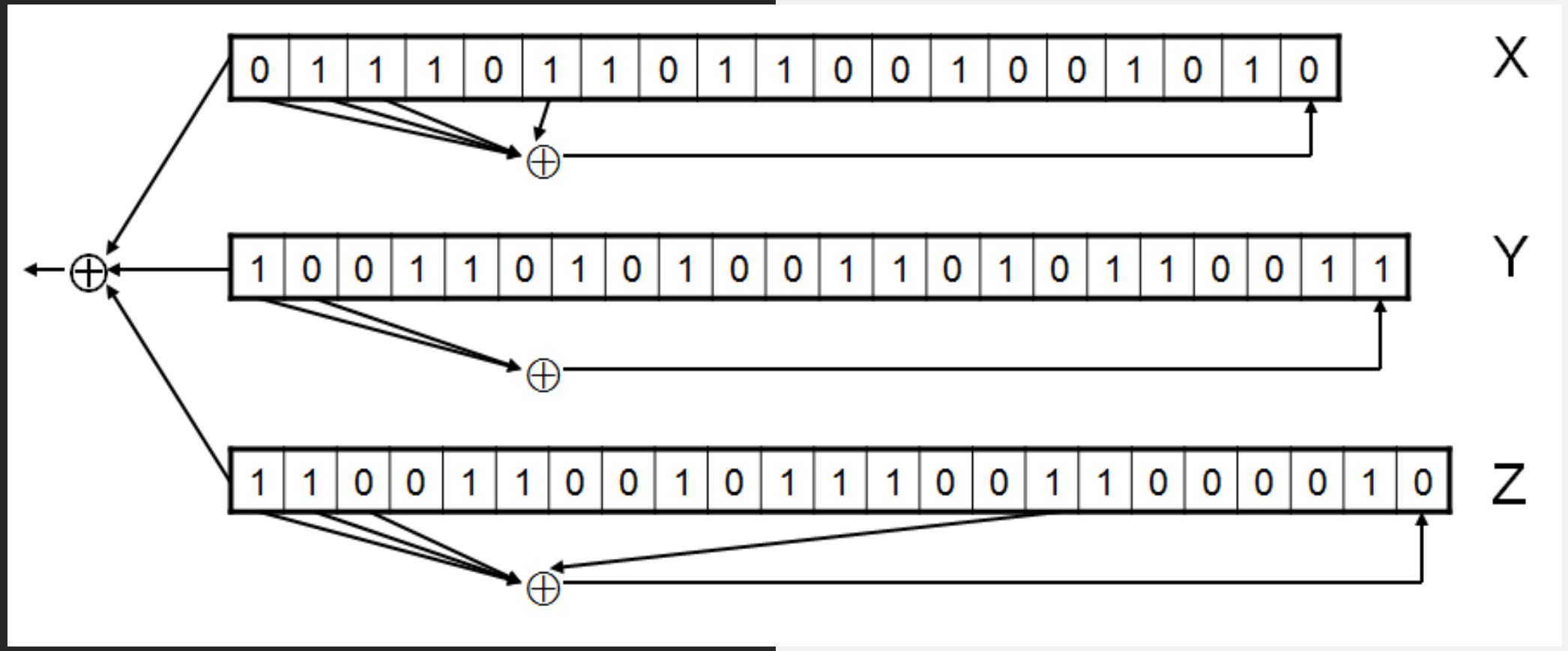
SOLUTION

- In this example, $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(\mathbf{1},\mathbf{0},\mathbf{1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$



A5/1 Challenge

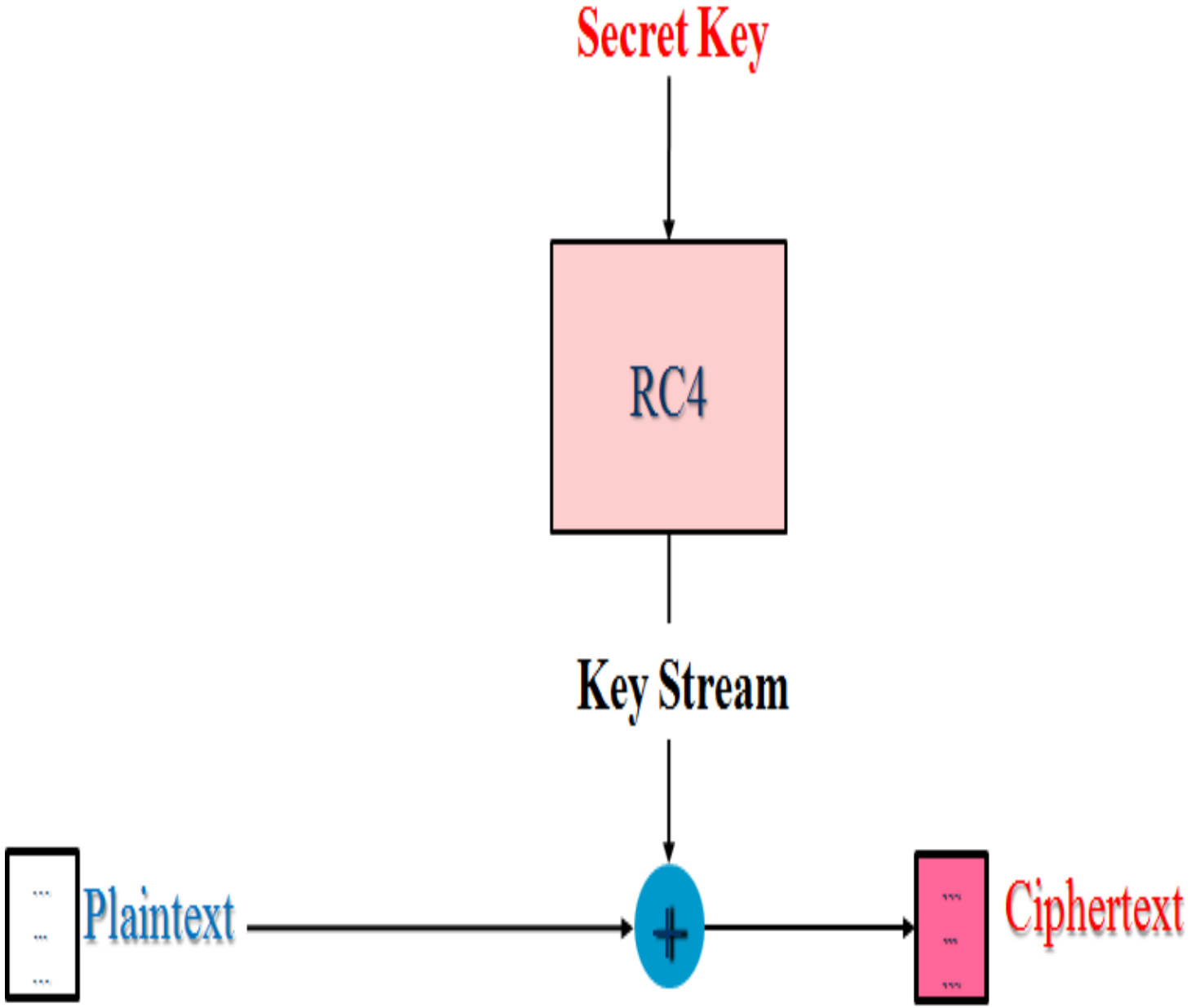
- Generate the first keystream bit using A5/1 cipher



RC4

- RC4(**Ron's Code** also known as **ARC4** or **ARCFOUR)** produces a keystream byte (**pseudorandom stream of bits**)at each step.
- Keystream can be used for encryption by combining it with the plaintext using bit-wise exclusive-or.
  - A proprietary cipher owned by RSA, kept secret. **Ron Rivest invented RC4 in 1987.**
  - Code released at the sites of Cyberpunk remailers
- Used in
  - **SSL/TL**S (Secure socket, transport layer security) between web browsers and servers,
  - IEEE 802.11 wirelss LAN std: WEP (Wired Equivalent Privacy), WPA (WiFi Protocol Access) protocol
  - Bit Torrent Protocol Encryption
  - Microsoft Point-to-Point Encryption

RC4 Block Diagram



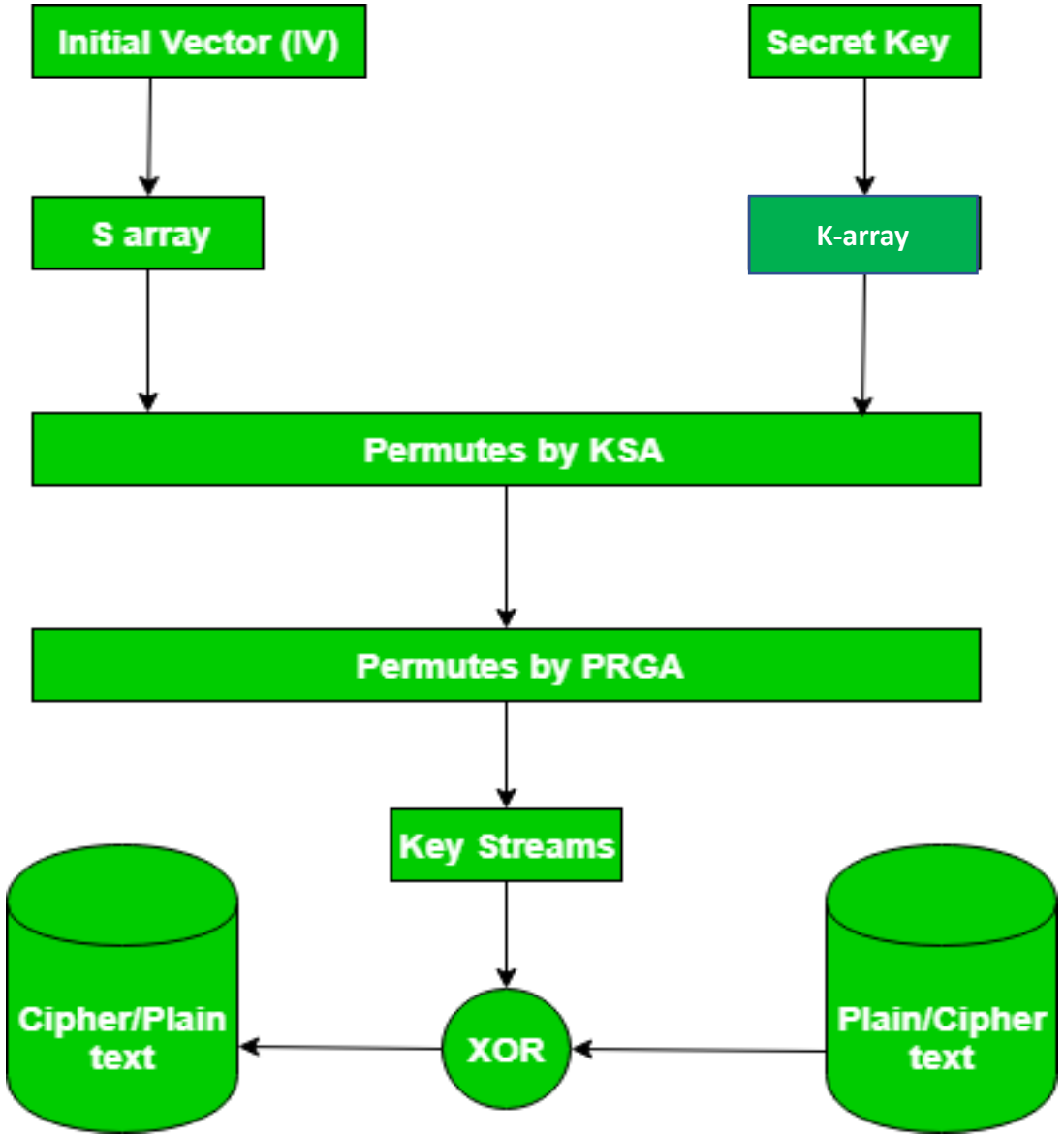
RC4 …Inside

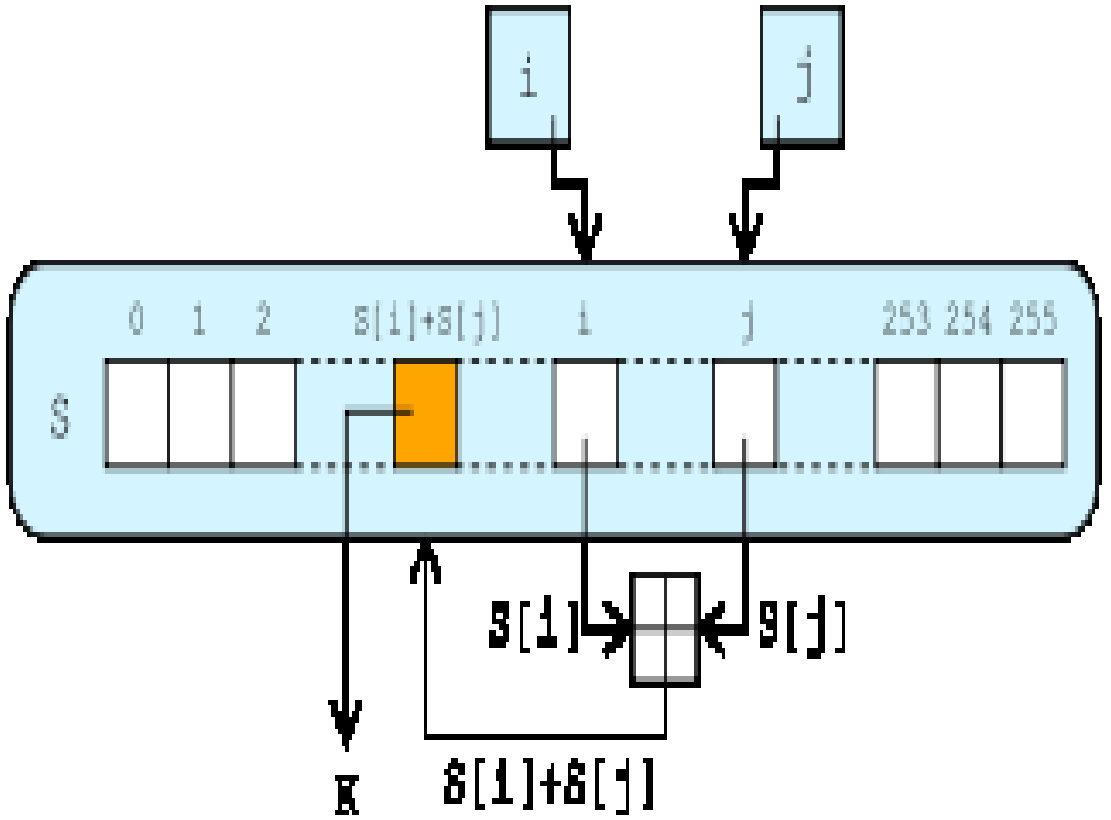
KSA

PRGA

- Consists of 2 parts:
  - Key Scheduling Algorithm (KSA)
  - Pseudo-Random Generation Algorithm (PRGA)
- KSA
  - Used to **generate State array** by **applying a permutation** using variable length **key** consisting of **0 to 256 bytes** .
  - **State vector is** identified as **S[0],S[1]….S[255]** is initialized with **{0,1,2…255}. The key K[0],K[1]….K[255] can be of any length from 0 to 256 bytes and is used to initialize permutation S.** Each K[I] and S[I] is a **byte.**
- PRGA on the KSA
  - Used to **generate keystream byte from State vector array after one more round of permutation.**
- XOR keystream generate using PRGA with the plaintext to generated encrypted stream
- The stream of bits is generated using the pseudo-random generation algorithm.

RC4





Lookup stage of RC4.

Key Schedule Algorithm

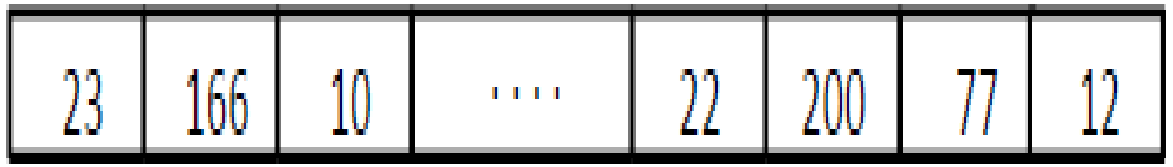
- KSA is a self-modifying lookup table. KSA initializes S[] with a permutation of the byte values 0,1,...,255 using Key. Key is of length between 1 and 256 bytes.
- K is a temporary array, if length of key is 256 bytes copy it to K else after copying remaining positions of K are filled with repeated Key Values untill full.

Key Schedule Algorithm

S[] is permutation of 0,1,...,255

key[] contains N bytes of key

```
for i = 0 to 255
    S[i] = i
    K[i] = key[i (mod N)] //Selects a keystream byte from table
    i++
j = 0
for i = 0 to 255
    j = (j + S[i] + K[i]) mod 256
    swap(S[i], S[j]) //Swaps elements in current lookup table
i = j = 0
```



Keystream: Pseudo-random generation algorithm (PRGA)

- PRGA modifies the state and outputs a byte of the keystream.

Keystream Generation(i := 0, j := 0)

while Generating Output:

i = (i + 1) mod 256

j = (j + S[i]) mod 256

swap(S[i], S[j])

t = (S[i] + S[j]) mod 256

keystreamByte = S[t]

- At each iteration, swap elements in table and select keystream byte
- **Note:** first 256 bytes should be discarded. Otherwise, related key attack exists

Encryption & Decryption in RC4

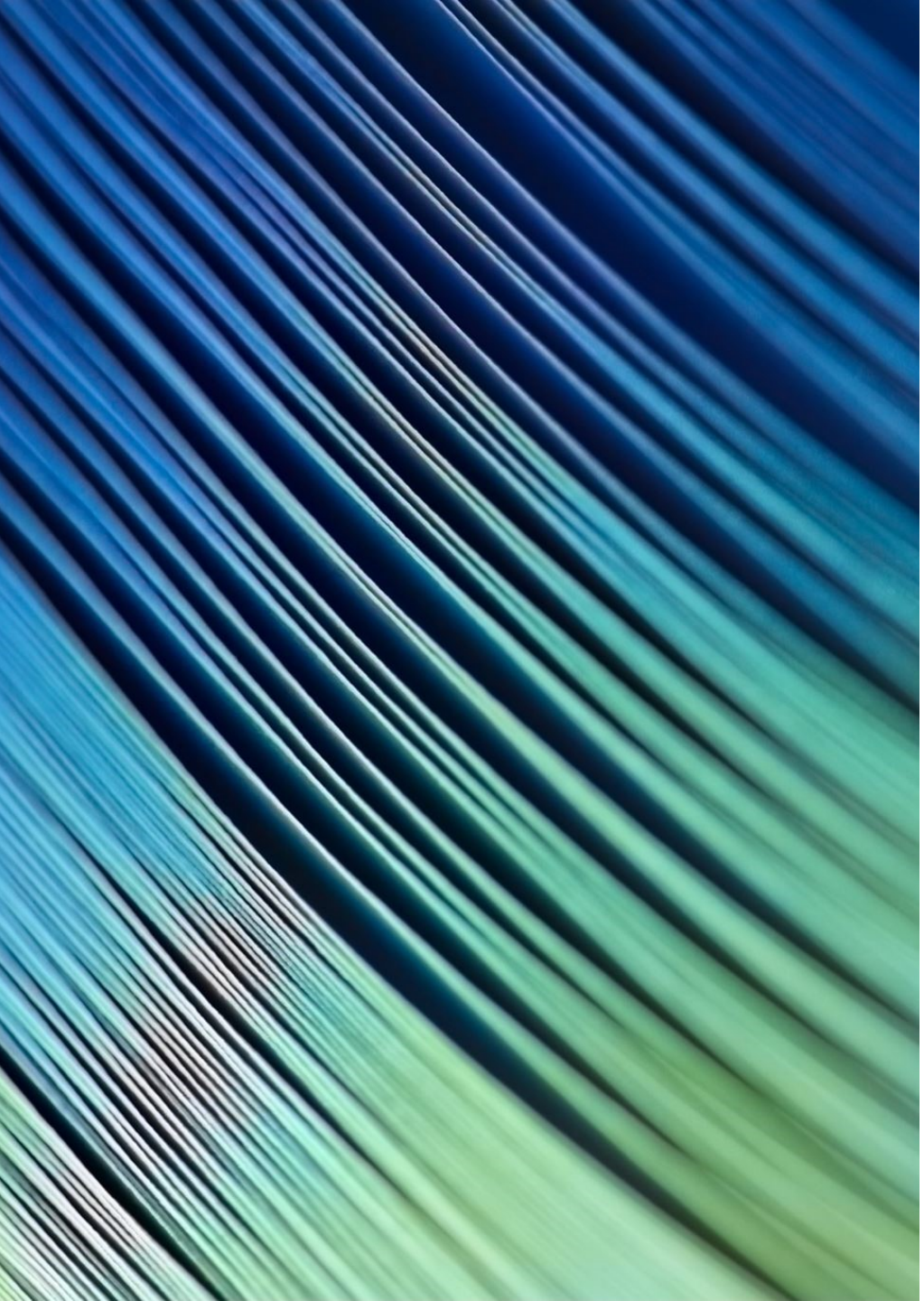
Encryption

- $C_i = M_i$ XOR S[k];

Decryption

Use the same secret key as during the encryption phase.

- $M_i = C_i$ XOR S[k];



EXAMPLE

Assume we use a state vector S is 8 × 3-bits and 4 x 3-bit Key, and plaintext P of 3-bits since S can take the values 0 to 7, which can be represented as 3 bits.
Given:

Key = [1 2 3 6] and P = [1 2 2 2]

EXAMPLE-KSA Algorithm

S = [0 1 2 3 4 5 6 7]

K = [1 2 3 6 1 2 3 6]

- Initially S = [0 1 2 3 4 5 6 7]
- For i = 0: j=0 and then j=1 <––(j + S[i] + K[i]) mod 8=0+0+1 mod 8) Swap(S[0],S[1]); S = [1 0 2 3 4 5 6 7]
- For i = 1: j=1 and then j=3 <––(j + S[i] + K[i]) mod 8=1+0+2 mod 8) Swap(S[1],S[3]); S = [1 3 2 0 4 5 6 7];
- For i = 2 :j=3 and then j=0 <–(j + S[i] + K[i]) mod 8=3+2+3 mod 8) Swap(S[2],S[0]); S = [2 3 1 0 4 5 6 7];
- For i = 3:j=0 and then j=6<––(j + S[i] + K[i]) mod 8=0+0+6 mod 8) Swap(S[3],S[6]) S = [2 3 1 6 4 5 0 7];
- For i = 4:j=6 and then j = 3<––(j + S[i] + K[i]) mod 8=6+4+1 mod 8) Swap(S[4],S[3]) S = [2 3 1 4 6 5 0 7];
- For i = 5:j=3 and then j= 2<––(j + S[i] + K[i]) mod 8=3+5+2 mod 8) Swap(S[5],S[2]); S = [2 3 5 4 6 1 0 7];
- For i = 6: j= 2 and then j = 5<––(j + S[i] + K[i]) mod 8=2+0+3 mod 8) Swap(S[6],S[5]) S = [2 3 5 4 6 0 1 7];
- For i = 7: j=5 and then j = 2<––(j + S[i] + K[i]) mod 8=5+7+6 mod 8) Swap(S[7],S[2]) S = [2 3 7 4 6 0 1 5];
- our initial permutation of S gives: S= [2 3 7 4 6 0 1 5];

EXAMPLE-PRGA Algorithm

First iteration:

- S = [2 3 7 4 6 0 1 5]
- i = (0 + 1) mod 8 = 1
- j = (0 + S[1]) mod 8 = 3
- Swap(S[1],S[3]) S = [2 4 7 3 6 0 1 5]
- t = (S[1] + S[3]) mod 8 = 7
- k = S[7] = 5 This is the 1st keystream bit generated
- P = [1 2 2 2] .first 3-bits of ciphertext is obtained by:
- k XOR P1
- 5 XOR 1 = 101 XOR 001 = 100 = 4

EXAMPLE-PRGA Algorithm

Second iteration:

- S = [2 4 7 3 6 0 1 5]
- i = (1 + 1) mod 8 = 2
- j = (3 + S[2]) mod 8 = 2
- Swap(S[2],S[2])
- S = [2 4 7 3 6 0 1 5]
- t = (S[2] + S[2]) mod 8 = 6
- k = S[6] = 1
- Second 3-bits of ciphertext are: 1 XOR 2 = 001 XOR 010 = 011 = 3

EXAMPLE-PRGA Algorithm

Third iteration:

- S = [2 4 7 3 6 0 1 5]
- i = (2 + 1) mod 8 = 3
- j = (2 + S[3]) mod 8 = 5
- Swap(S[3],S[5])
- S = [2 4 7 0 6 3 1 5]
- t = (S[3] + S[5]) mod 8 = 3
- k = S[3] = 0
- Third 3-bits of ciphertext are: 0 XOR 2 = 000 XOR 010 = 010 = 2

EXAMPLE-PRGA Algorithm

Final iteration:

- S = [2 4 7 0 6 3 1 5]
- i = (1 + 3) mod 8 = 4
- j = (5 + S[4]) mod 8 = 3
- Swap(S[4],S[3])
- S = [2 4 7 6 0 3 1 5]
- t = (S[4] + S[3]) mod 8 = 6
- k = S[6] = 1
- Last 3-bits of ciphertext are: 1 XOR 2 = 001 XOR 010 = 011 = 3

EXAMPLE-PRGA Algorithm

RC4:

- P = [1 2 2 2]
- K = [5 1 0 1]
- C = [4 3 2 3]
- In binary:
- P = 001010010010
- K = 101001000001
- C = 100011010011

RC4 Challenge

- **Choose *n*=3 and Key= 011001100001101_2 and Plaintext is “hsk”.**

RC4 REVIEW

- A PRNG expands a short random seed into a long string that “looks random”
- A PRNG-based stream cipher has fundamental weaknesses
  - same stream cannot be used twice
  - highly malleable
- RC4 is no longer considered secure.
- Not tamper resistant.
  - **Solution**: use a MAC.
- XOR of ciphertexts with same key yields XOR of plaintexts.
  - **Solution**: hash key with nonce.

RC4

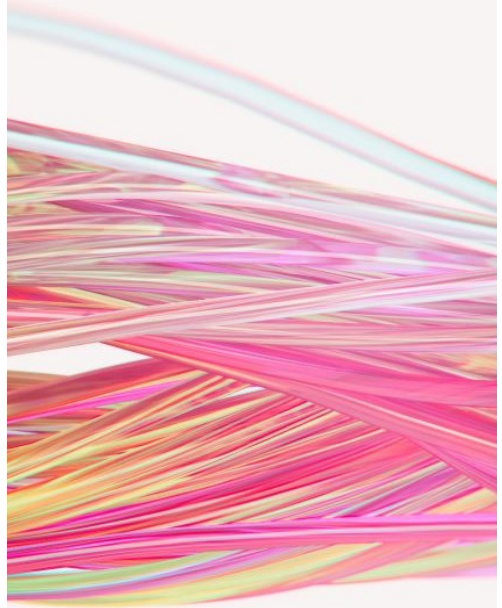
- Each step of RC4 produces a byte
- Efficient in software
- •Used in SSL/TLS, Bit Torrent Protocol Encryption, Microsoft Point-to-Point Encryption

A5/1

- Each step of A5/1 produces only a bit
- Efficient in hardware
- Used in GSM phones

Stream Ciphers

- Stream ciphers were popular in the past
  - Simple and fast
  - Not very secure
  - Efficient in hardware
  - Speed was needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - Shamir declared “the death of stream ciphers”



Stream cipher

- stream ciphers process messages a bit or byte at a time when en/de-crypting
- Each block of plaintext is encrypted with a different key.
- Identical bits of text get encrypted in the same way if same key is used

Eg: A5/1, RC4

Block Cipher

- block ciphers process messages in blocks, each of which is then en/decrypted
- Plaintext divided into blocks and each block encrypted with the same key. Blocks can vary in length starting from 1 character
- Identical blocks of text do not get encrypted the same way in a message

Eg: DES, AES

BLOCK CIPHERS

BLOCK CIPHER

- A block cipher is a method of producing ciphertext in which a **cryptographic key (K**)and **symmetric algorithm(function F**) are applied **to a fixed size block of data (for example, 64 contiguous bits)** at once rather than to one bit at a time in several rounds.
- Ciphertext is obtained from plaintext by **iterating a function F** over **some number of rounds**, and always it **depends on the output of previous round and key K**. Hence the function is known as **round function.**
- Decryption algorithm must be the inverse of encryption algorithm and should use same key
- Eg: AES,DES,BLOWFISH,IDEA
- A block cipher is designed based on three critical aspects:
  - **Number of Rounds**
  - **Design of Function F**
  - **Key Schedule Algorithm**

Block Cipher Features

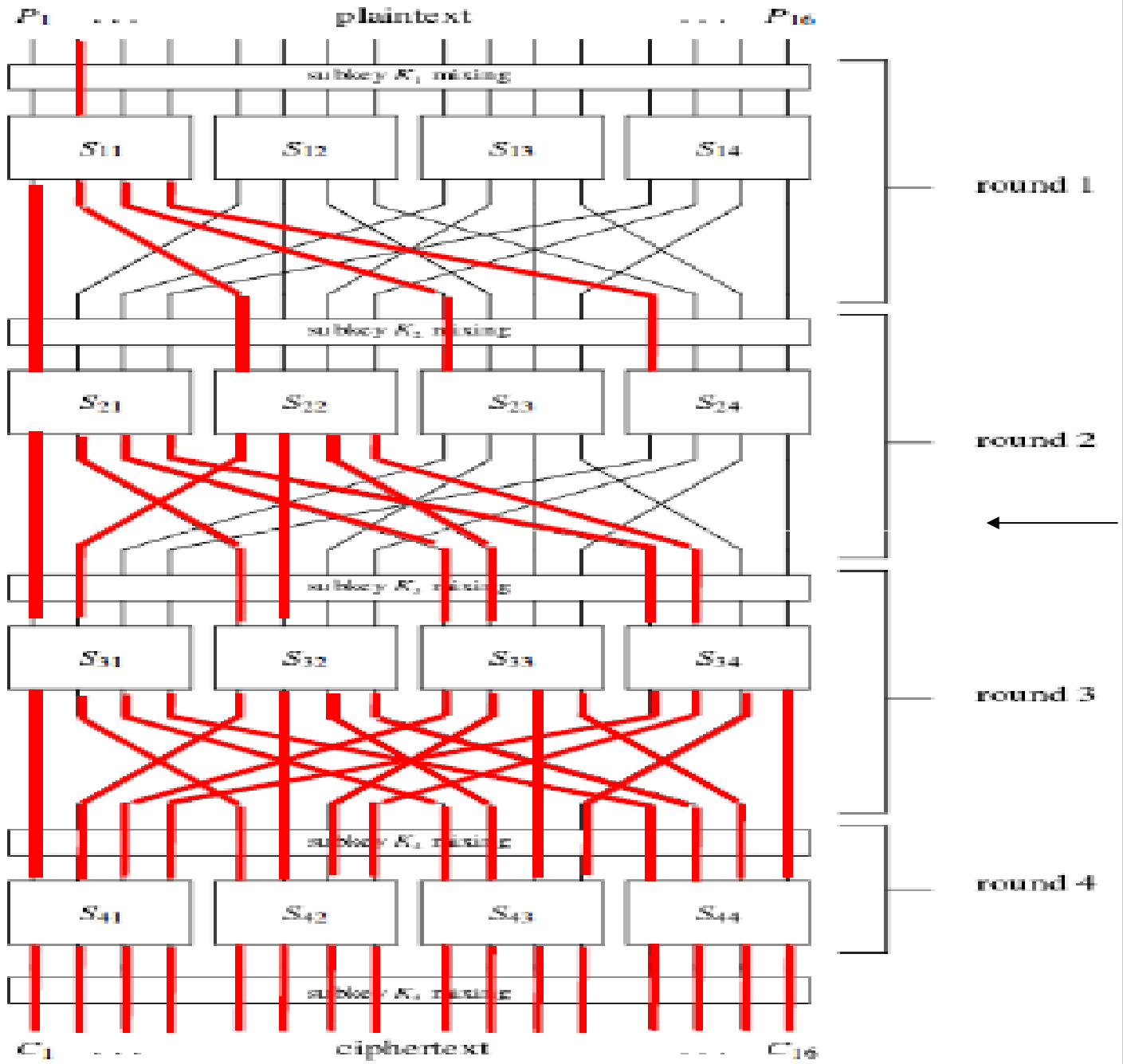
- **Identical blocks of text do not get encrypted the same way in a message** (Block ciphers are used in chaining mode).
- It offers Security and efficiency.(as stream ciphers are more time consuming)
- **Block ciphers** uses both confusion and diffusion techniques.

BLOCK SIZE

- Avoid very small block size and very large block size
- Multiples of 8 bit
- The choice of block size does not directly affect to the strength of encryption scheme. The **strength of cipher** depends up on the **key length.**
- The size of block is fixed in the given scheme.
- ***Initialization vector*** derived from a *random number generator* is the key that is combined with the text in the first block. Subsequent keys are generated from IV by applying round function. This ensures that all subsequent blocks result in ciphertext that doesn't match that of the first encryption.
- If last block may not be having same size as that of initial blocks then extra bits are added to make it equivalent to other blocks. This is referred to as **padding**.

Block Cipher Components

- **S boxes(Substitution Box)** – Replaces its input with another. Its invertible lookup table which has non-linear functions that depends on key
- **P boxes(Permutation Box)** – reorder bits (may also depend on key)
- **Key schedule** – function of key (e.g. bit selection or simple hash)

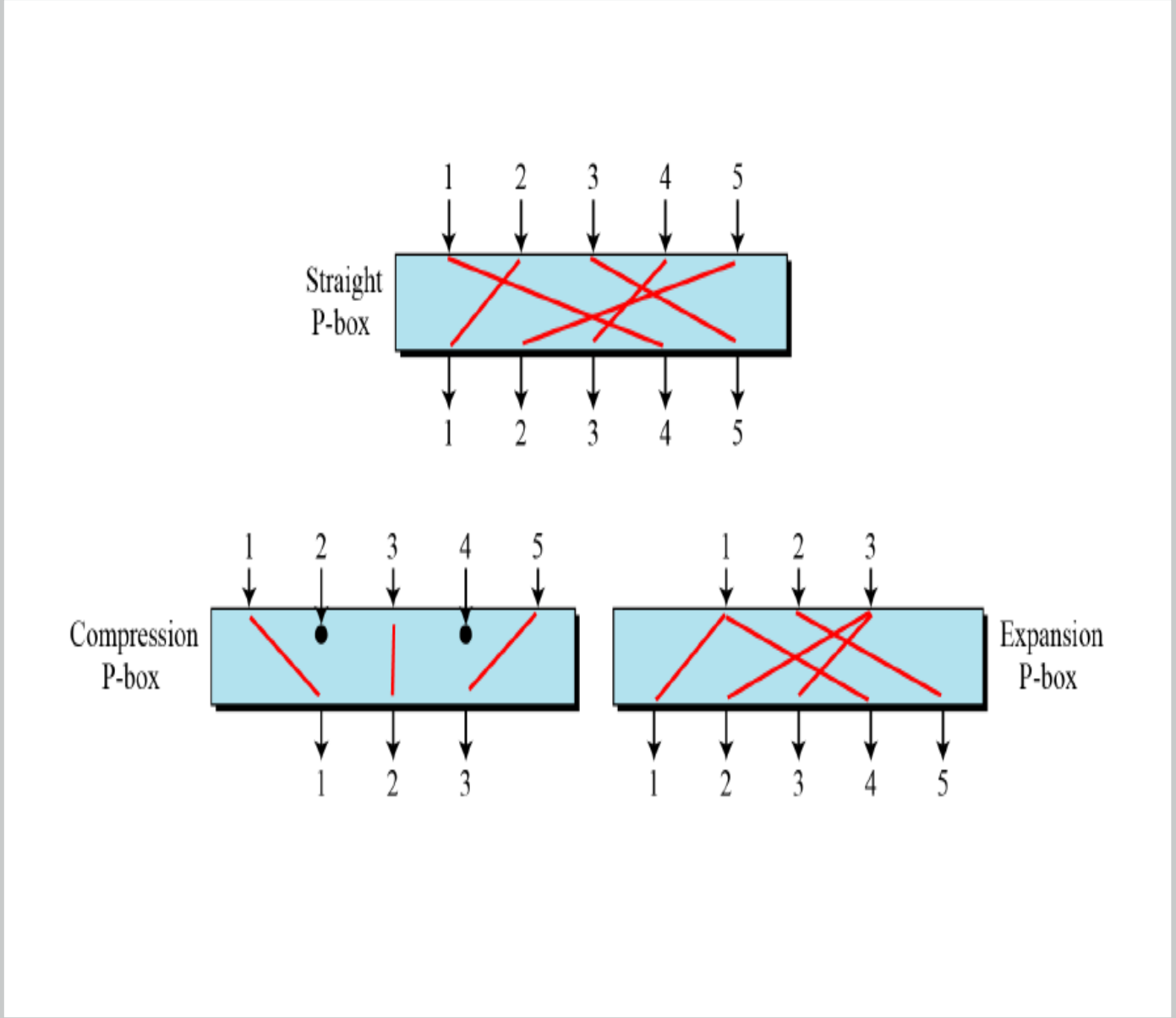


Key Schedule

Types of P boxes

- Types of P-Boxes
  1. **straight P-box (mxm)**
  2. **expansion P-box (mxn, m<n)**
  3. **compression P-box (mxn, m>n)**
- Common **permutation operations** which are used in block ciphers are

1. **Circular shift:** Circular shift input N bits to right (or left)
2. **Swap:** Special case of circular shift with shift = N/2



How confusion and diffusion is achieved?

- **Confusion** achieved by small **substitution functions**
- If, even a single bit of the key is wrong, half the bits of the ciphertext is flipped
- **Diffusion** achieved by **diffusion functions**
  - Permutations
  - Linear Transformations
- A single bit of plaintext gets diffused to all bits of the ciphertext. If a single bit in the plaintext is flipped
  - Each bit of the ciphertext will flip with probability 1/2
  - In other words, half the bits of the ciphertext will flip.

Classes of Block Ciphers

- Modern block ciphers are divided into two classes.
  - Feistel ciphers

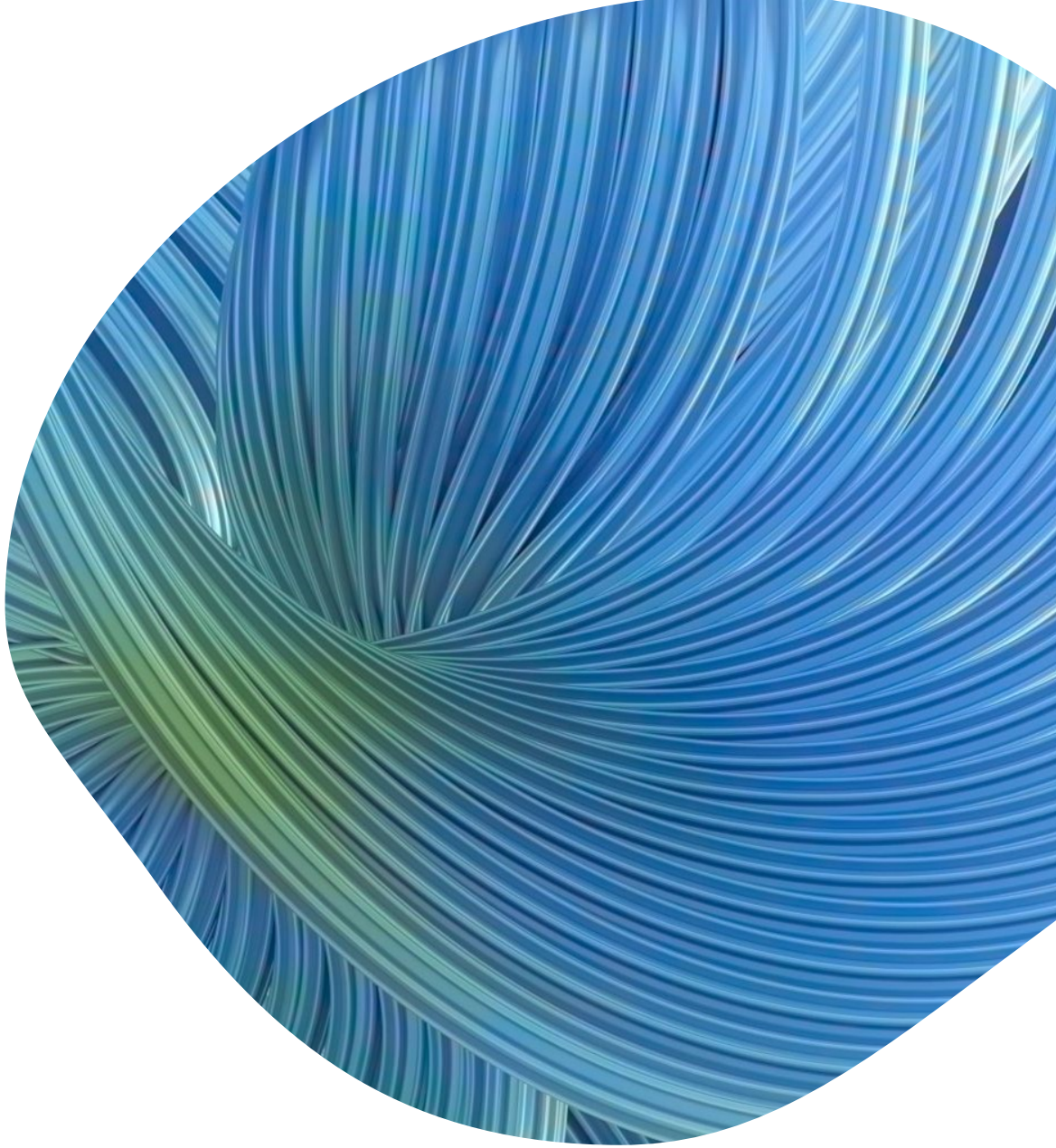
Eg: DES, CLEFIA, SERPENT, RC5, …

  - Non-Feistel ciphers(Substitution-Permutation Networks)

Eg: AES, PRESENT, SHARK

Block cipher design principles

- Block size
- Key size
- No of rounds*
- Subkeys count*
- Round function*
- Plain text in 2 equal halves



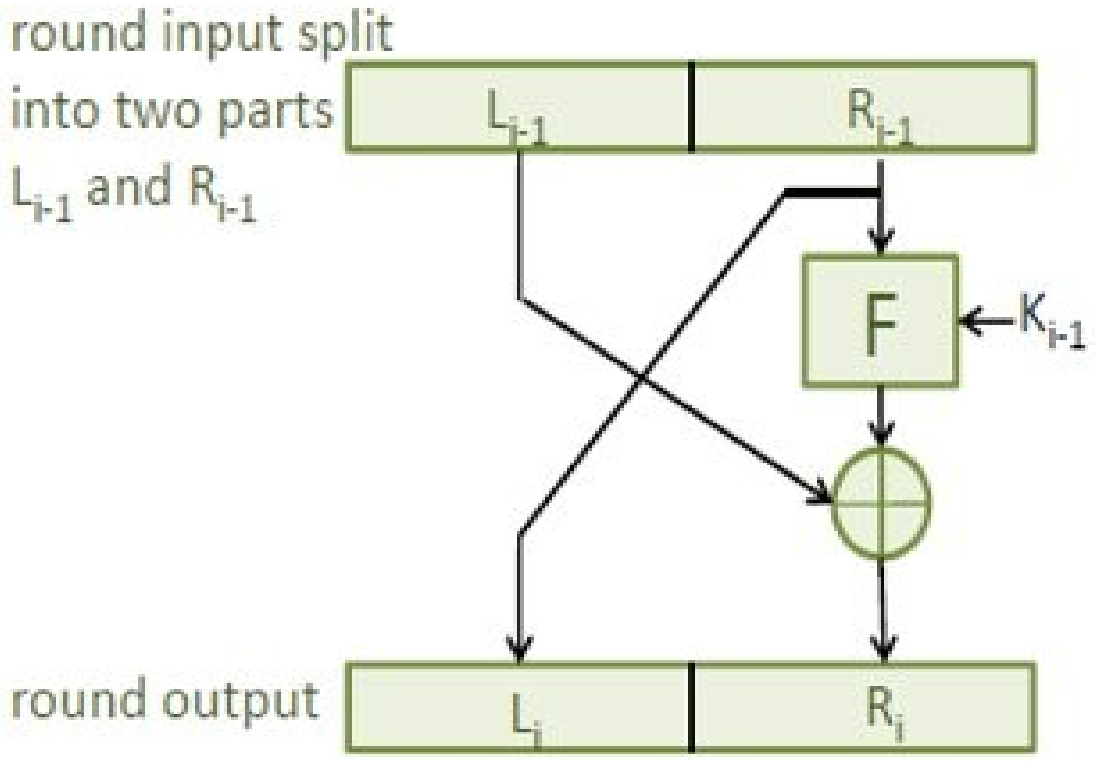
Feistel Cipher

Feistel Cipher

- Feistel cipher is a **cipher design principle** based on **Feistel cipher structure**. Eg: DES, IDEA, RC5
- It uses the same algorithm for both encryption and decryption.
- A Feistel Network is fully specified given
  - Block size: n = 2w
  - Number of rounds: d
  - d Round functions
- Uses the Feistel structure consisting multiple rounds(depending on algorithm) of processing of the plaintext, each round consisting of a “substitution” step followed by a “permutation” step.
- Permutation step at the end of each round swaps the modified L and unmodified R.
- For encryption the right piece of one round becomes the left piece of the next. The left piece is x-ored with the result of performing the function F on the right piece.

Horst Feistel

Feistel Cipher Structure for Single Round



Encryption:

$L_i = R_{i-1}$

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

Decryption:

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

$R_{i-1} = L_i$

FEISTEL CIPHER

ENCRYPTION PROCESS

1.Alice and Bob exchange a secret key, S, through a secure channel

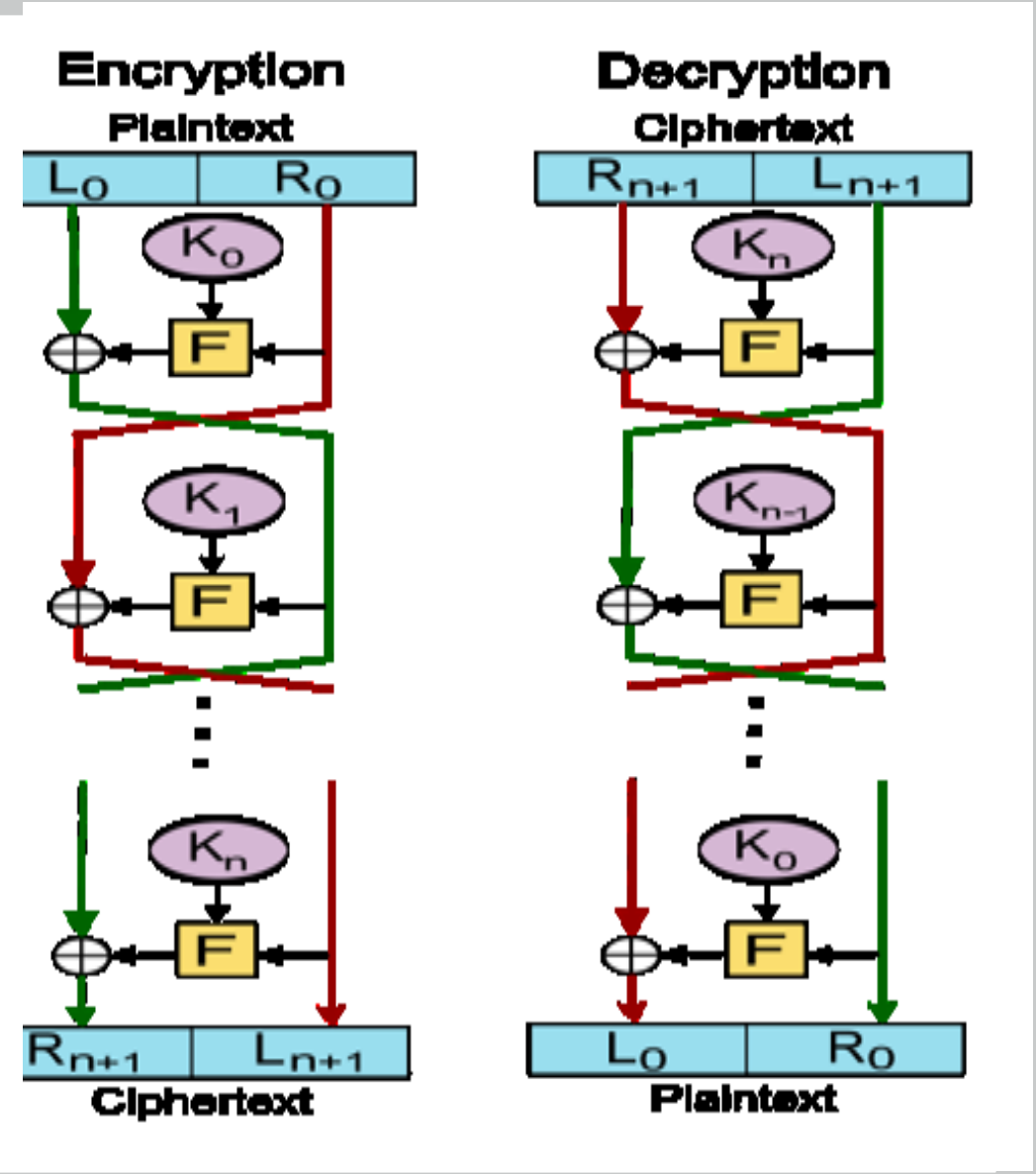
2.Alice selects a plaintext, P, to send to Bob and breaks it into blocks of the length that the cipher acce
For each block, the following steps are followed.

3.Plain text is split into left and right halves $P=(L_0,R_0)$

4.Alice sets the value of round key zero to the initial secret key. That is, $K_0 = S$. where, K is Round Key.
Let F be the round function used for encryption and let $K_0,K_1,\ldots ,K_n$ be the sub-keys for the rounds $0,1,\ldots ,n$ respectively. Sub-key is derived from key K according to a specific Key Schedule Algorithm.

5.For each round $0,1,\ldots ,n$ perform $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus F(R_{i-1}, K_{i-1})$

6.Finally ciphertext $C=(L_n,R_n)$ All ciphertext blocks are combined to a single ciphertext C. Alice sends the result C to Bob.



FEISTEL CIPHER

PROBLEM

- Consider a block cipher using 8-bit blocks that is based on the basic Feistel architecture . The scrambling function for round i is $f_i(x, K) = (2i \cdot K)^x \bmod 15$, for i = 1, 2, where the key K is a member of Z15. If K = 7 and the plaintext is 00101000, what is the ciphertext? Draw the picture of the Feistel Cipher network and show your intermediate results

SOLUTION

- The computation of $f_i(x)$ in the i th round is $(2i \cdot 7)^x \bmod 15$.
- $f_i(x, K) = (2i \cdot K)^x \bmod 15$.
- $f_1(8, 7) = (2*1 \cdot 7)^8 \bmod 15 = 1$
- $f_2(3, 7) = (2*2 \cdot 7)^3 \bmod 15 = 7$

- So the ciphertext is 00111111.

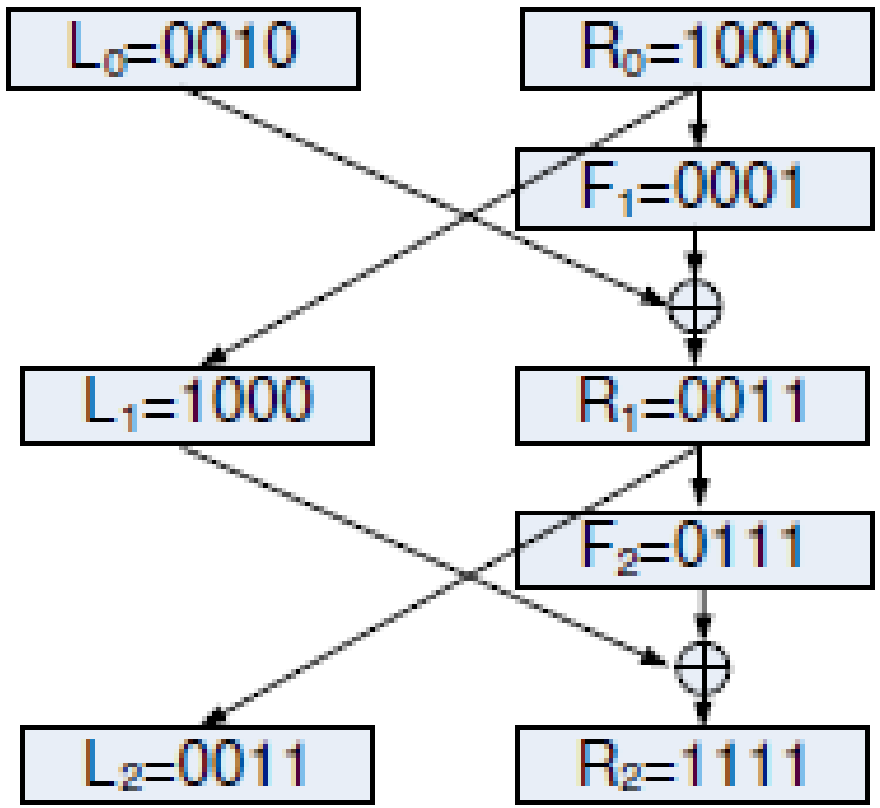


Figure 1: Feistel Network

FEISTEL CIPHER

DECRYPTION PROCESS

In decryption, we use the round keys in reverse.

1.Alice and Bob exchange a secret key, S, through a secure channel and Alice sends Bob a ciphertext, ***C***.

2.Bob calculates the round keys for all rounds using the key scheduling functions ***K***.

3.Bob breaks ***C*** into blocks of the length that the cipher accepts. For each block, the following steps are followed.

4.Bob splits the ciphertext block into a left piece and a right piece, L_n and R_n.

5.For each round ***0,1,…. ,n*** right side of round ***i*** is simply the left side of round ***i+1***. where $R_{i-1} = L_i$ and $L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$. This works because the xor-function ($\oplus$) has the property that A = B $\oplus$ C implies B = A $\oplus$ C.

6.Finally plain text $P=(L_0,R_0)$ and L_0 and R_0 are combined to create the plaintext block for this ciphertext block. All plaintext blocks are combined to a single plaintext ***P***.

PROBLEM

- Consider a block cipher using 8-bit blocks that is based on the basic DES architecture (Feistel network) with two rounds and no initial or final permutation. The scrambling function for round i is $f_i(x, K) = (2i \cdot K)\, x \bmod 15$, for i = 1, 2, where the key K is a member of Z15. If K = 7 and the ciphertext is 00111111, what is the plaintext?

SOLUTION

Decryption:

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

$R_{i-1} = L_i$

Given ***L2= 0011 and R2= 1111***

R1 = L2 =0011

L1 = R2 ⊕ F(R1, K2)= 1111 ⊕ 0111=1000

- The coputation of $f_i(x)$ in the i th round is $(2i \cdot 7)^x \bmod 15$.
- $f_i(x, K) = (2i \cdot K)^x \bmod 15$.
- $f_2(3, 7) = (2*2 \cdot 7)\ 3 \bmod 15 = 6$
- $f_2(3, 7) = (2*2 \cdot 7)^3 \bmod 15 = 7$

- So the ciphertext is 00111111.

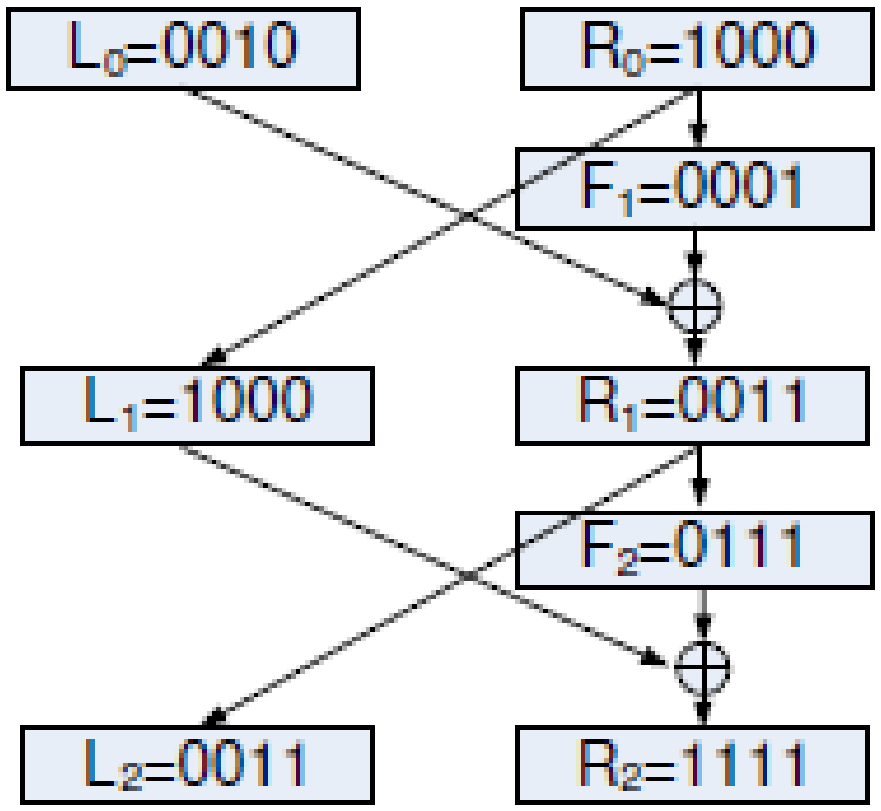


Figure 1: Feistel Network

Security

- Depends on **no of rounds, round function** and **no of keys.**
- Each round use separate sub-keys which should be generated from master key.

Advantages

1.same structure can be used for encryption and decryption as long as the key schedule is reversed for decryption.

Disadvantages

1.Feistel ciphers only change part of the internal state each round so limited ability.

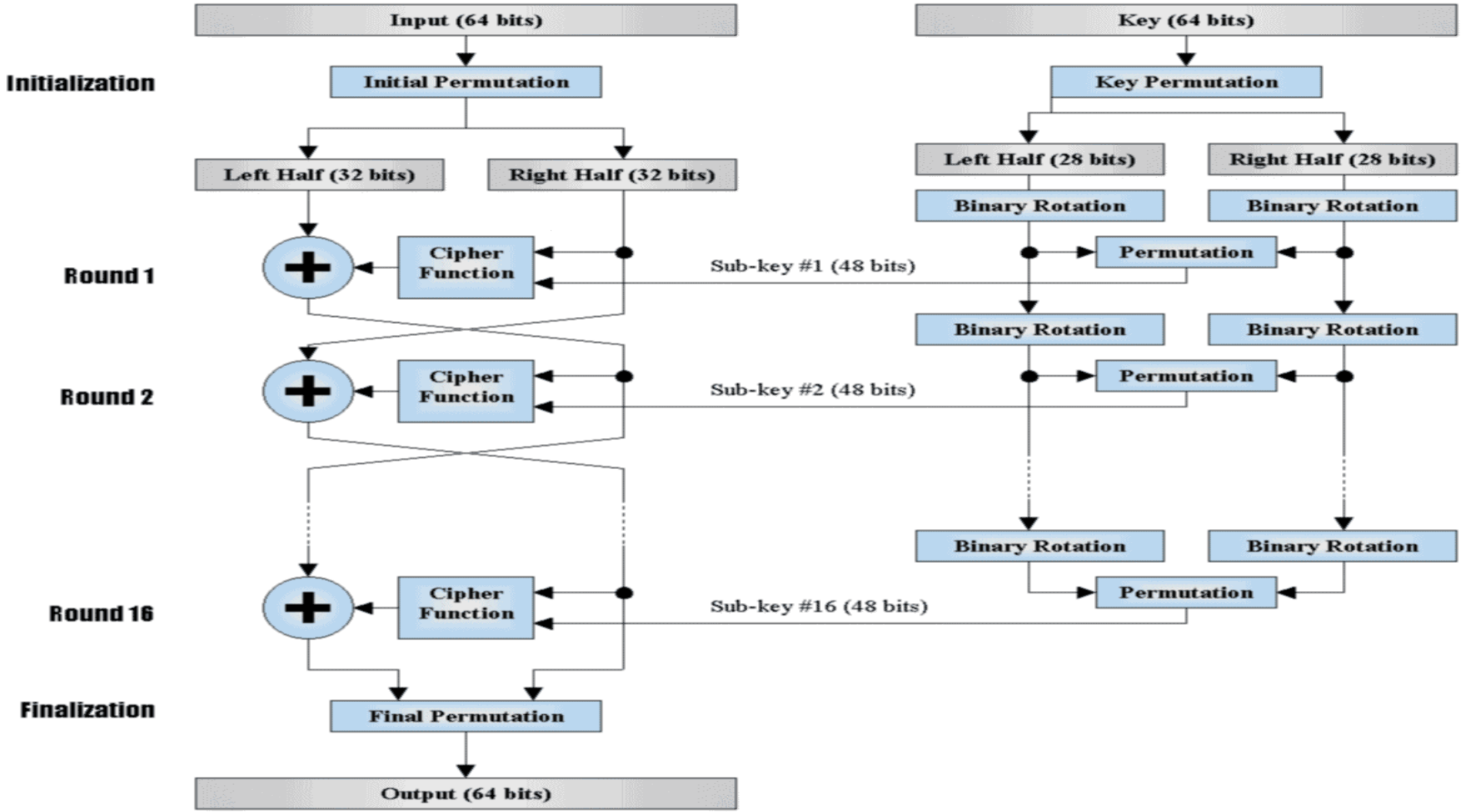
DES

DATA ENCRYPTION STANDARD

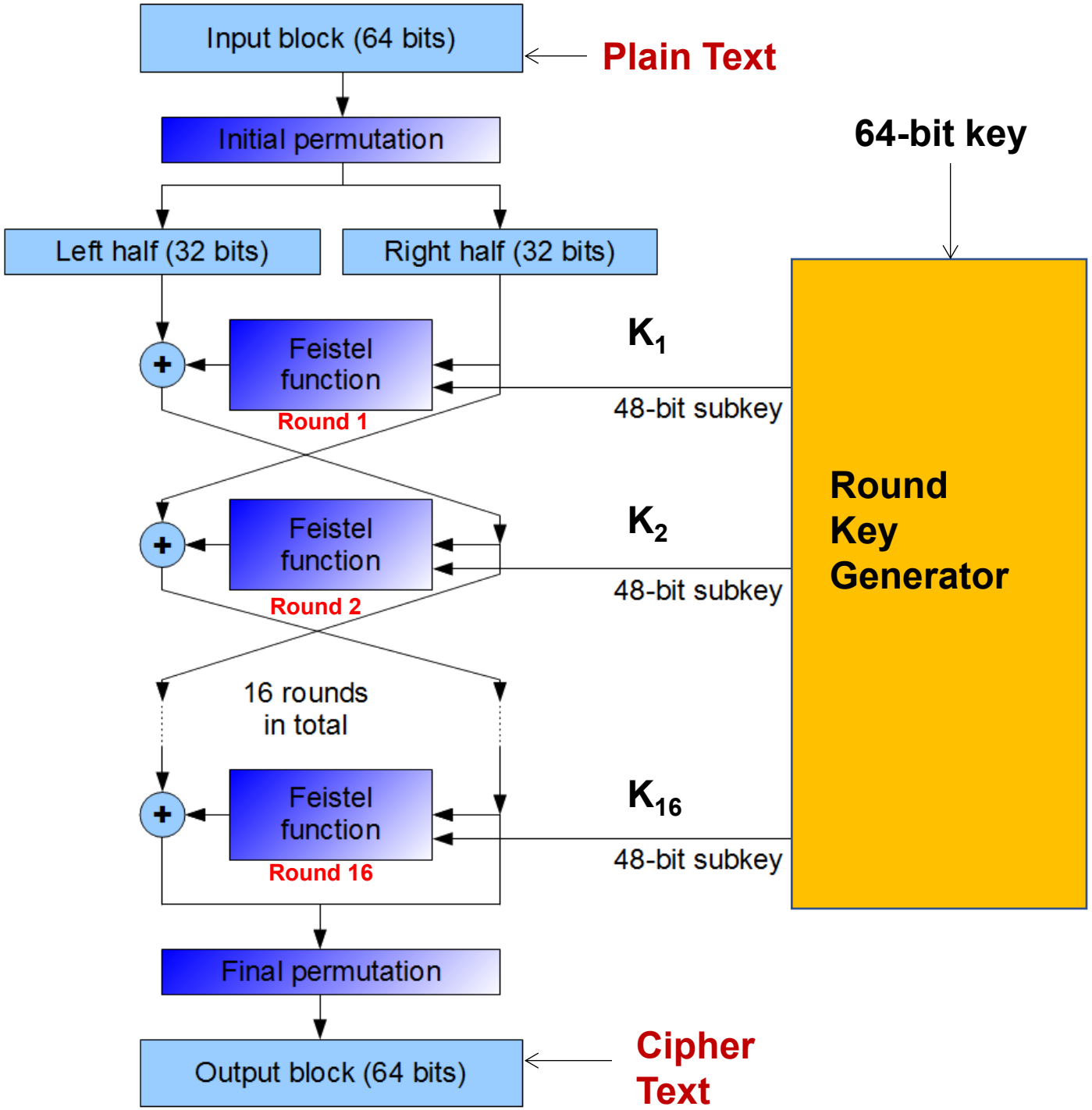
Data Encryption Standard (DES)

- DES is a **symmetric-key block cipher** published by the National Institute of Standards and Technology (NIST) (former NBS) & IBM, 1977
- DES implements **16 round Feistel structure**.
- **Block size(Plain Text)** is **64-bit** and **Key length** is **64-bit**, but effective key length is **56 bits**, since 8(Every 8th bit at positions 8, 16, 24, 32, 40, 48, 56 and 64 of the key is discarded. It facilitates as **parity check bits.**Each **parity-check bit** is the XOR of the previous 7 bits) of the 64 bits of the key are not used by the encryption algorithm.
- 16 intermediary keys are generated by key generation algorithm. **Each round key is of size 48 bits, generated from 56-bit initial key**.
- DES is based on **substitution** (also called as **confusion**) and **transposition** (also called as **diffusion**) at each round.
- **Security:** Security depends heavily on “S-boxes” which maps 6 bits to 4 bits.
  - No longer considered secure: 56 bit keys are vulnerable to exhaustive search

DES in Nutshell



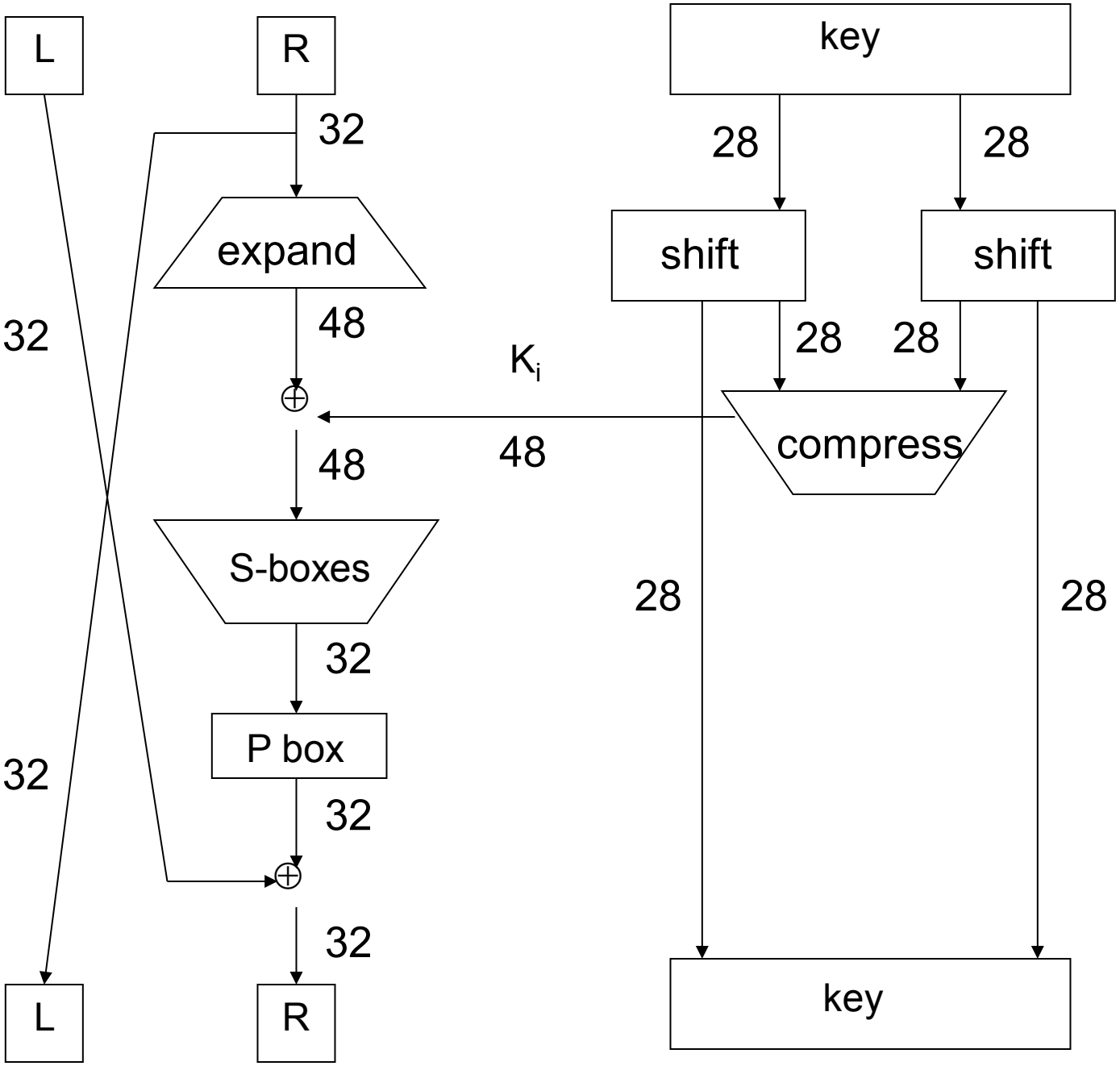
DES


Input block (64 bits)
Plain Text
Initial permutation
64-bit key
Left half (32 bits)
Right half (32 bits)
Feistel function
Round 1
K1
48-bit subkey
Feistel function
Round 2
K2
48-bit subkey
Round Key Generator
16 rounds in total
Feistel function
Round 16
K16
48-bit subkey
Final permutation
Output block (64 bits)
Cipher Text

Basic Steps in DES

1. **64 bit plain text block** is handed over to an **initial Permutation (IP1) function**. The initial permutation performed on plain text.(does not enhance the security of algorithm but introduced to make passing data into encryption machines easier.)
2. **Plain Text** is **divided to two halves** of the permuted block **32-bit each, say Left Plain Text (LPT) and Right Plain Text (RPT).**
3. **Right Plain Text (RPT-32bit)** is **expanded to 48 bits** using **expansion permutation**.
4. **Right plaint text(48 bit)** is **XORed with Key**(48.bit generated in each round of **Key generation algorithm**-(56 bits are selected from the 64-bit key (Permutation PC-1). They are then divided into two 28-bit parts. 48 sub-key bits are selected by Permutation PC-2 for 16 rounds from 56-bits.)
5. Xored 48 bit is **divided into 6-bit blocks(8 blocks)** and is **fed to S-box which generates 4 bits**. To which s-box block has to enter is defined by 1st and last bit
6. The **output of S boxes are combined** which form **32 bit** and undergoes **permutation using P-boxes**.
7. Finally **permuted RPT is XORed with LPT to get RPT for next round**.
8. Each Right Plain Text (RPT) is taken as next round's LPT and RPT for next round will be output of step 7.Now each LPT and RPT to go through 16 rounds of encryption operations called as **Feistel functions**(steps 3,4,5,6) along with 48-bit key.
9. After all 16 rounds LPT and RPT together undergoes final permutation

- **L(i) = R(i-1)**
- **R(i) = L(i-1) ⊕ P(S(E(R(i-1))⊕K(i)))**



One Round of DES

Key:

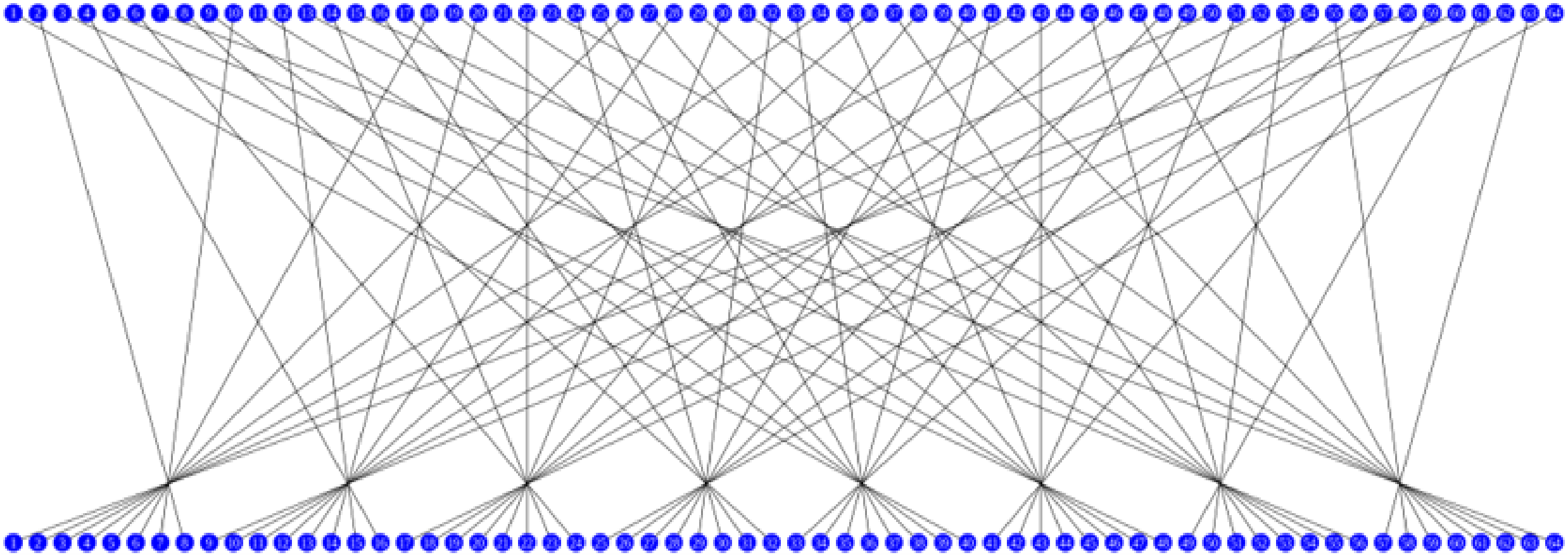
1. Dividing them into 2 halves (28bit each)
2. Bit-shift left
3. Compress the key (56bit → 48bit)

Block of the Plaintext:

1. Dividing them into L_i and R_i (32bit each)
2. Expansion permutation (32bit → 48bit)
3. Addition of subkey (48bit ⊕ 48bit)
4. S-boxes (map 6bits → 4bits)
5. P-box (change permutation)

STEP 1) Initial permutation

- Initial and final permutations are applied on each block of data(64-bit) and are straight Permutation boxes (P-boxes) that are inverses of each other. It rearranges bits in a certain, predefined way. it is a **vector**, not a matrix.
- The meaning is as follows: : the first bit of the output is taken from the 58th bit of the input; the second bit from the 50th bit, and so on, with the last bit of the output taken from the 7th bit of the input.



Initial & Final Permutation Tables

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

3) Expansion Permutation

- Expansion Permutation initiates each round of Feistel functions.
- It expands the **right half** of data from **32 bits** to **48 bits**. 16 bits appear twice, in the expansion.
- 32 bit is divided to 8blocks of 4bits each ,each block is expanded to 6bit by adding 2 bits(1^{st} and 4^{th} bit of each block)

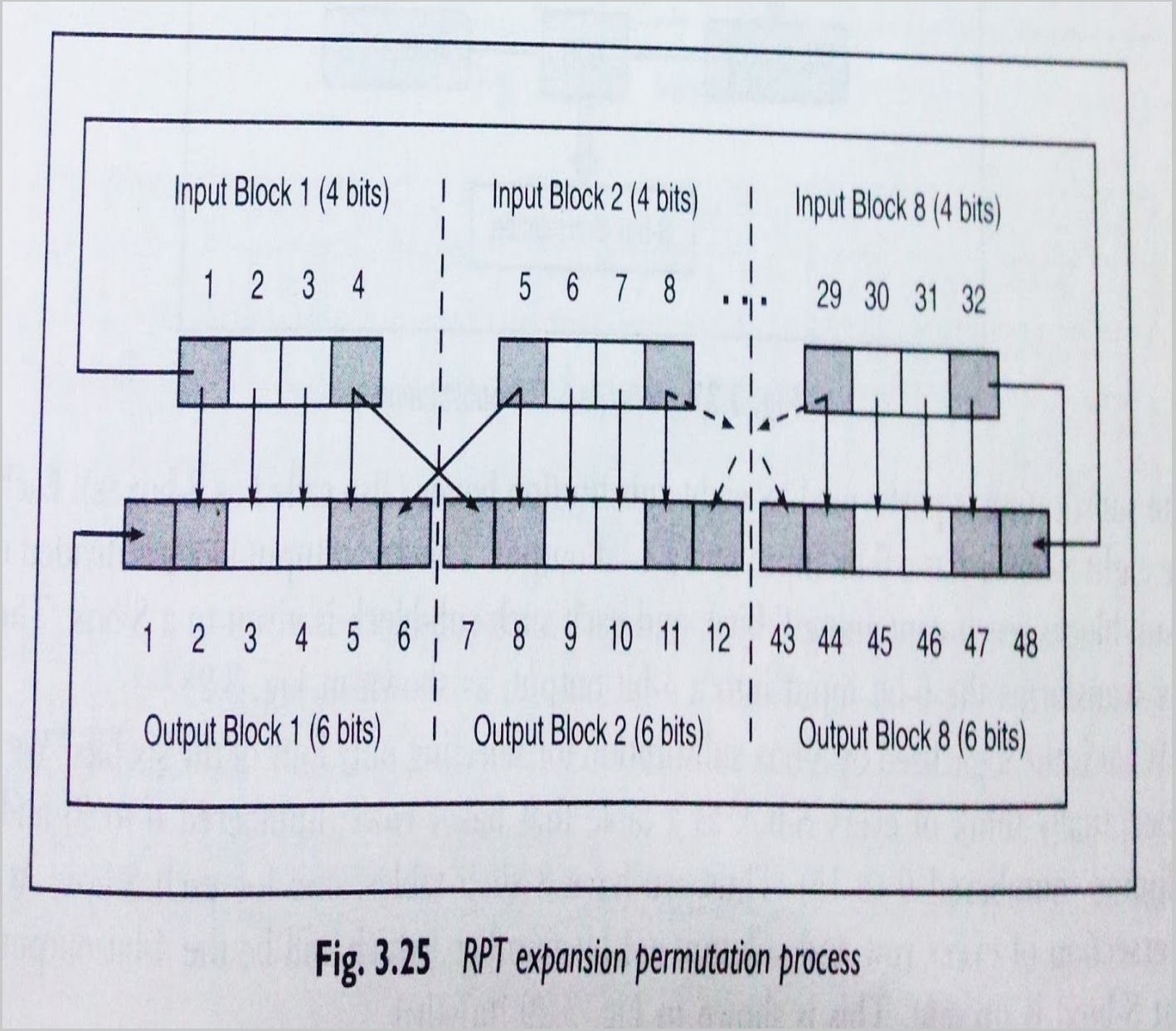


Fig. 3.25 *RPT expansion permutation process*

Expansion D-Box Table

Input 32 bits

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

Output

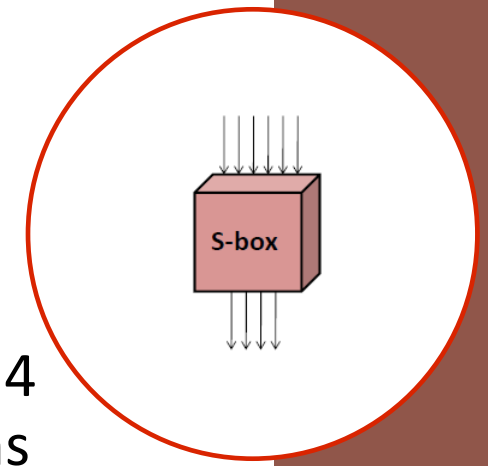
?

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

OUTPUT

5) S-Blocks

- **48 bit** after XOR-operation is **divided to 8 blocks of size 6 bit** and given to each **S-box**. Each S-box is a table that has **4 rows and 16columns and at the intersection we have 4bit output**.
- Each **6-bit input block** is replaced by **4-bit output** using 8 “substitution boxes” or S-boxes or compression functions
- If input bits are marked as a1, a2, a3, a4, a5 and a6, then the a1 and a6 comprise a 2-bit figure standing for a row, and the a2, a3, a4 and a5 comprise a 2-bit figure standing for a column (both columns and rows are numbered from zero). Where the row and column cross, there is the output figure of the block. For example, if the input string of bits is 101010, the output will be 0110.



DES S-Box 1

	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

input bits (0,5)
↓

input bits (1,2,3,4)

b_0b_5	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

PROBLEM

- The input to S-box 1 is 100011. What is the output?

SOLUTION

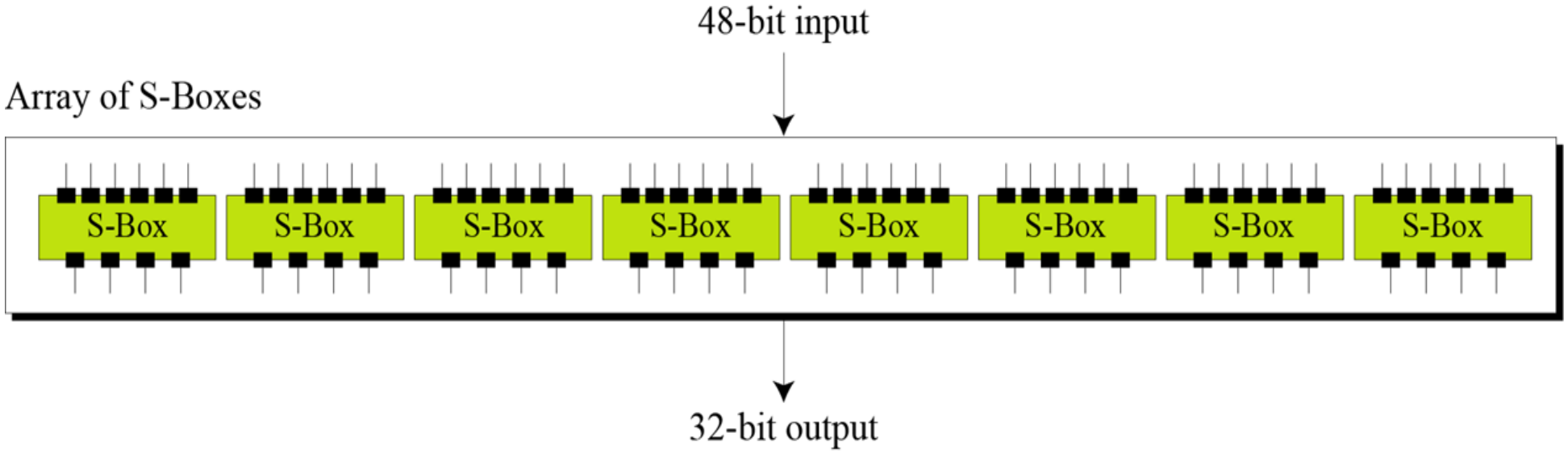
SOLUTION

- If we write the first and the sixth bits together, we get 11 in binary, which is 3 in decimal. The remaining bits are 0001 in binary, which is 1 in decimal. We look for the value in row 3, column 1, in Table(S-box 1). The result is 12 in decimal, which in binary is 1100. So the input 100011 yields the output 1100

The S-boxes

S_i: $\{0,1\}^6 \longrightarrow \{0,1\}^4$

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



8 S-boxes

שורה	מספר עמודה															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S_1															
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	3	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	13	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
	S_2															
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
	S_3															
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
	S_4															
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
	S_5															
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
	S_6															
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
	S_7															
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
	S_8															
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

6) P-Blocks

- **For each round *i* , there are 32 bits output from the S-Boxes, are changed by applying permutation as:**
- **Input 32 bits**

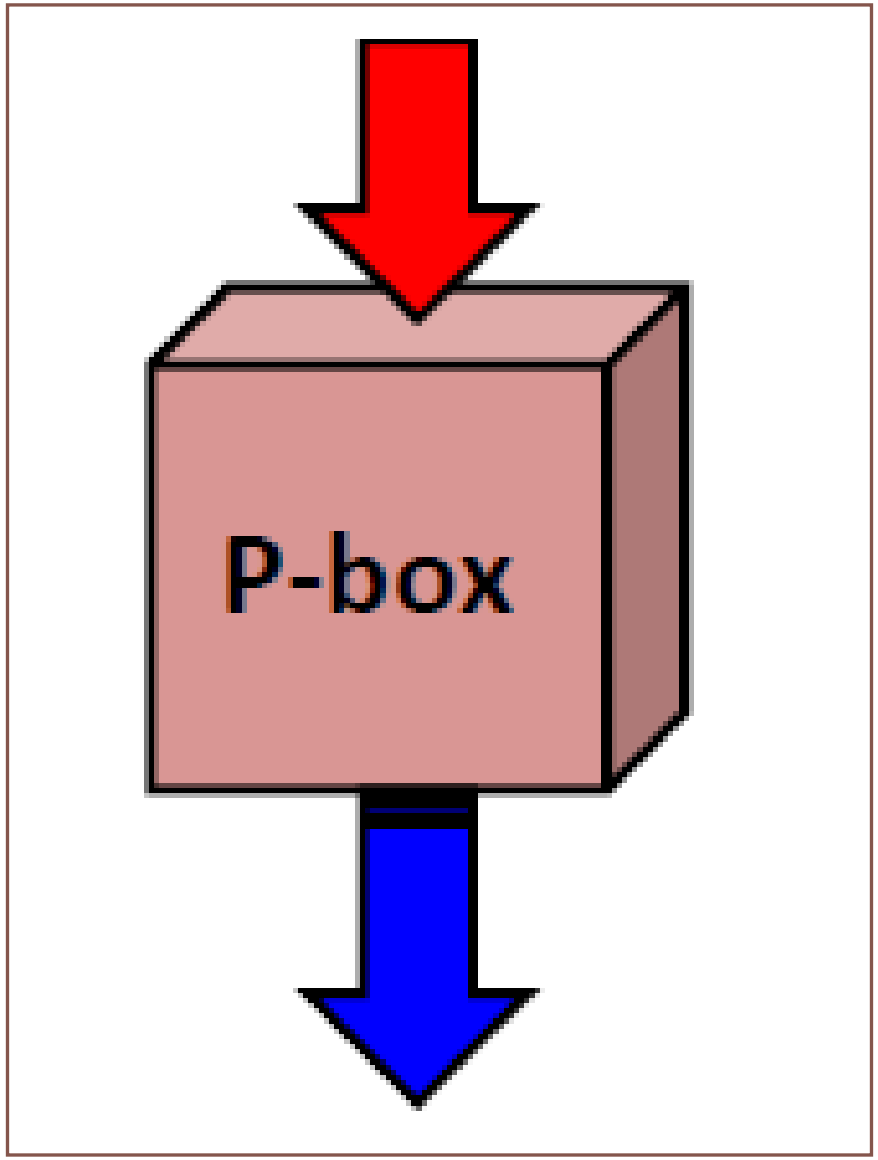
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- **Output 32 bits**

15 6 19 20 28 11 27 16 0 14 22 25 4 17 30 9

1 7 23 13 31 26 2 8 18 12 29 5 21 10 3 24

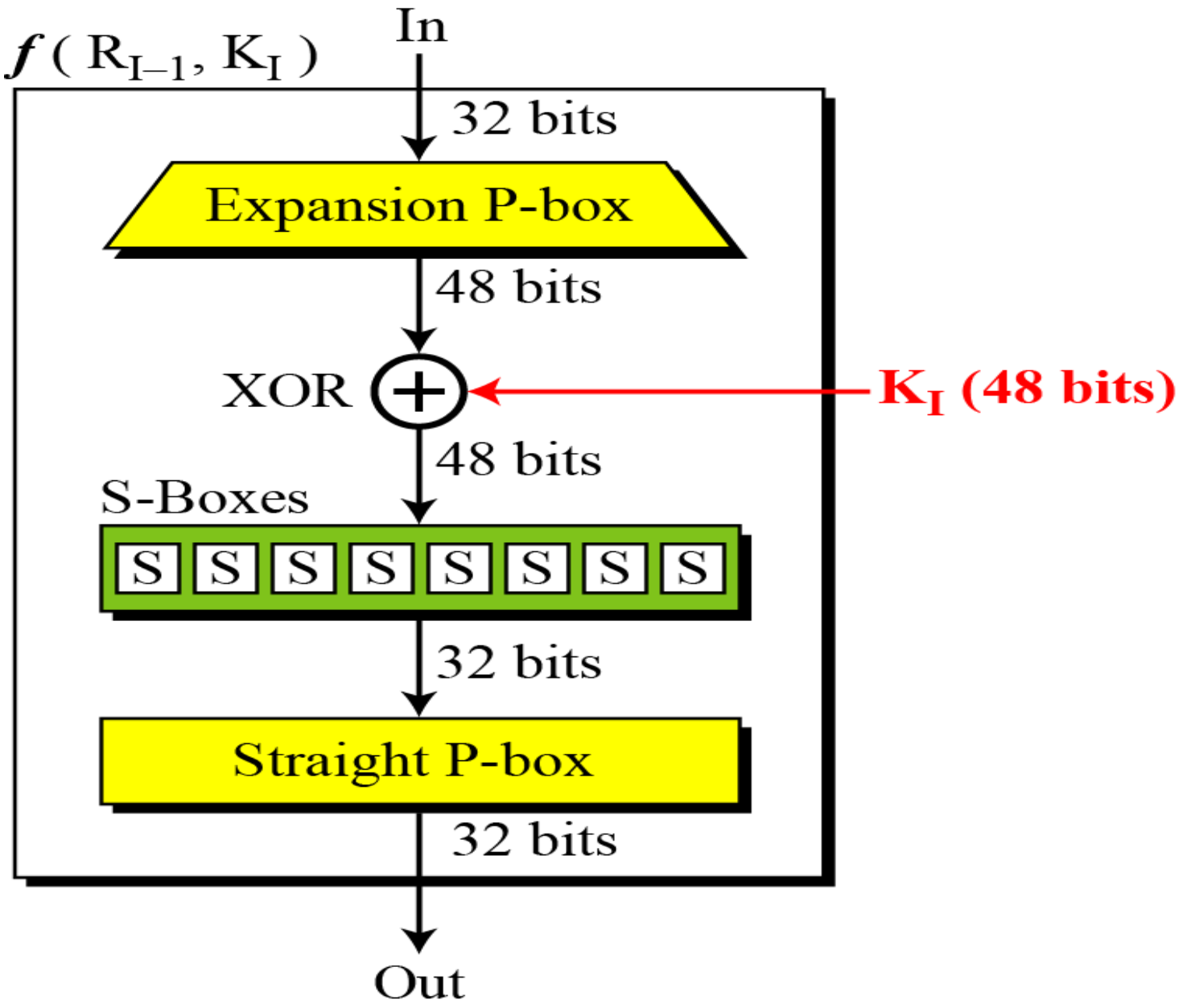


40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Final Permutation

Final Permutation P is processed on the 32-bit output block from eight S-Boxes after the last round of Feistel functions. It is the inverse of the Initial Permutation

Feistel Function: Steps 3,4,5 & 6



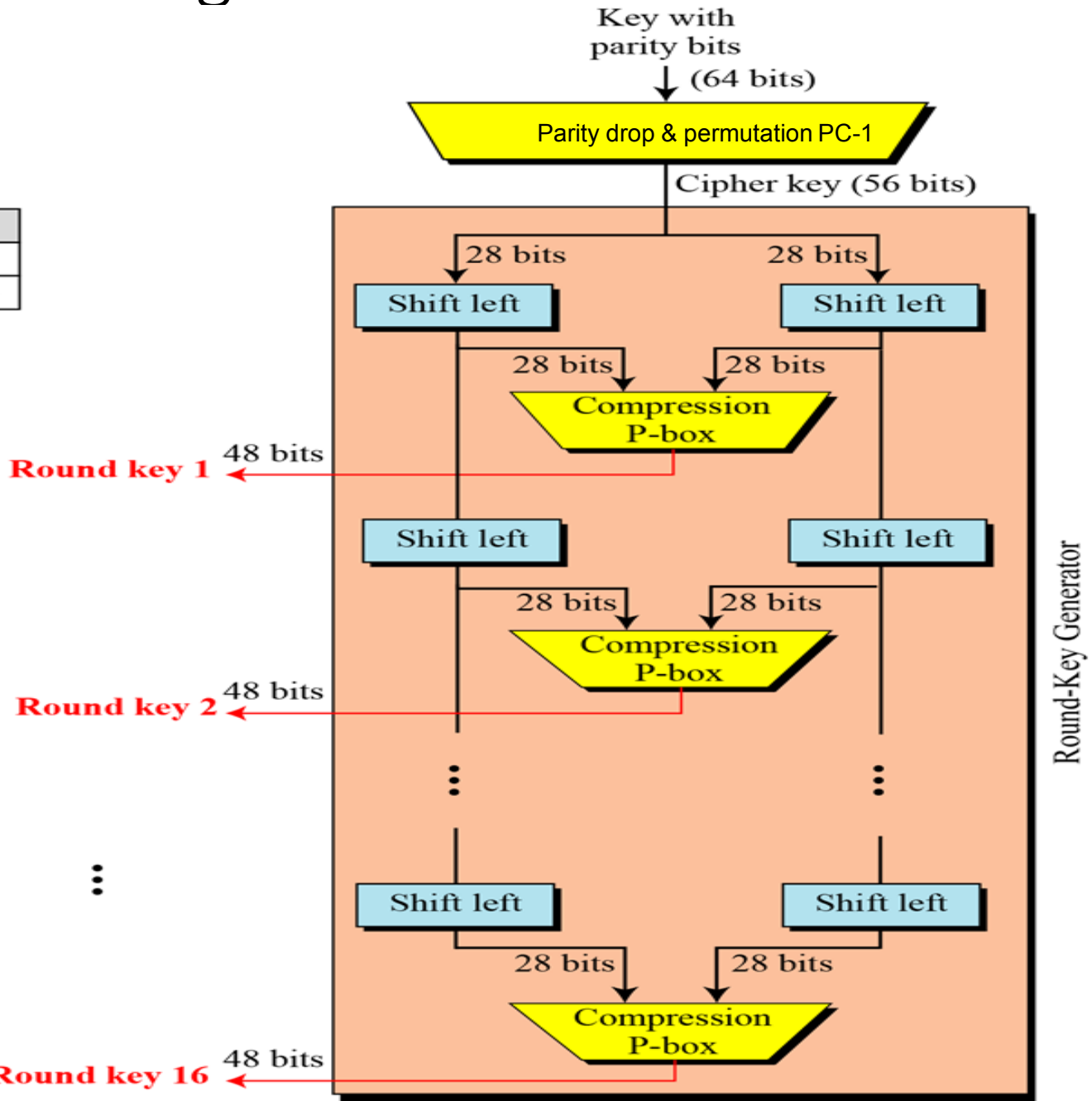
Key Generation Algorithm

- **Creating 16 subkeys, each of which is 48-bits long:**
  - a) **Both halves of 64-bit key**(only 56 bit appears since 8[every 8th bit at positions 8, 16, 24, 32, 40, 48, 56 and 64 of the key is discarded] of the 64 bits of the key are used as **parity-check bit[** is the XOR of the previous 7 bits]) are **permuted** according to the **table PC-1** .
  - b) Then **56 bits** are **split to left and right sub-part** of **28 bits** each
  - c) **Left and right halves(28-bit)** are then **rotated left** by **one or two bit**s specifically for each round.
  - d) **From 56 bit key 4,8 sub-key bits(8,17,21,24--> LK 6,9,14,25-->RK)** are selected by **Compression Permutation PC-2.** These bits form sub-keys for each round
- It is repeated for **16 rounds**

Key Schedule Algorithm

Shifting

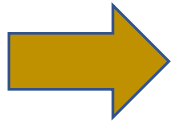
Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits



Generation of 56 bit key

- Drop parity bit as per *Parity-bit drop table*:

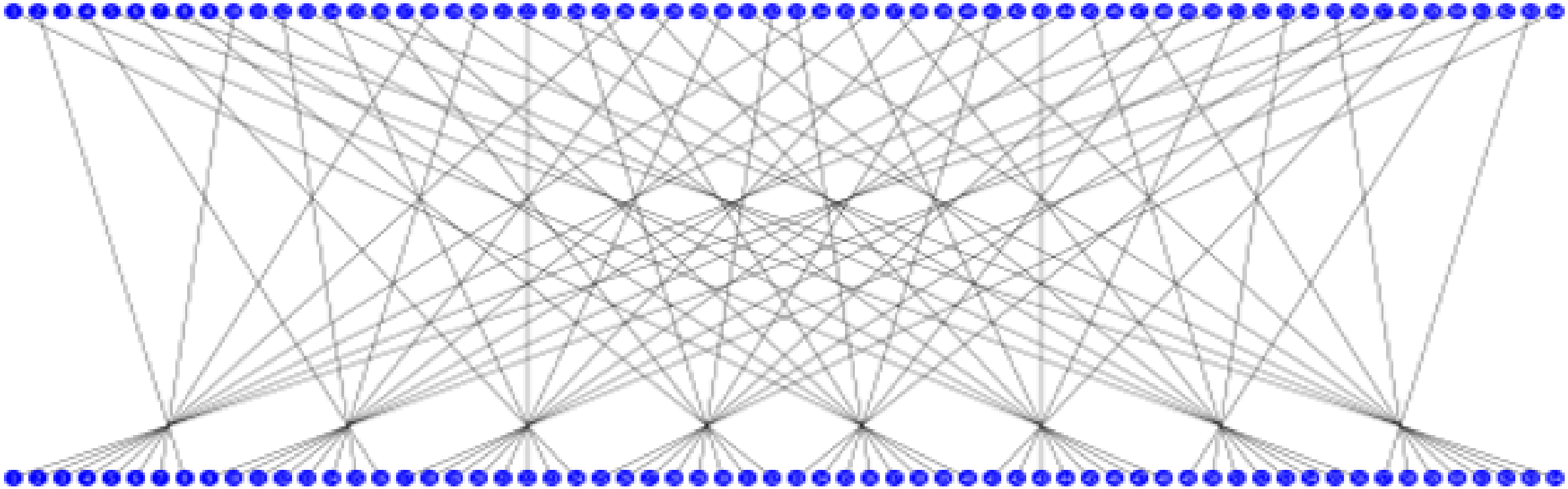
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64



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

a) Initial & Final Permutations

- initial and final permutations is applied on each block of data and are straight Permutation boxes (P-boxes) that are inverses of each other. It rearranges bits in a certain, predefined way. it is a vector, not a matrix.
- with out positions 8,16,24,32,40,48,56,64 marked with "F“. discards the parity bits and transposes the remaining 56 bits as below: {1,2,3.....,64}-{8,16,24,32,40,48,56,64}
- The meaning is as follows: the first bit of the output is taken from the 57th bit of the input; the second bit from the 49th bit, and so on, with the last bit of the output taken from the 4th bit of the input.



INITIAL PERMUTATION TABLE AFTER PARITY BIT DROP

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

b) Key Permutation

- 56 bits are selected from 64-bit key which is split to left and right sub-part of 28 bits each are left shifted. In each round of Feistel functions, the 28-bit halves of key are rotated left by one or two bits.
- 56 bit DES key, numbered 0,1,2,…,55

Left half						
49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31
Right half						
55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3

c) Left Shift Operation

- In each round of Feistel functions, the 28-bit halves of key are rotated left by one or two bits.
- It is a circular shift of some positions. The number of shifted positions is given below

No of cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Amount of bits	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

d) Permutation PC-2

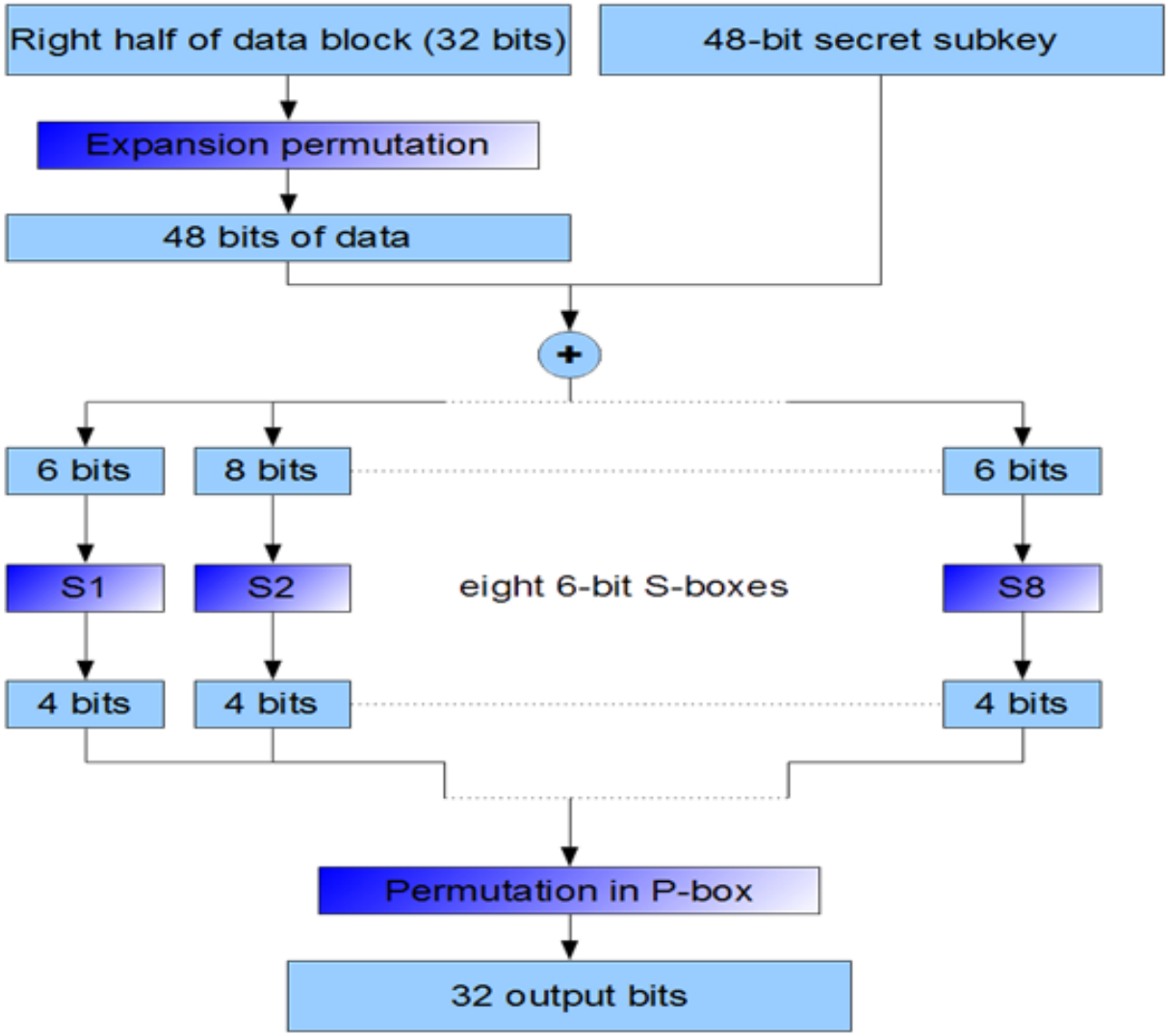
- Bits 8,17,21,24 of LK omitted each round
- Bits 6,9,14,25 of RK omitted each round
- During Key Permutation PC-2 48 bits are selected from the 56-bit subkey obtained for a given round of Feistel functions. This is known as **Compression permutation**

Left half of subkey K_i is of LK bits											
13	16	10	23	0	4	2	27	14	5	20	9
22	18	11	3	25	7	15	6	26	19	12	1
Right half of subkey Ki is RK bits											
12	23	2	8	18	26	1	11	22	16	4	19
15	20	10	27	5	24	17	13	21	7	0	3

Compression permutation table

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Feistel Function of DES



Encryption Modes

- **Encryption modes:** define how messages larger than the block size are encrypted, very important for the security of the encrypted message.
- **DES Modes of Operation**
- The DES algorithm turns a 64-bit message block **M** into a 64-bit cipher block **C**. If each 64-bit block is encrypted individually, then the mode of encryption is called ***Electronic Code Book*** (ECB) mode.
- There are two other modes of DES encryption, namely ***Chain Block Coding*** (CBC) and ***Cipher Feedback*** (CFB), which make each cipher block dependent on all the previous messages blocks through an initial XOR operation.

DES Strengths

- 2 desired properties of a block cipher are:
  - **Avalanche Effect:** A small change in P.T/Key should create a significant change in C.T. DES is strong with this regard(almost 45% difference if 1 bit is changed)
  - **Completeness Effect:** Each bit of C.T needs to depend on many bits of P.T. Diffusion and confusion produced by D-box and S-box shows strong completeness effect. S-box is non-linear
- 16 rounds of DES is another strength.
- Different S-boxes used for each round also increases security.

DES Weakness

- **Weakness in Cipher Design**
- **S Box**
  - 2 specifically chosen i/p to S-box array can create same o/p
  - Same o/p in single round can be obtained by interchanging 3 neighbouring S-boxes
  - **D Box**
  - Designer's haven't explained the need of initial and final permutation
  - In expansion permutation 1^{st} and 4^{th} bit series are repeated
- **Weakness in cipher Key**
- **Key Size**
  - To brute force attacker has to check 2^{56} keys. For a computer with 1 processor takes more than 2000 years assuming 1 million keys can be checked in one second. But current super computer has taken 112 days or with a computer n/w(3500 computers) it can be done in 120 days.
- **Weak Keys**
  - 4 out of 2^{56} keys are weak keys(After parity drop operation keys which consists of all 0's or all 1's or half 0's and half 1's are known as weak keys). Round keys generated from weak keys either will be same or may have same pattern. This will cause a block to be repeated in subsequent encryption.

Types of Weak Keys

- **Weak Keys:** **4** out of 2^{56} keys are **weak keys**(After parity drop operation keys which consists of all 0's or all 1's or half 0's and half 1's are known as weak keys). Round keys created from weak keys have same pattern as cipher key. Double encryption with weak keys reveal plain text. Each weak key is inverse of itself.
- **Semi weak keys:** **6 key pairs** out of 2^{56} keys are **semi-weak keys.** A semi-weak key creates only **2 different round-keys** and each of them is **repeated 8 times.**
- **Possible weak keys:** **48** out of 2^{56} keys are **possible weak keys**. A possible-weak key is a key that creates that creates only **4 different round-keys** and will be **repeated in 16 rounds**
- **Key Complement:** Out of 2^{56} keys **half of the keys are compliment of other**. Attacker can break CT if key is complement of other since CT will also be complement of original CT.
- **Key Clustering:** A situation in which **2 or more keys can create same CT from same PT**. Each pair of **semi-weak key is a cluster**.

WEAK KEYS Example

0000000	0000000
0000000	1111111
1111111	0000000
1111111	1111111

WEAK KEYS

First key in the pair	Second key in the pair
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E0 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEF0 FEF1 FEF1

SEMI WEAK KEYS

Security of DES

- Security depends heavily on S-boxes.
  - Everything else in DES is linear
  - S-box constants weren't explained what is the purpose
- 35+ years of intense analysis has revealed no back door
- **Possible Attacks:**
- **Brute-Force Attack:** Attacker can break DES in 2^{55} encryptions. Today 3-Des with 2 keys or 3 keys are used as they are resistant to brute force attacks.
- **Differential Cryptanalysis:** Eve is having access Alice's computer and selects chosen plain text and find's correspond Ct in order to retrieve key. But DES is resistant to this attack because of 16 different S-boxes used.
- **Linear Cryptanalysis:** DES is not resistant to linear cryptanalysis since S-box is linear and it can be solved by using linear eqn approximation. It can be broken using 2^{43} pairs of known plain text.

3DES

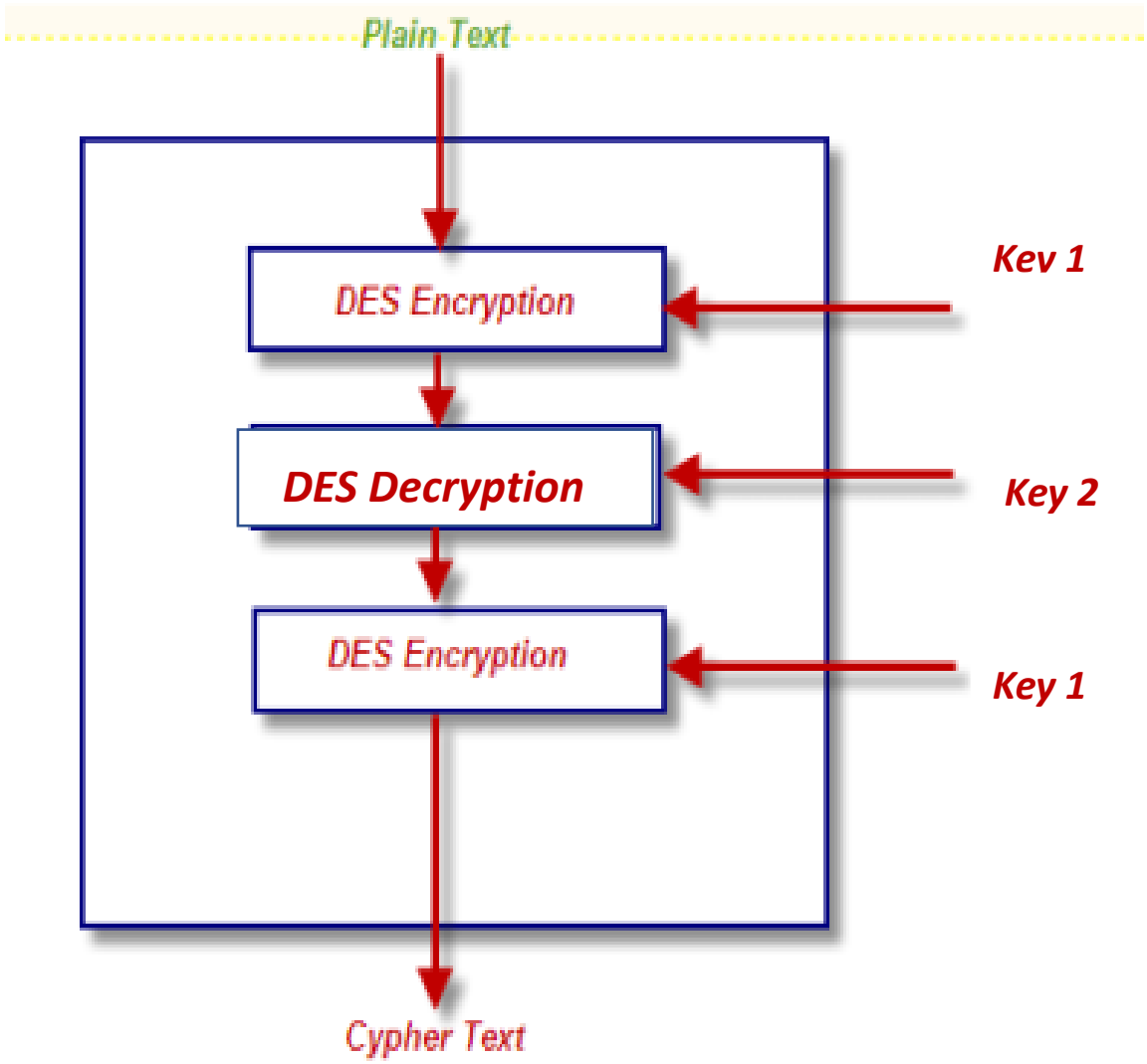
TRIPLE DES

TRIPLE DES

- The enhancement in DES is **Double DES** that uses **two keys for 2 rounds of DES.** It is inadequate because of the **meet-in-the-middle attack.**
- **Triple Data Encryption Standard (3DES)**, also known as *Triple Data Encryption Algorithm* (TDEA or Triple DEA) runs the **DES algorithm three times** to each data block of data, with **three 56-bit keys**.
  - **Key one** is used to **encrypt** the plaintext.
  - **Key two** is used to **decrypt** the text that had been encrypted by key one. Decrypting with a separate key only serves to jumble up the data even further.
  - **Key three** is used to **encrypt** the text that was decrypted by key three.
- Triple DES uses 48 passes through the algorithm which results three 64-bit keys, in total an overall key length of 192 bits.
- Slower than regular DES but improve security in billion times.
- 3DES is Likely to Be disallowed after 2023

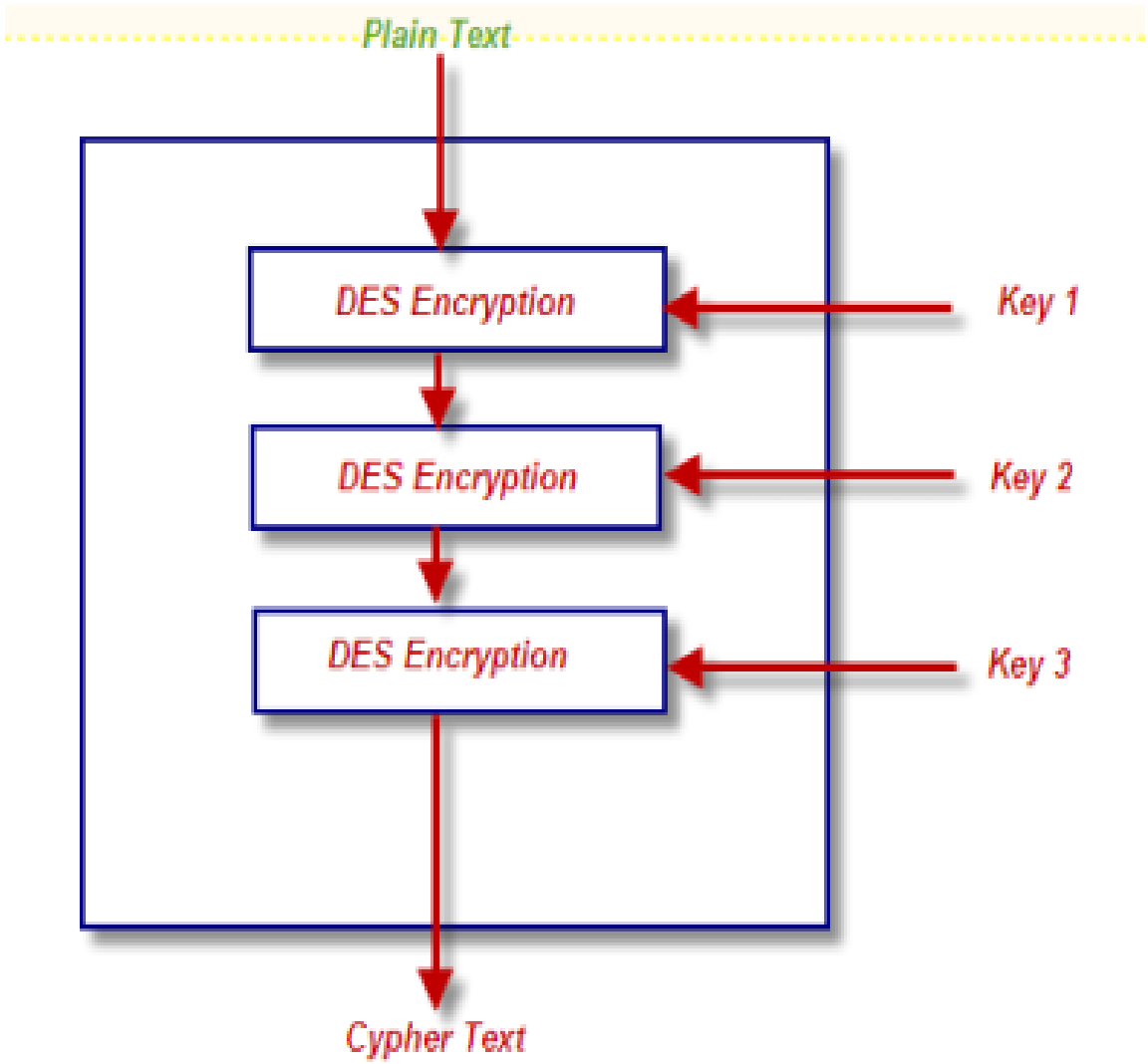
TRIPLE DES WITH 2 KEYS

- Triple DES is a variation of DES that uses **3 stages** of **DES encryption and decryption**.
- The **1 st , 3 rd stage** use $K1$ **key** and **2 nd stage** use $K2$ ***key***.
- To make triple DES compatible with single DES, the **middle stage uses decryption** in the encryption side.
- The function follows an encrypt-decrypt-encrypt (EDE) sequence.
- **C = E($K1$, D($K2$, E($K1$, P)))**
- **P = D($K1$, E($K2$, D($K1$, C)))**



TRIPLE DES WITH 3 KEYS

- Triple DES is a variation of DES that uses **3 stages** of **DES encryption and decryption**.
- The **1 st stage** use ***K*1 key , 2 nd stage** use ***K2 key*** and **3 rd stage** use ***K3 key*** .
- 3-key 3DES requires a key of length 56*3= 168 bits and is defined as,
- Encrypt: **C = EK3 [DK2 [EK1 [P]]]**
- Decrypt: **P = DK1 [EK2 [DK3 [C]]]**



AES

ADVANCED ENCRYPTION STANDARD

AES

- **AES** is a modern block symmetric cipher, developed in 1997 by **Vincent Rijmen** and **Joan Daemen** in Belgium. It is known as **Rijndael cipher.**
- **AES** uses the **Rijndael algorithm** in combination with symmetrical block ciphers as its encryption method.
- Iterated block cipher (like DES) but not a Feistel cipher (unlike DES)
- It works in parallel over the whole input block (processes data as block of 4 columns of 4 bytes). So **encryption is fast.**
- Used in numerous protocols and transmission technologies, for example the WPA2 protection of WiFi networks, SSH or IPsec Standard, Voice-over-IP technology (VoIP)

AES Rounds

- AES is a non-Feistel cipher that encrypts and decrypts a **data block** of **128 bits**.
- Rounds of AES can vary s per **key-size** as follows

Key Size	Rounds	Key Scheduling
128	10	44
192	12	52
256	14	60

- Round Keys are always 128 bits

Data Units of AES

- **Bit:** A binary digit with a value 0 or 1
- **Byte:** A byte is a group of 8 bits
- **Word:** A word is a group of 32 bits.
- **Block:** AES encrypts and decrypts data blocks as a row matrix of 16 bytes

AES Algorithm

AES will be having a one time initialization procedure; For remaining rounds except last round the steps remain same. Final round omits multiplication of columns(only 3 stages)

One time Initialization Process

1.Plain text 16-byte(**128-bit)** (data block of 4 columns of 4 bytes is state) **is** divided into 4bytes × 4 bytes **blocks** data and are copied into a 2d array(4 bytes × 4 bytes) in order, called as **state.** This is known as one time initiali of sate.

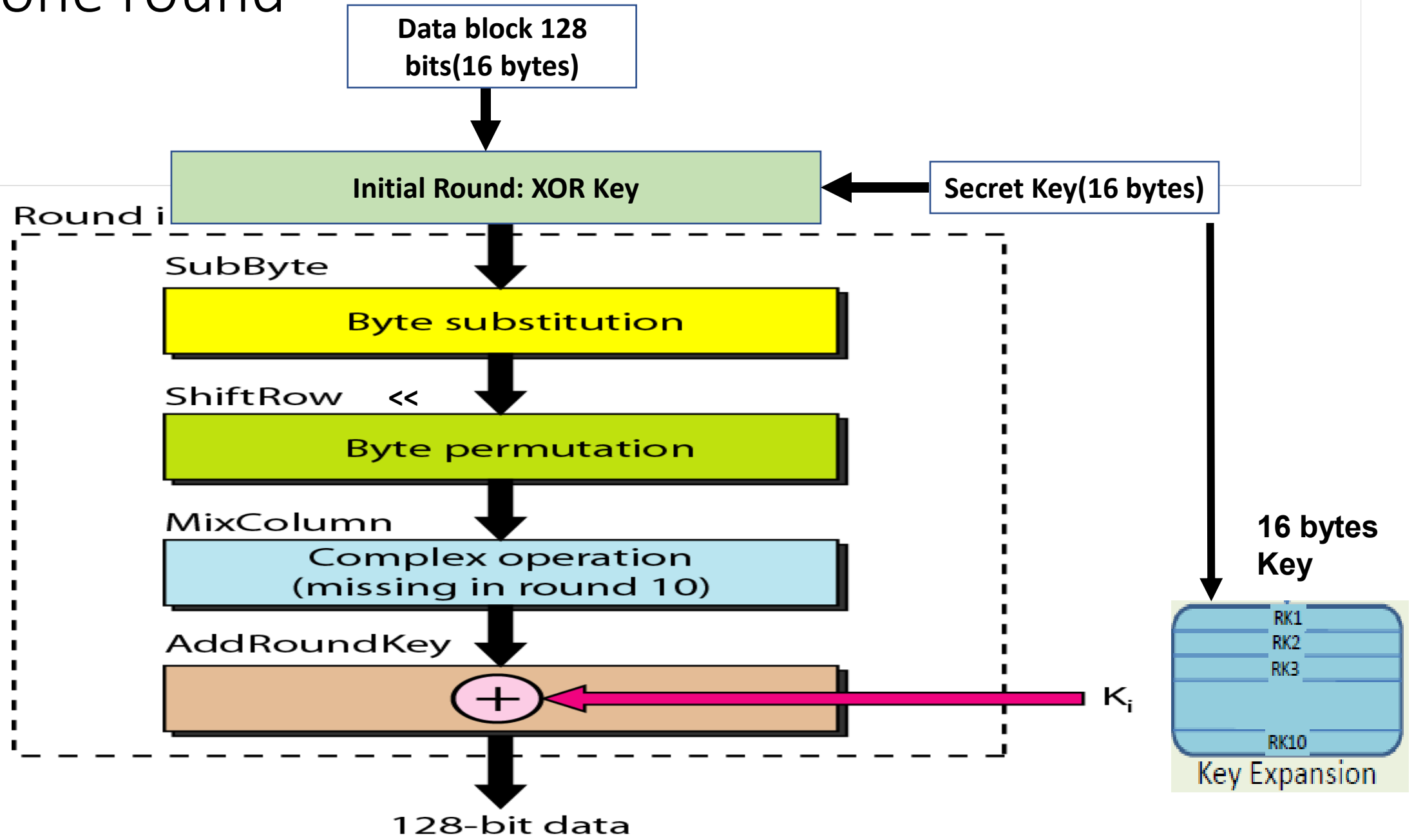
2.Every bytes of data block(state array) are XORed with corresponding bytes of the expanded sub-key.

AES ROUND STRUCTURE

For each round do the following. Each round uses 4 functions (3 “layers”).

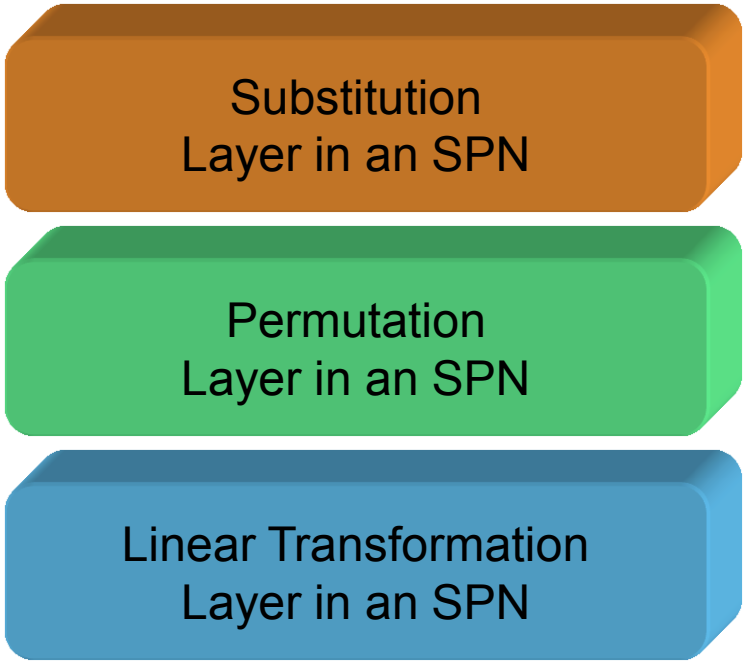
1. **ByteSub** (nonlinear layer: byte-by-byte substitution of state array using Rijndael’s S-box. S-box only 1 s-box present) is invertible and constructed using Galois field.
2. **ShiftRow** (linear mixing layer: Its a permutation, which cyclically shifts the last three rows in the State Array in such a way 2nd row by 1 bit position 3rd row by 2-bit positions and 3rd row by 3 bit positions to left. Last row remains same. In decryption shifting is done in same manner to right.
3. **MixColumn** (nonlinear layer: Sate matrix after step 2 is multiplied using a constant matrix of size 4X4. In matrix multiplication operation addition operation is replaced by XOR and multiplication is donee in Galois field. Multiplication of bits is done by referring L-table and E-table such that a⊗b= E(L(a)+L(b))
4. **AddRoundKey** (key addition layer: bit-by- bit XOR with an expanded key). XOR sate matrix after step 3 with sub-key block generated using key generation algorithm.

AES one round

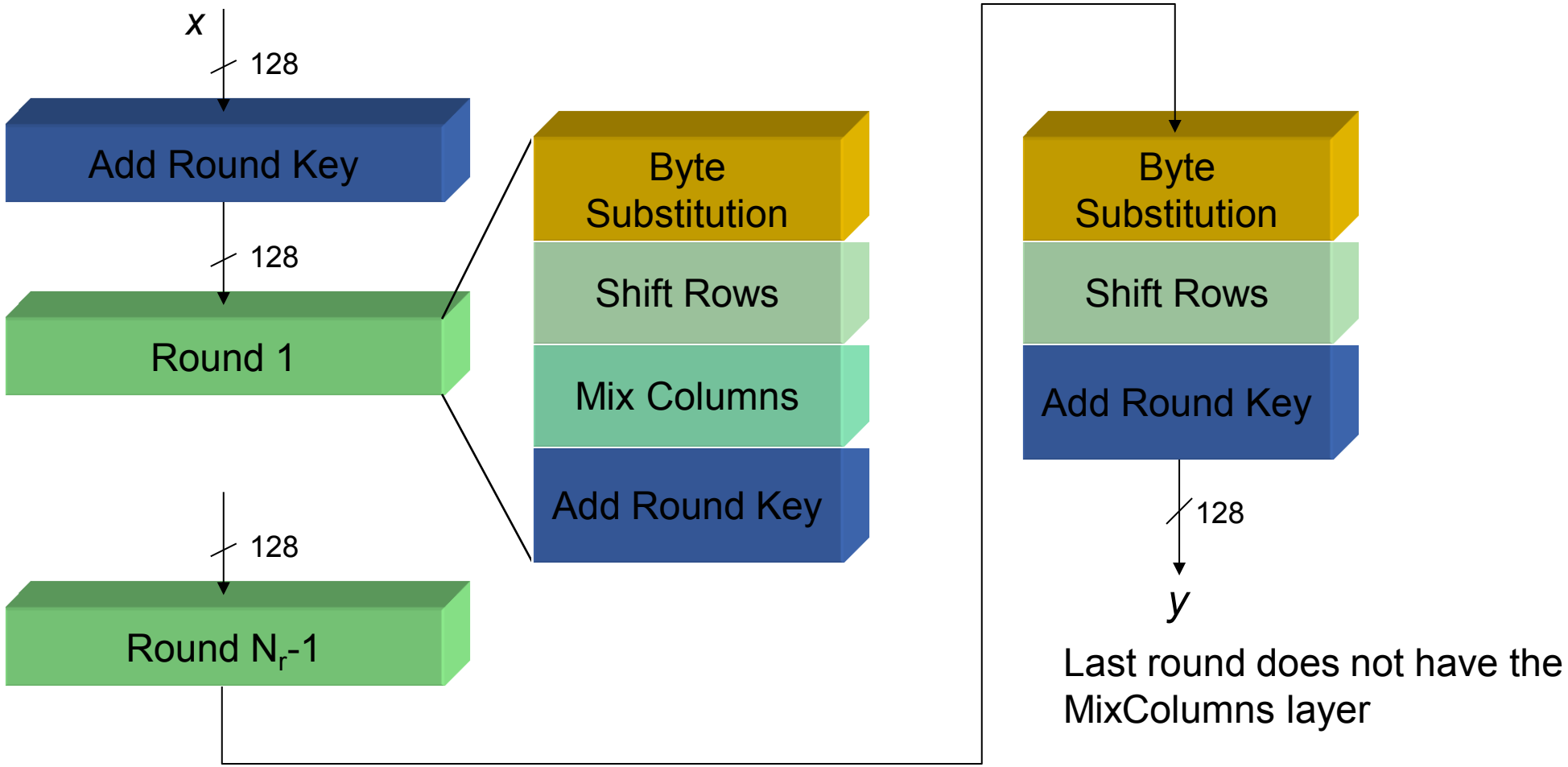


Layers for encryption in AES

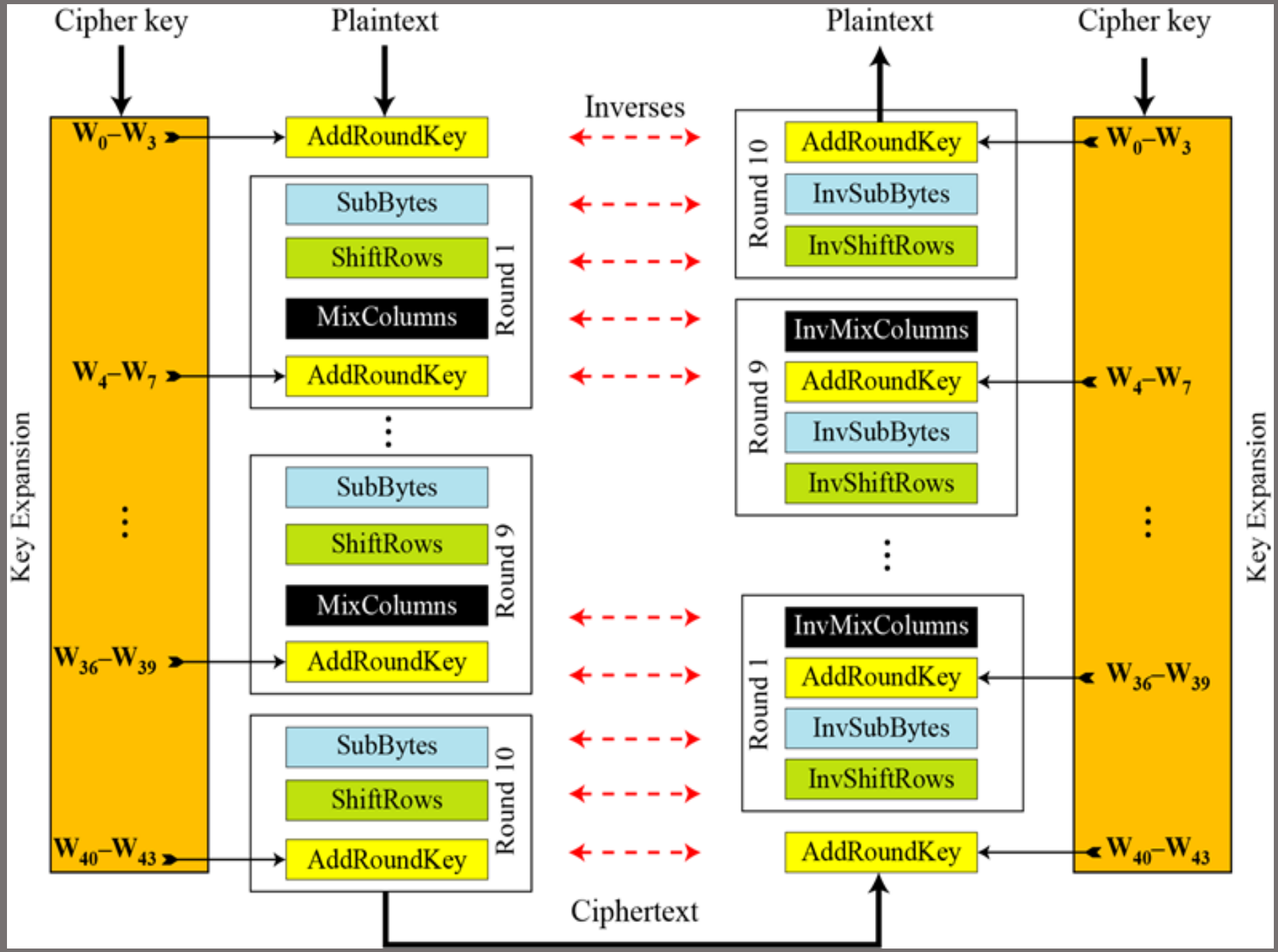
- **Confusion Layer**
  - A byte substitution layer creates the confusion block
  - It is non-linear to prevent easy analysis
- **Diffusion Layer**
  - There are two layers that introduce diffusion
    - Shift rows (ShiftRow)
    - Multiply columns (MixColumn)
- **Key Addition Layer**



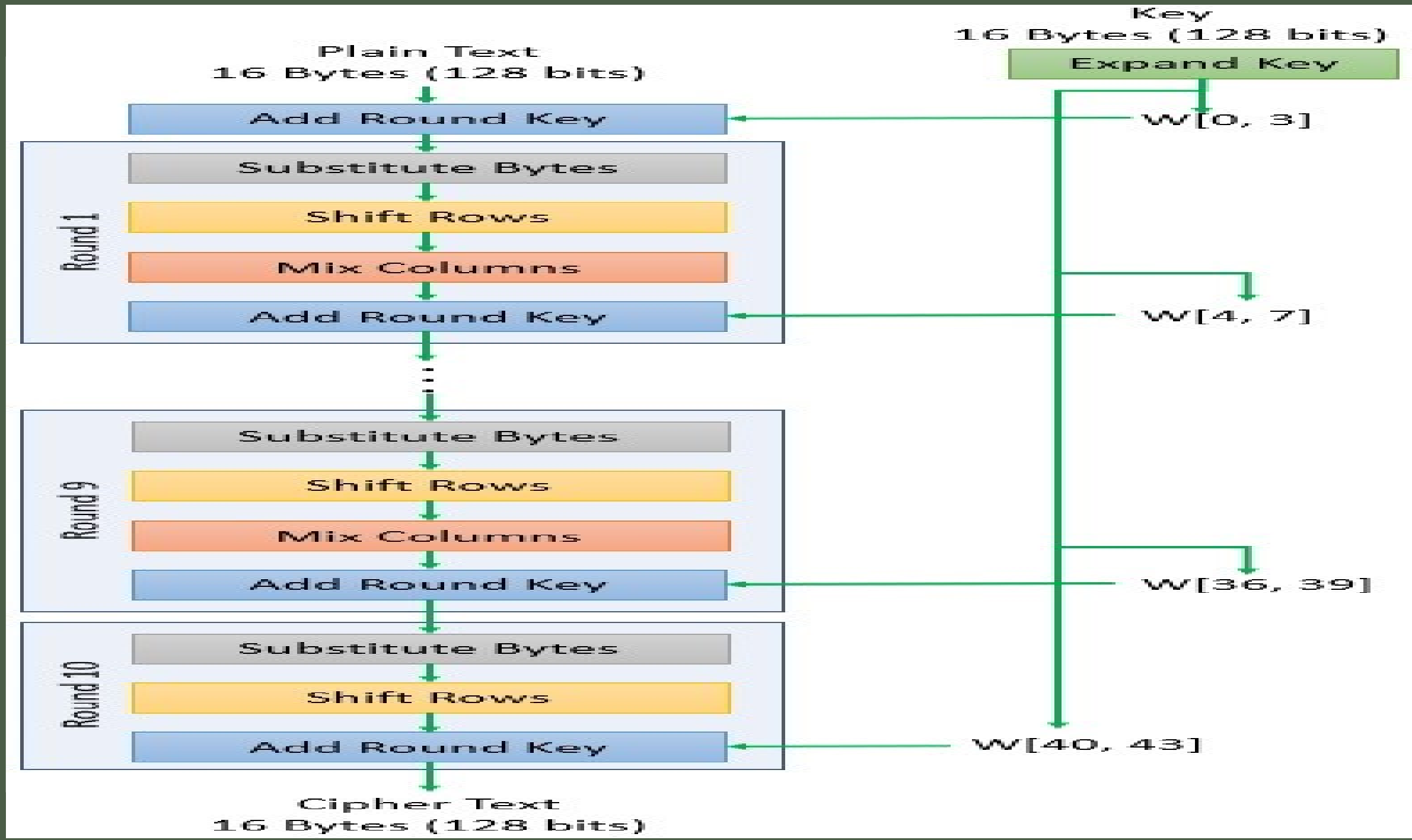
BLOCK DIAGRAM OF AES ENCRYPTION

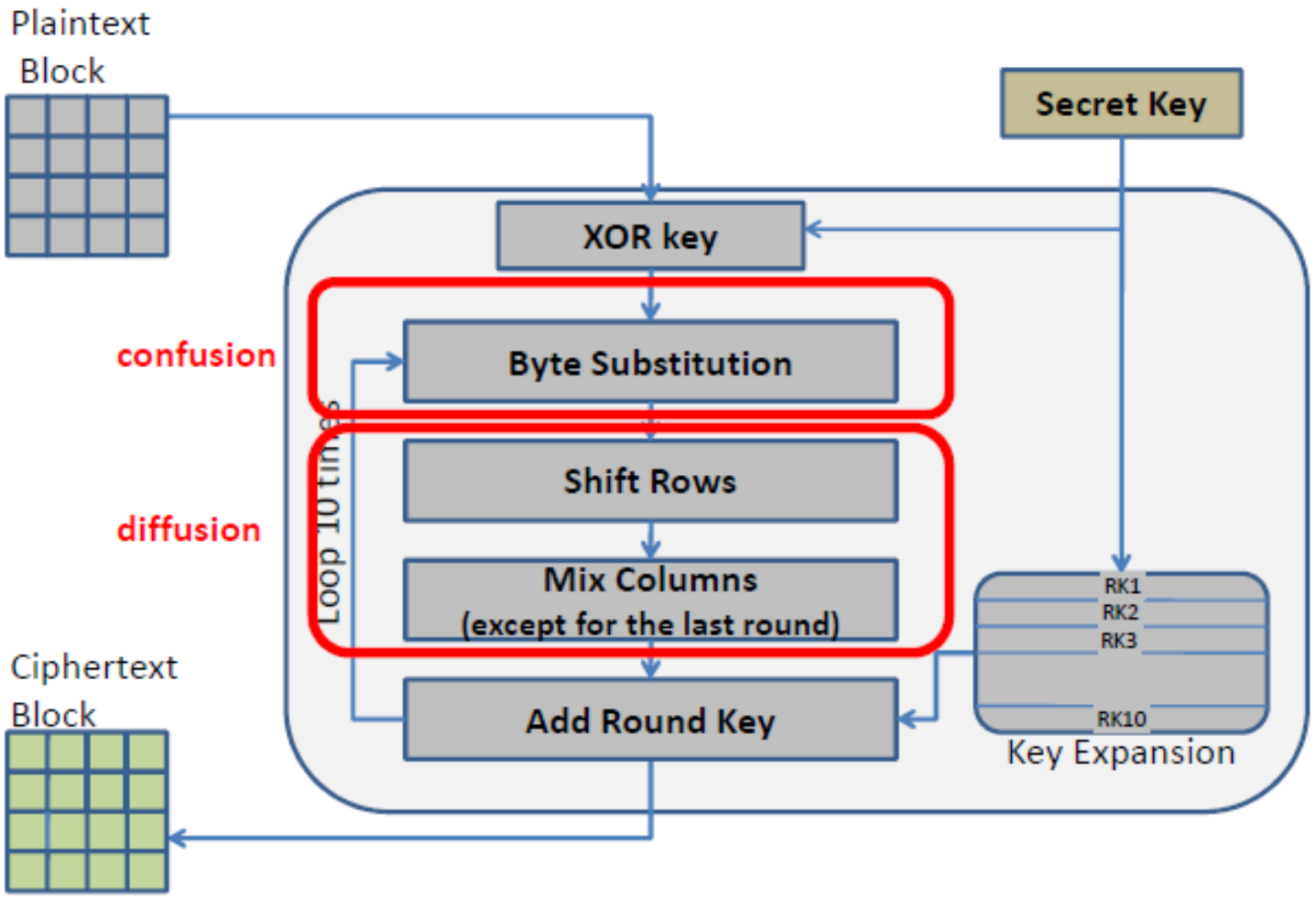


Last round does not have the MixColumns layer



AES STRUCTURE

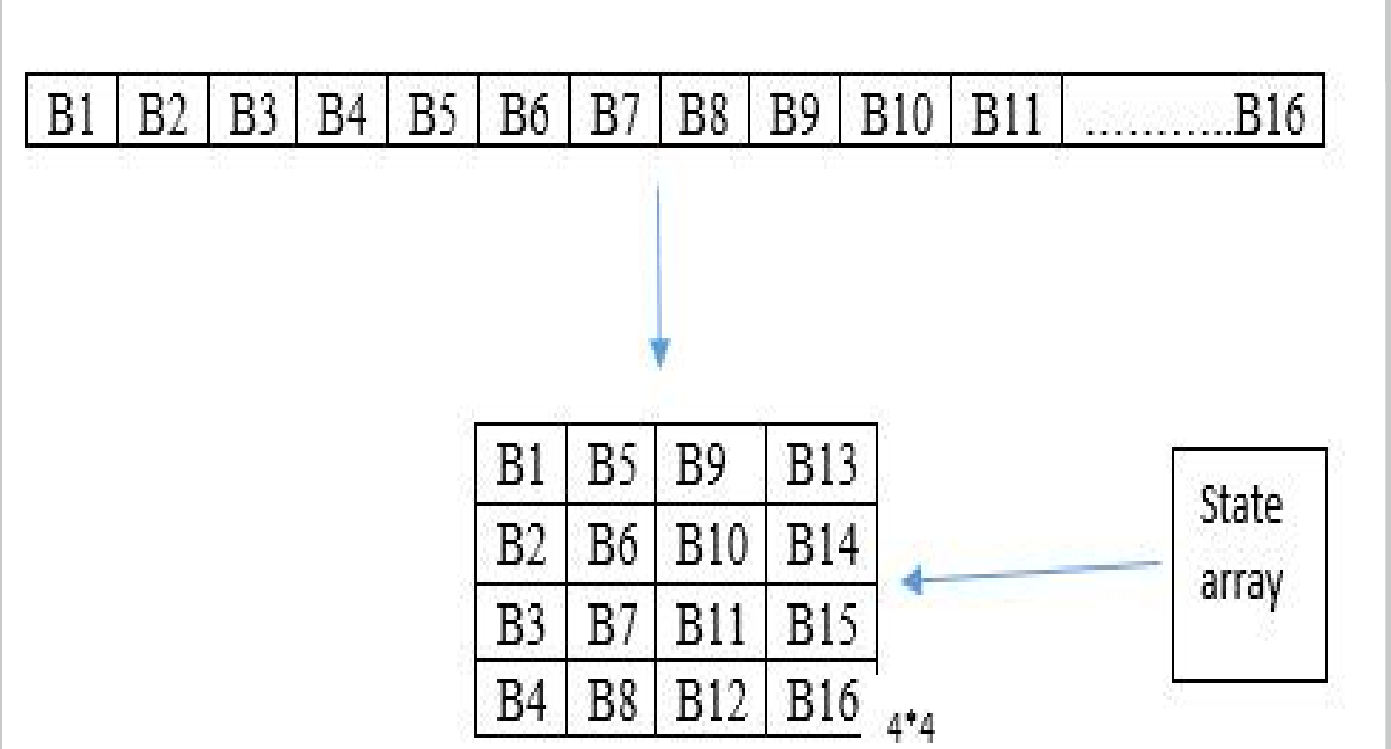

Key
16 Bytes (128 bits)
Expand Key
Plain Text
16 Bytes (128 bits)
Add Round Key
W[0, 3]
Round 1
Substitute Bytes
Shift Rows
Mix Columns
Add Round Key
W[4, 7]
Round 9
Substitute Bytes
Shift Rows
Mix Columns
Add Round Key
W[36, 39]
Round 10
Substitute Bytes
Shift Rows
Add Round Key
W[40, 43]
Cipher Text
16 Bytes (128 bits)



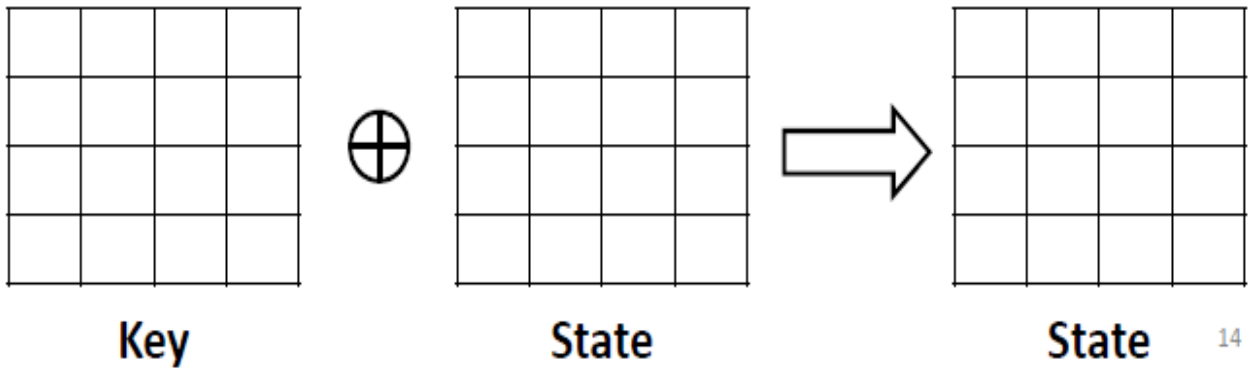
CONFUSION AND DIFFUSION IN AES

0. One time initialization of State

- In **one time initialization** of **16 byte plain text block** is copied into a **two dimensional array of size 4X4** called **state**
- Order of copying is **column order**
- **Every byte** of state array is **XORed** with first **16bytes of expanded Key.**
- One time initialization is complete.



- **AddRoundKey(State, Key):**



EXERCISE

Write state matrix for the message:"AES uses a matrix"

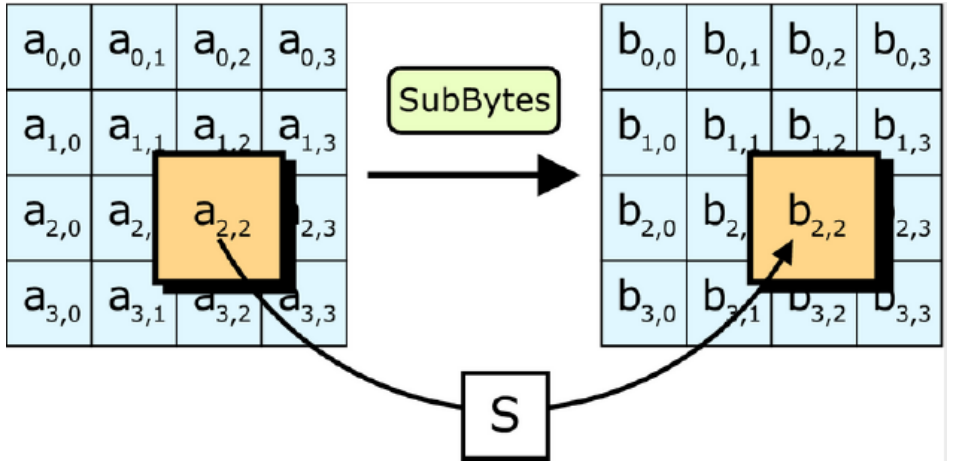
SOLUTION

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

1. ByteSub

- **SB (Substitute Bytes) Operation**: **Each byte** of the **state matrix** is **replaced with another byte**, based on a lookup table, called **Rijndael's S-Box**.
- The lookup table assures its **non-linear**(but invertible) composition of two math operations.
- S-box constructed using defined transformation of values in GF(2^8)
- S-box is represented as a 16x16 array, rows and columns indexed by hexadecimal bits.
- Designed to be resistant to all known attacks.
- Only 1 S-box is present.



Rijndael S-box Table

- 16X16 matrix that contains a permutation of all possible 256 8-bit values.
- Each byte of state array is mapped to new byte as follows: **leftmost 4 bits** as **row** and **rightmost 4 bits** as **column** value
- The first 4 bits in the byte(the first hexadecimal value, hence) indicate the row index, the last 4 bits individuate the column index.
- **Example:** hexa **53** is replaced with hexa **ED**
- Find values for substitute byte operation: **EA 04 65 85**

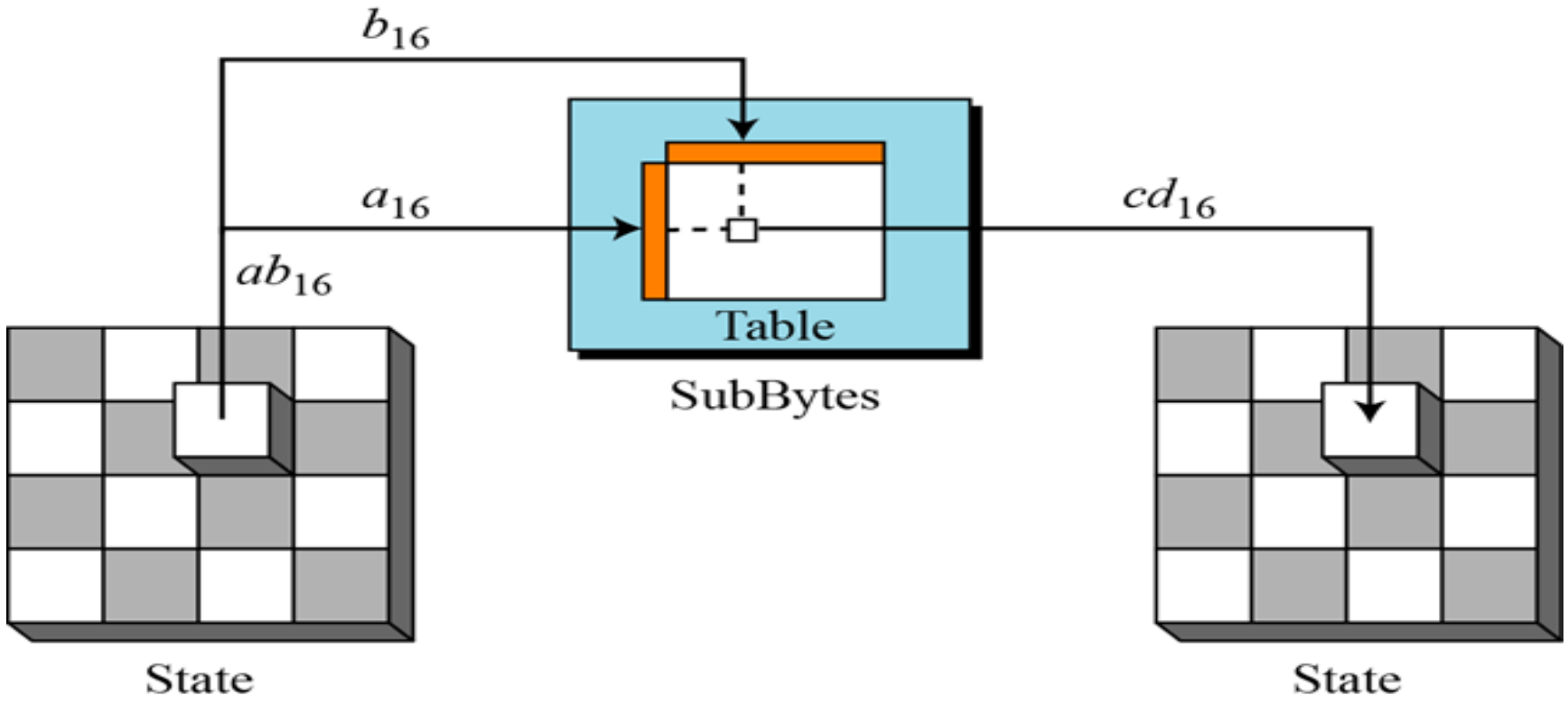
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	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	3	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

Substitute Bytes

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

→

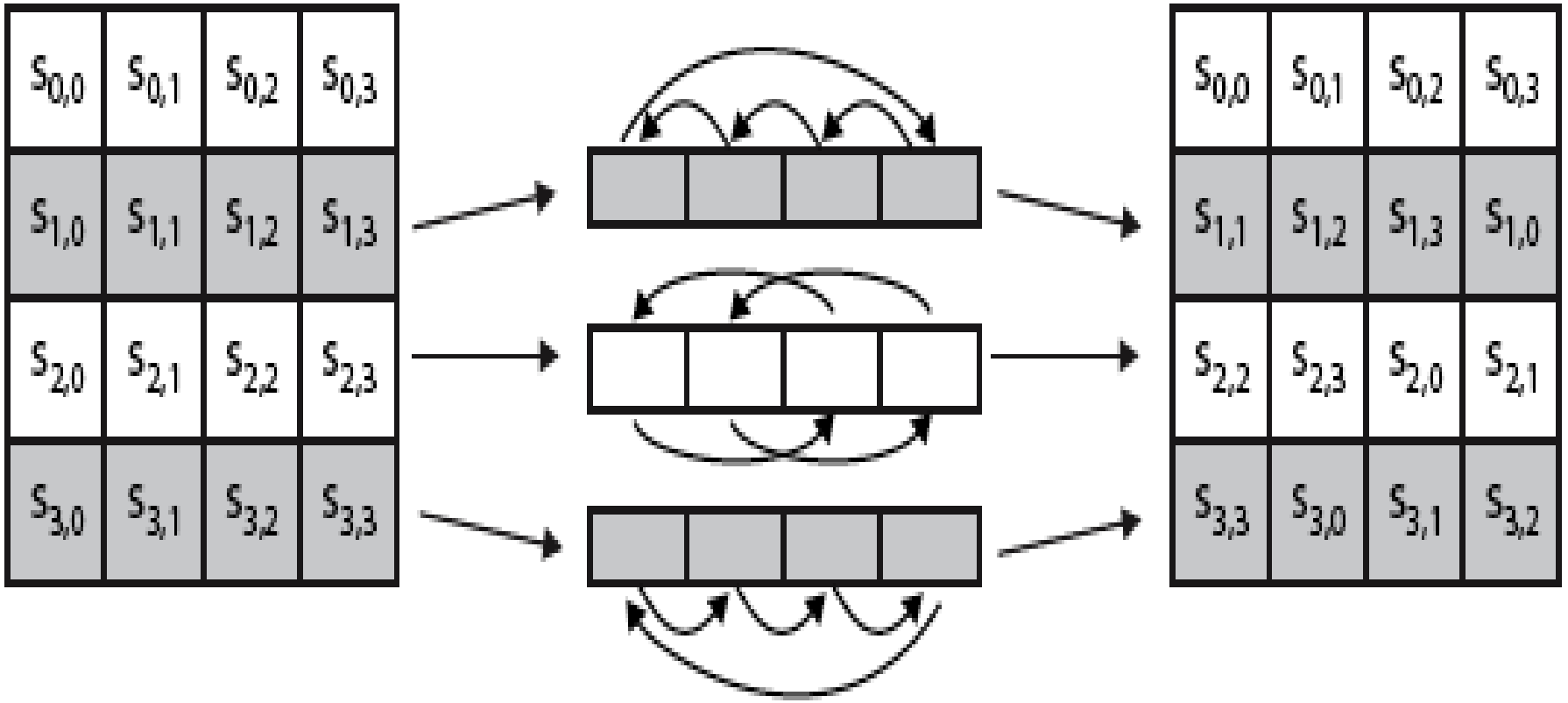
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6



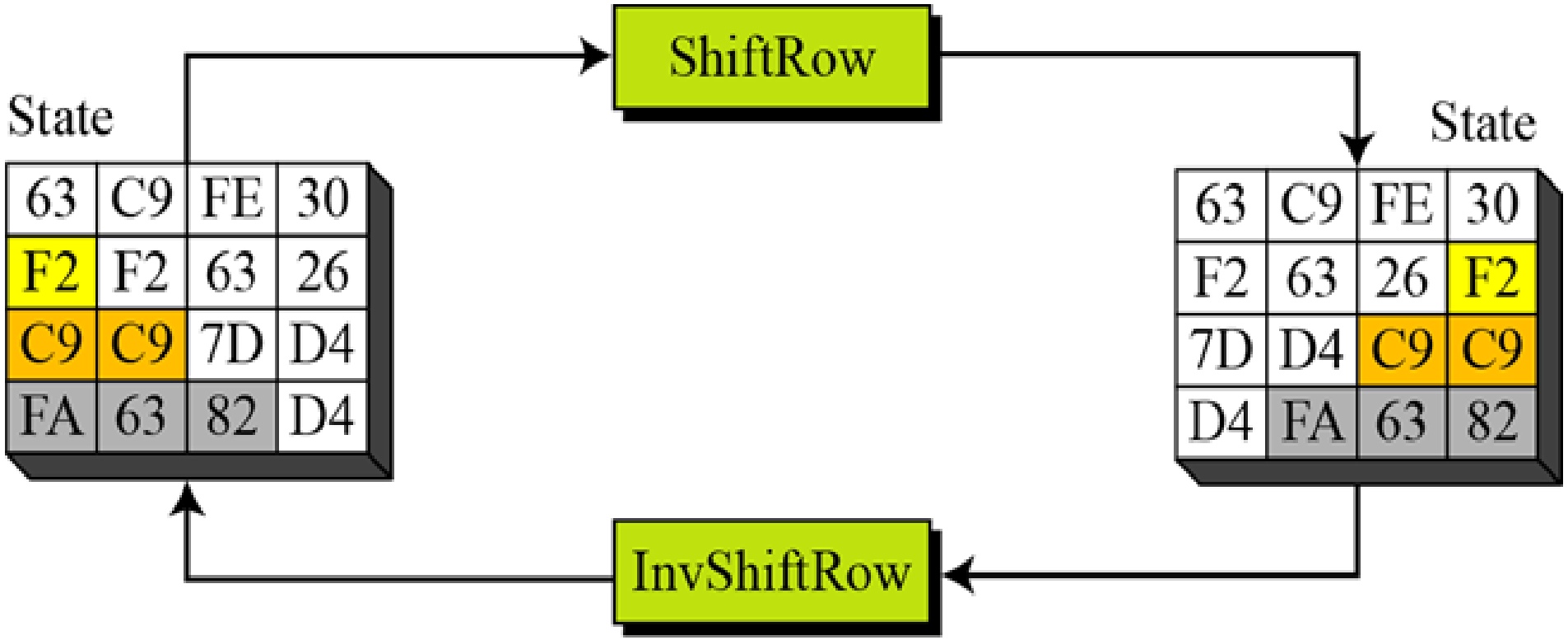
2. Shift Row

- Bytes stored in state matrix are left circular shifted as follows
  - **1st row** is **unchanged**
  - **2nd row** does **1 byte** circular shift **to left**
  - **3rd row** does **2 byte** circular shift **to left**
  - **4th row** does **3 byte** circular shift **to left**
- This state is called SR (Shift Rows) Operation.
- The leftmost bytes in each row moves to the right side of the same row.
- Decryption inverts using shifts to right.
- Since state is processed by columns, this step permutes bytes between the columns.

Shift Rows


$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

Shift Rows


ShiftRow
State
63 C9 FE 30
F2 F2 63 26
C9 C9 7D D4
FA 63 82 D4
State
63 C9 FE 30
F2 63 26 F2
7D D4 C9 C9
D4 FA 63 82
InvShiftRow

Shift Rows

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

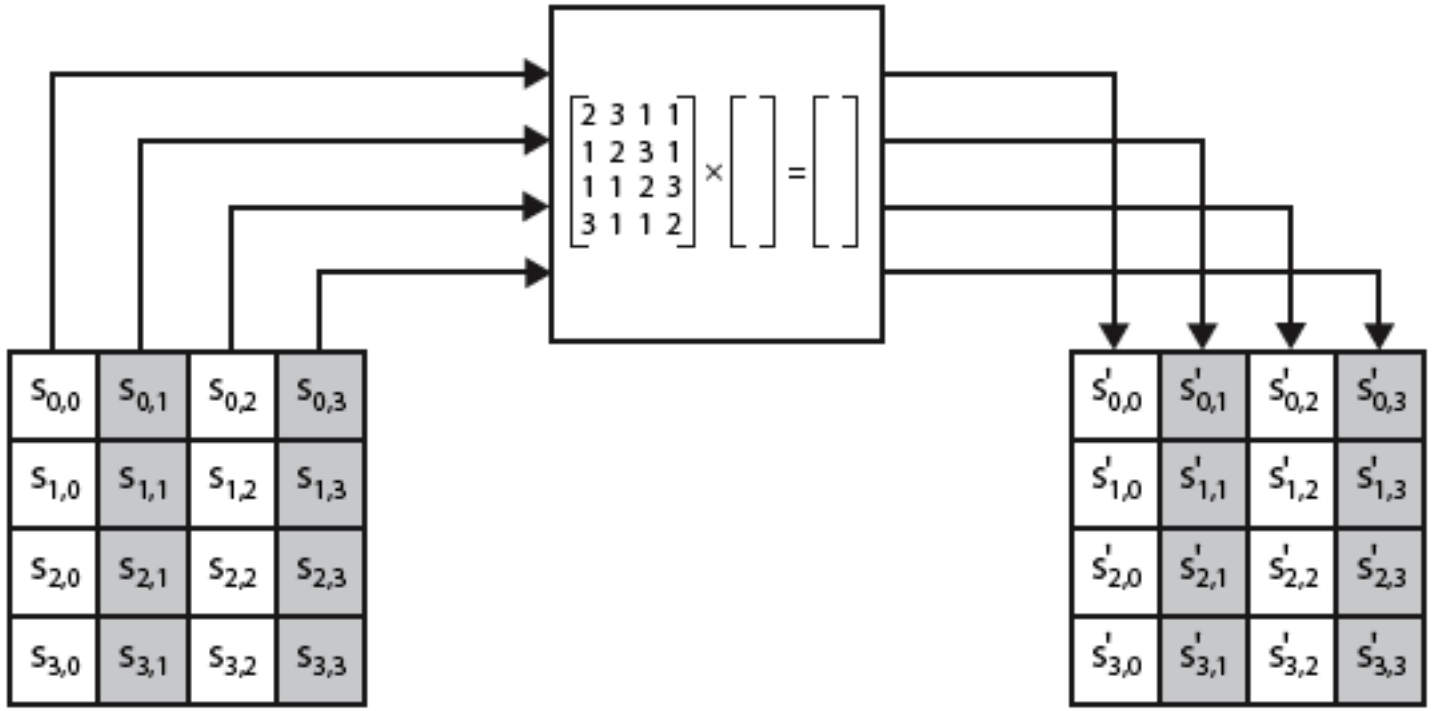
→

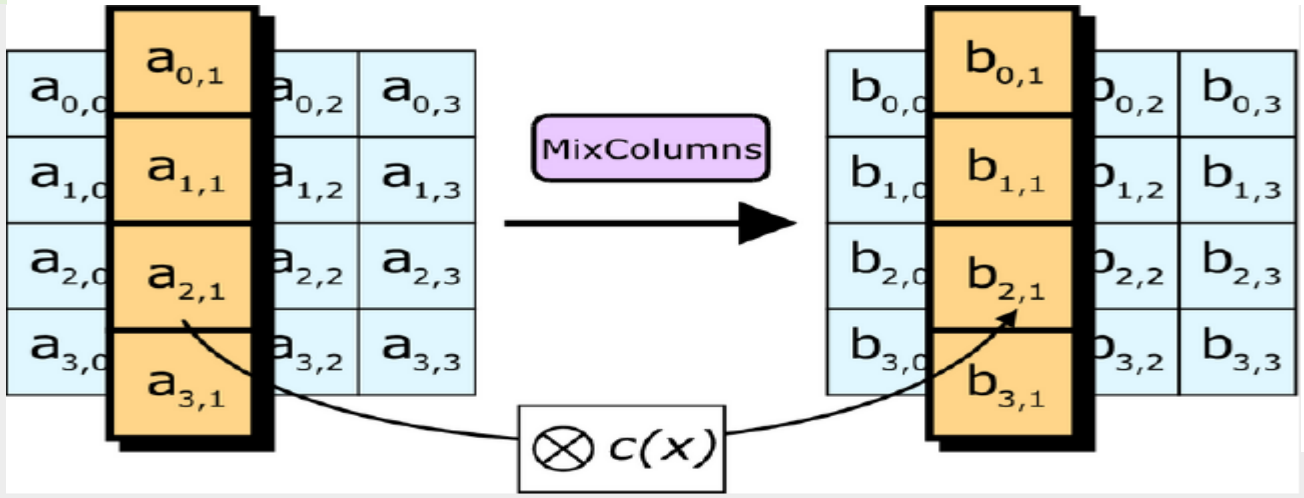
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

3. MixColumn

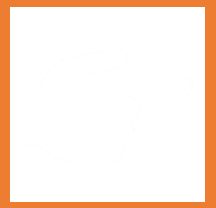
- Mix Column operation process each column individually. **State matrix** is **multiplied** by a **predefined constant matrix** which is known as **circulant MDS matrix** of size of 4bytes x 4bytes. The result of each multiplication is a new column which contains different 4 bytes. Its an **Invertible linear operation**.
- In matrix multiplication each column of state matrix is multiplied by a row in constant matrix. Addition operation is replaced by XOR operation.
- As a result of Multiplication of the square matrix each byte is replaced by a value dependent on all 4 bytes in the column
- matrix multiplication mentioned above is performed in GF(2^8) using prime poly $m(x) = x^8+x^4+x^3+x+1$.So values will be in the range 0-255
- Multiplication output is the result from L-table and addition output is from E-table

Mix Columns

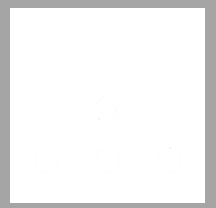

2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$


$a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,1}$ $b_{3,2}$ $b_{3,3}$
⊗ c(x)

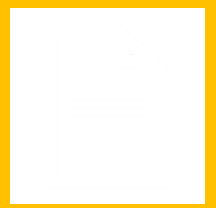
Why MDS(Maximum Distance Separable codes) matrix?



If the input of a column changes then all outputs change



Values [2,3,1,1], are the smallest which result in MDS matrix that is also circulant



Has an inverse in the AES field

AES OPERATIONS

- All AES operations are performed in the field GF(2^8).
- The field's irreducible polynomial is: $x^8 + x^4 + x^3 + x + 1$
- in binary notation $(1\ 0001\ 1011)_2$ In hex notation $(11B)_{16}$

Mix Coulmn Example

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

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

$$\{87\} \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\} \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{03\} \bullet \{A6\}) = \{94\}$$

$$(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) = \{ED\}$$

E Table

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

L Table

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

Galois field Multiplication using E-table & L-table

- One multiplication in Mix column is defined as follows: **a*b= E(L(a)+L(b))**

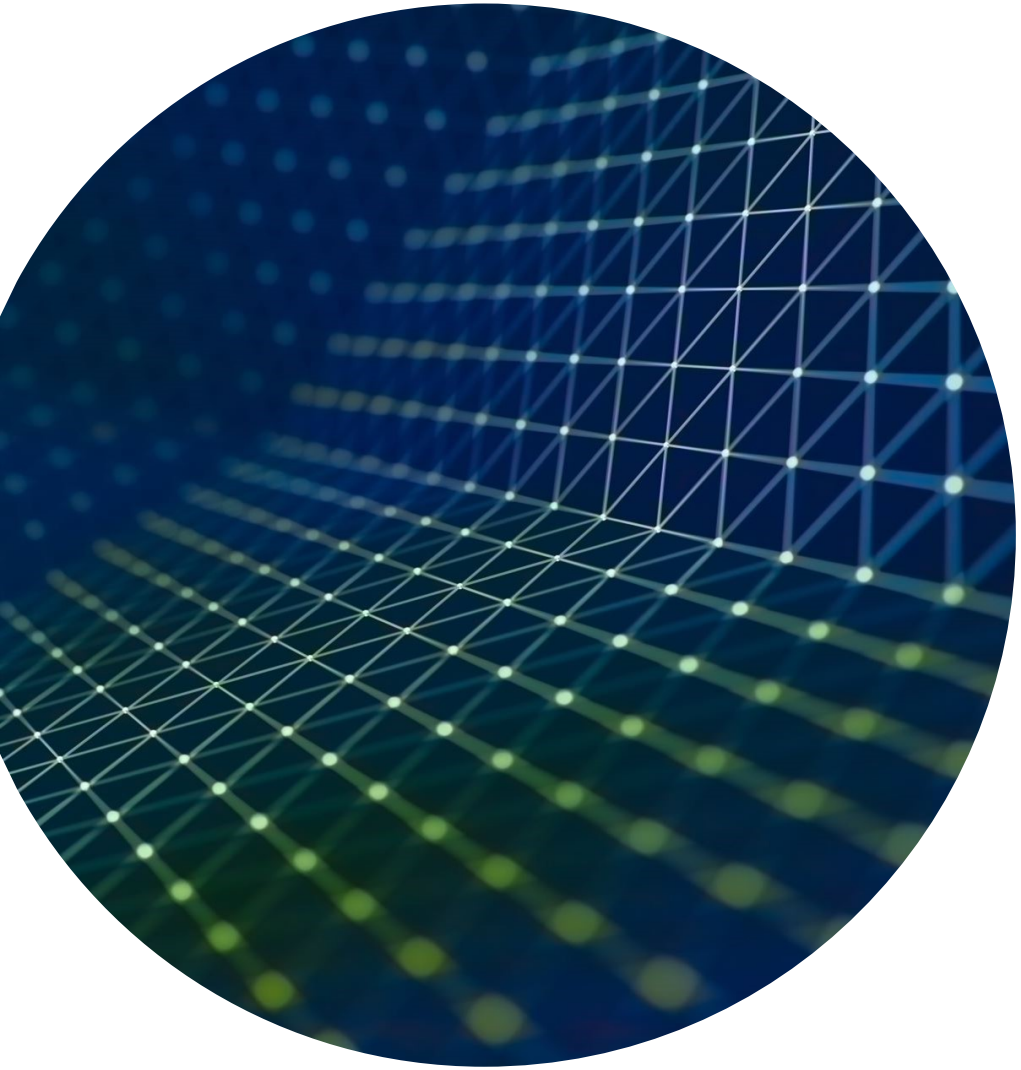
Eg: AF *8= E(L(AF)+L(8))==>E(L(AF)+L(08))

=E((B7+4B))=E(1011 0111+01001011)=E(1|0000|0010)

=E(102)=E(102-FF)=E(1|0000|0010-1111|1111)-E(0|0000|0011)

=E(03)

=0F



Yet Another option??

The multiplication part follow the following mentioned rules:

- Multiplication with 1: No change in the value
- Multiplication with 2: Left shift with no carry. In addition to that XOR with `1B` if the shifted bit is 1.
- Multiplication with 3: Perform multiplication with 2 then XOR with the initial value.
- While the addition part is simply XOR.

Direct Galois Field Multiplication

- Eg: {02} • {87} = (1000 0111)=
- After left shift=1 0000 1110 mod {11b}= (1 0000 1110) = So (0000 1110)xor (1 0001 1011) = (00001 0101)=15
- Using reducing polynomial $x^8 + x^4 + x^3 + x + 1$for multiplication in GF(2^8)
- (02)*(87)=Perform AF *8

EXAMPLE

Eg: {02} • {87} =E(L(02)+L(87))

=E(19+74)

=E(00011001+01110100)

=E(10001101)

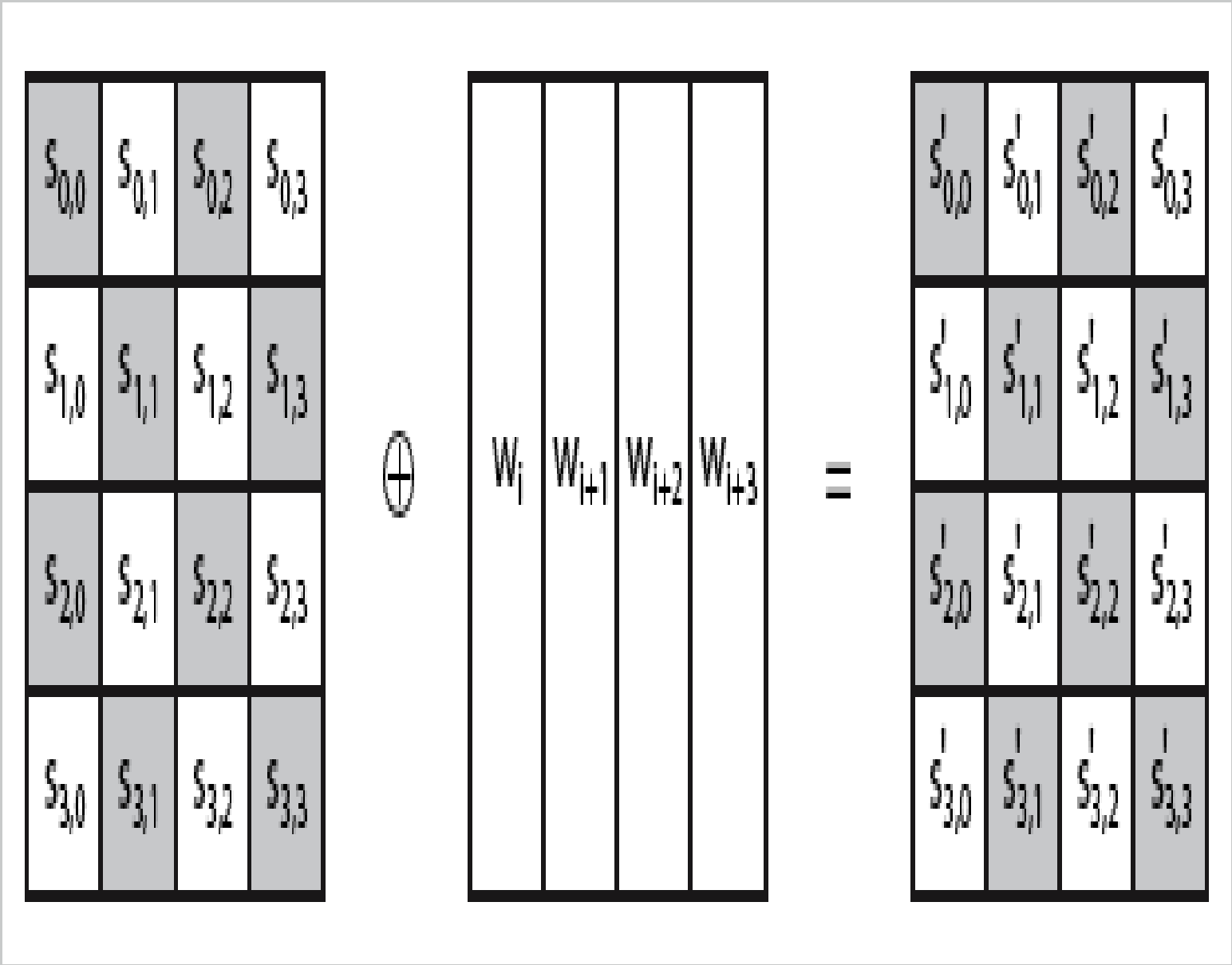
=E(8D)=15=00010101

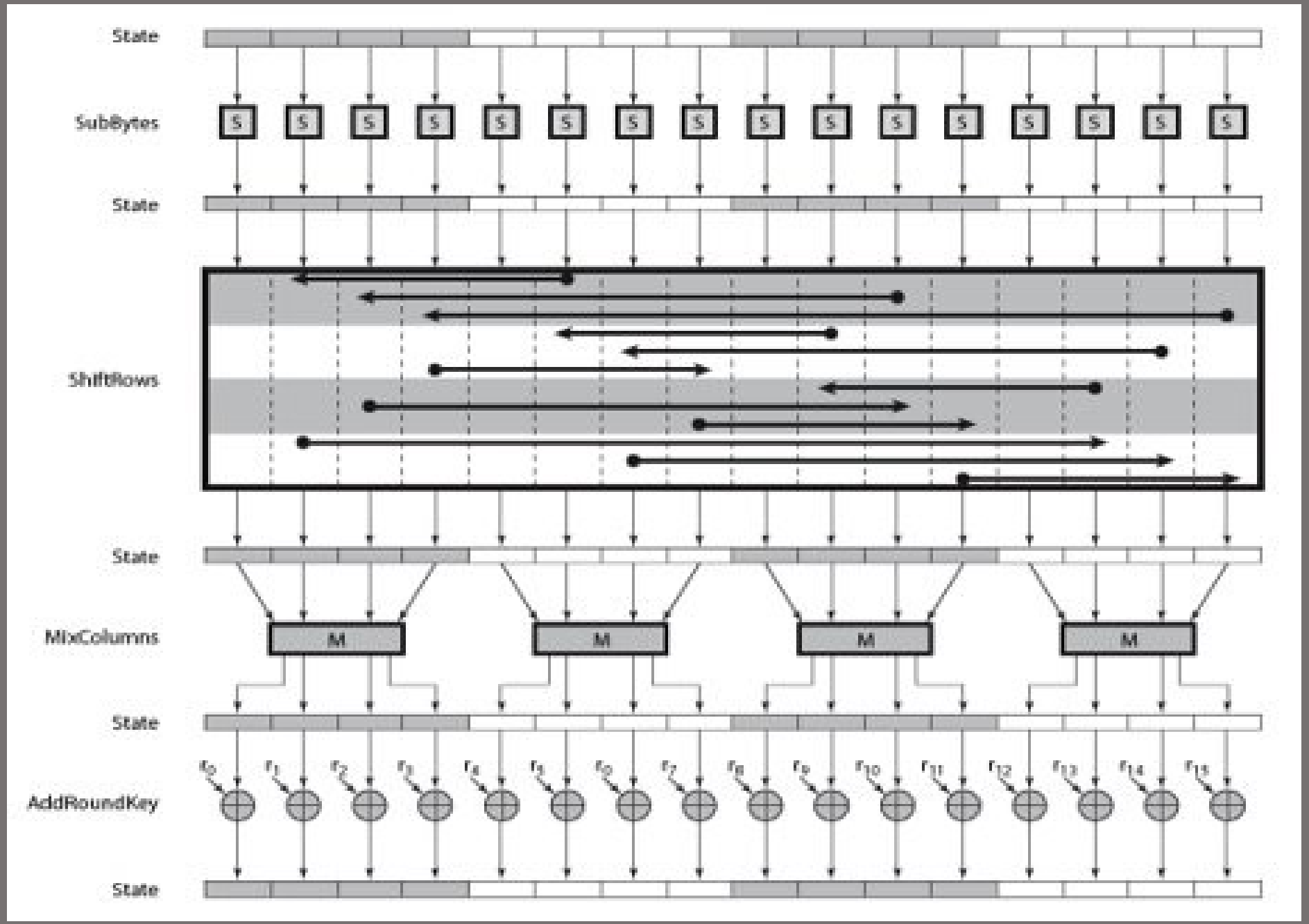
Regular addition not XOR

{02} • {87} ⊕ {03} • {6E} ⊕ {46} • {01} ⊕ {A6} • {01}=0001 0101 ⊕ 1011 0010⊕ 0100 0110⊕ 10100110=01000111= {47}

4. AddRoundKey

- XOR state bytes with 128-bits (16-byte long)of the sub-key generated in each round.
- Again processed by column (though effectively a series of byte operations)
- inverse for decryption is identical since XOR own inverse, with reversed keys
- RoundKey(subkey) determined by **key schedule** algorithm




State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
State

AES Round

AES KEY

AES KEY

- AES requires 10 rounds it will need 10 keys and 1 more key for OTI.
- In all **eleven keys** are required. Out of 11 arrays 1 array is used for initialization and other 10 arrays are used in 10 rounds one per round
- 16 byte keys are copied into initial state array which is to be extended for further 10 arrays where each array contains 4 rows and 4 columns
- So the 16 byte key is expanded to get the actual block ie the 16 byte key is expanded into a key containing 4*4 entries which is 4 words ie, 11X4X4=176bytes =44 words. A word means 4 bytes.
- If number of round is Nr, Key expansion routine creates (Nr+1) 128-bit round keys from one single 128 bit cipher key.

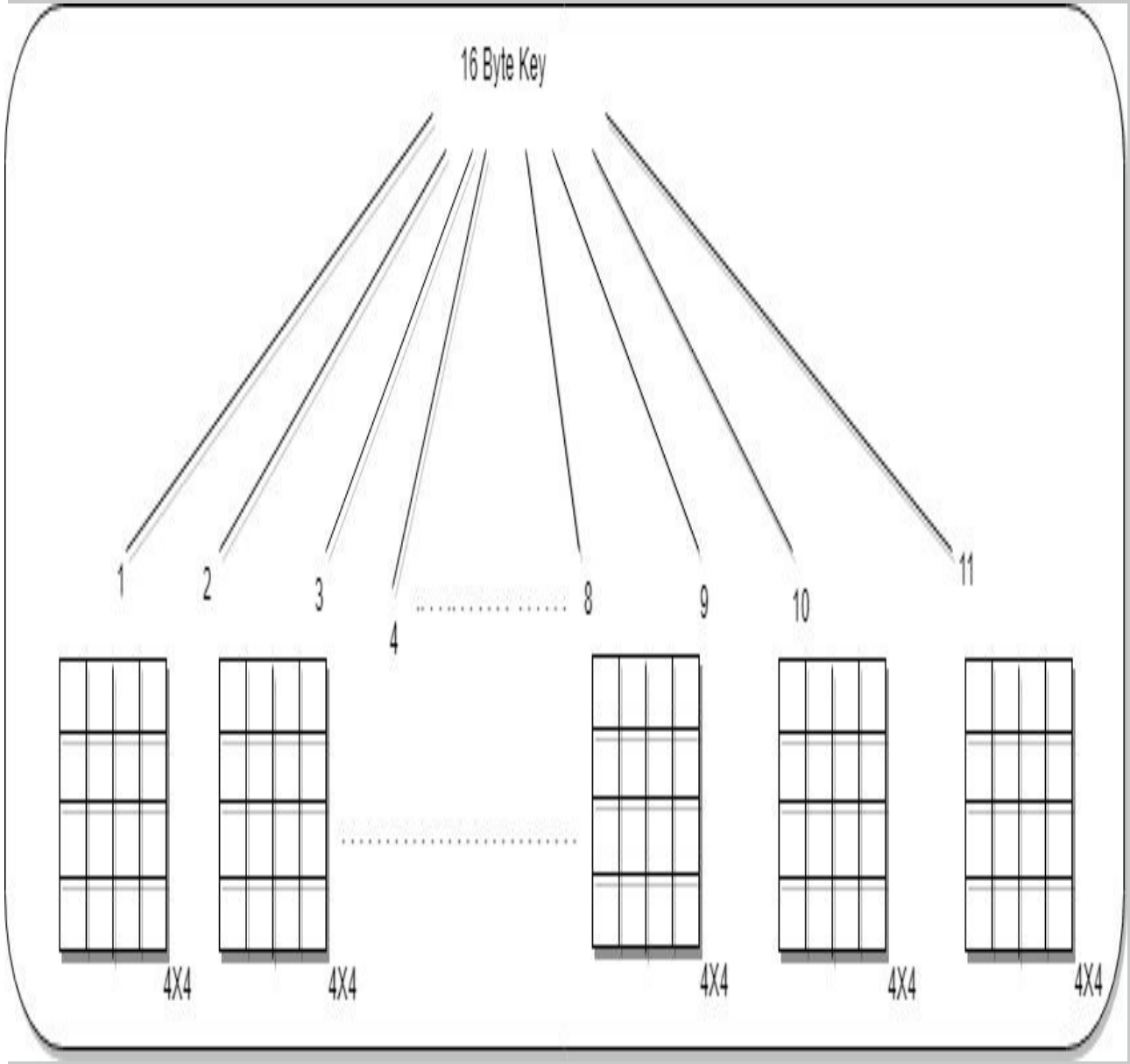


Table 7.3 *Words for each round*

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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

AES Key Schedule Algorithm

Key expansion routine creates round keys word by word(array of 4 bytes) ie W0, W1....W4(Nr+1)-1

ALGORITHM

- 16 byte keys are copied into 11 arrays where each array contains 4 rows and 4 columns. Key is expanded to 11X4X4=176bytes ie 44 words. A word means 4 bytes. The first 4 words(w0, w1,w2, w3) are made from cipher key(K0,K1,…,K16).
- The rest of the words are made as follows(wi for i=4 to 43)
  - If 'i' isn't a multiple of 4 ie , **i mod 4!=0** $W_i = W_{i-1}$ **XOR** W_{i-4}
  - If 'i' is a multiple of 4 ie , **i mod 4==0** $W_i = t$ **XOR** W_{i-4} . Here **t** is a temporary word derived as follows from last 4 bytes of key ie Wi-1.

t=SubWord(RotWord(W_{i-1})) ⊕ $RCon_{i/4}$

1.Rotate(Rotword) : Contents of the word array undergoes **left circular shift byte by byte** or 8 bits. Eg:[B1,B2,B3,B4]-->[B2,B3,B4,B1]

2.Substitute(subword) :**Byte by byte** substitution using **S-box.**

- ***Round Constatnt(Rcon Operation):*** **Each word is XORed** with a **constant** $RCon_{i/4}$**. Constant is a word** consisting of **4 –bytes.** Value of constant depends on round number. **Last 3 bytes** of constant words **consists of 0.**(so just XOR first byte) **2** raised to the power number equal to *(number of current iteration - 1)*

Key Expansion in AES-128

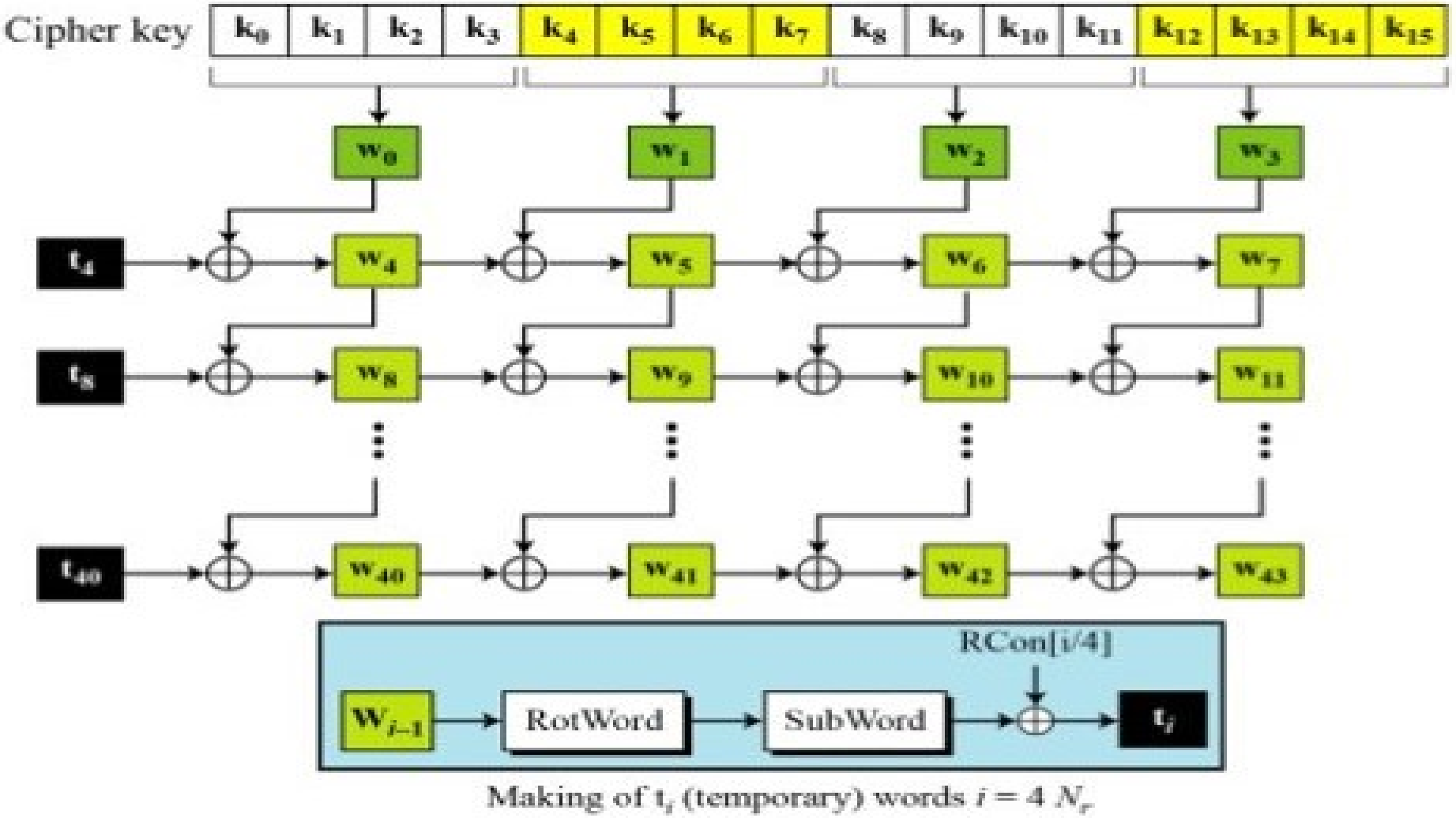
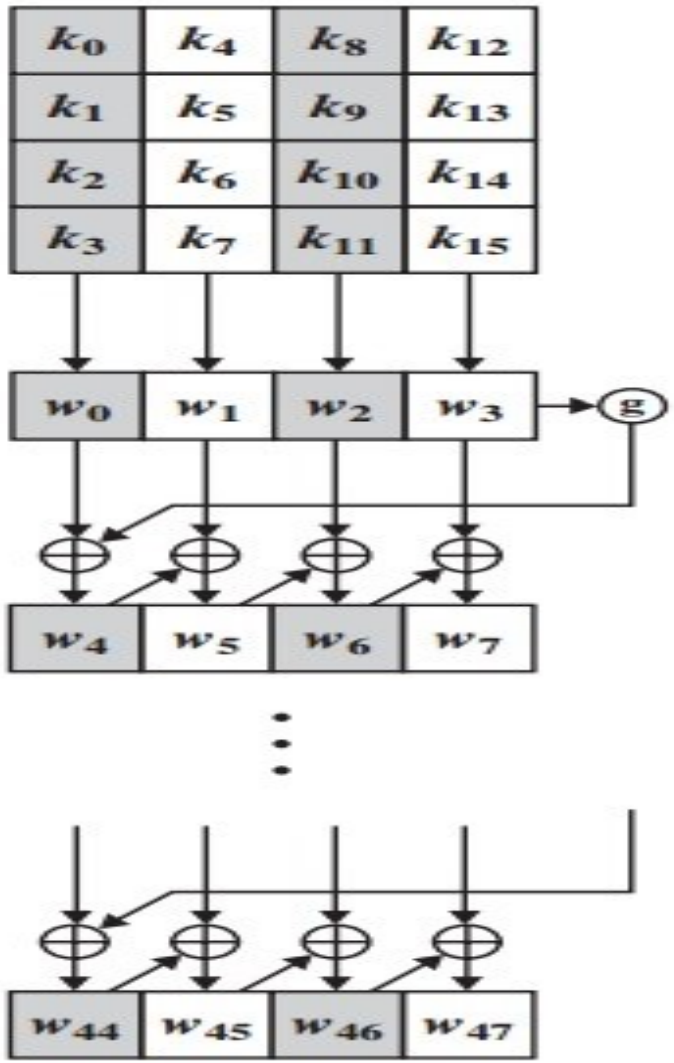
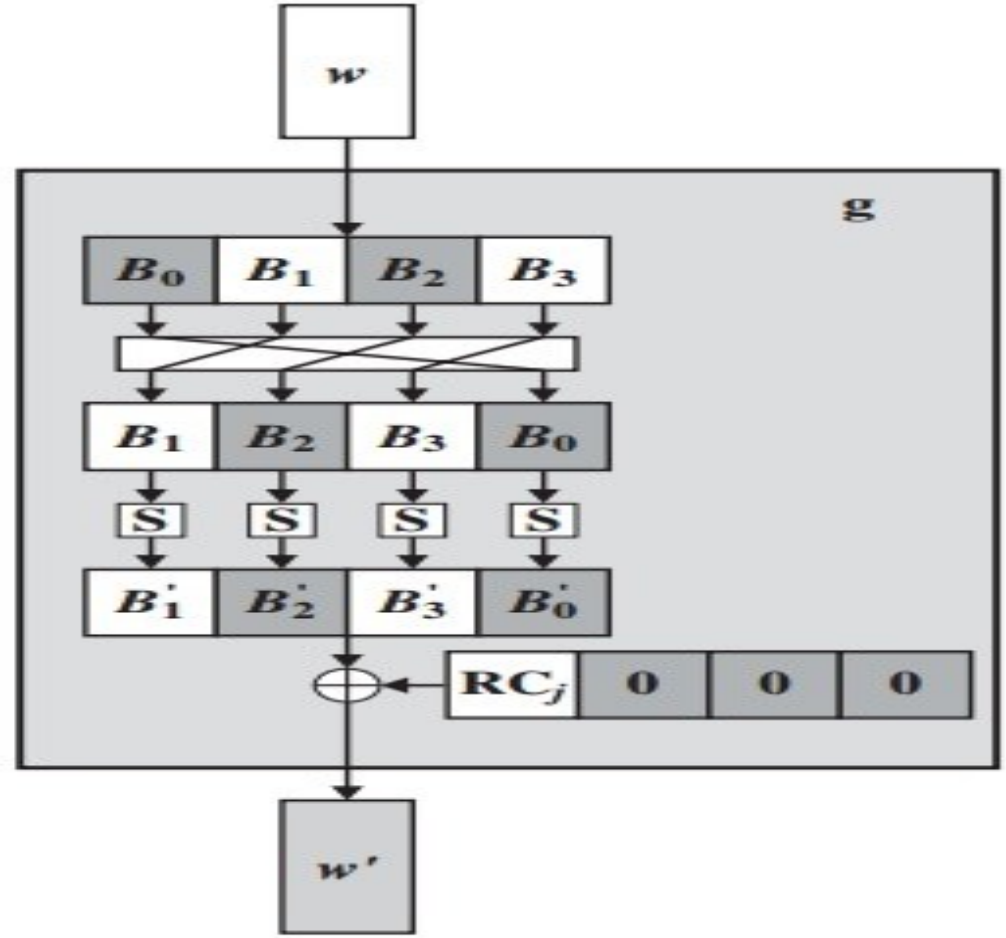


Figure 11: ***Key expansion in AES***



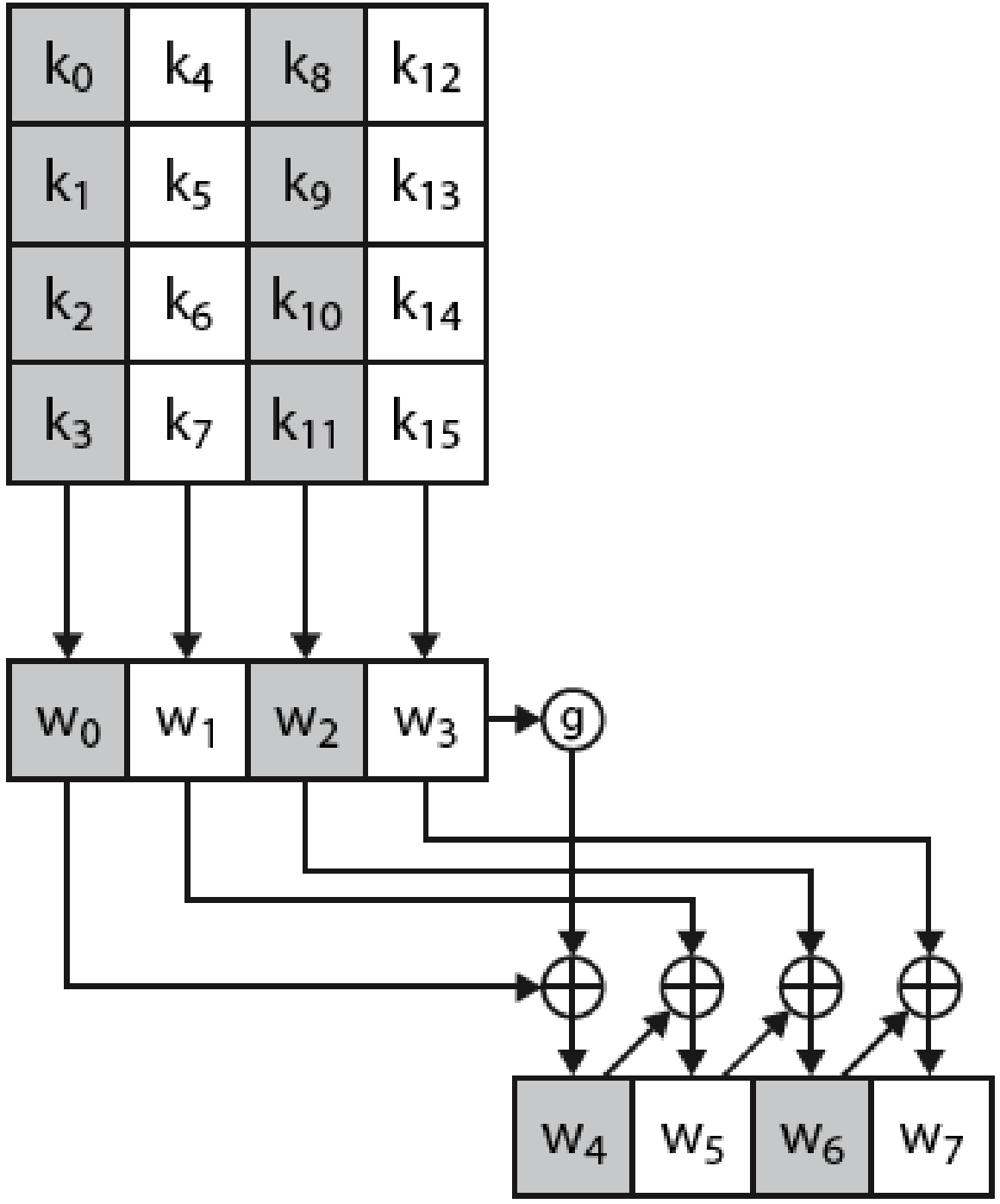
(a) Overall algorithm



(b) Function g

Figure 5.9 AES Key Expansion

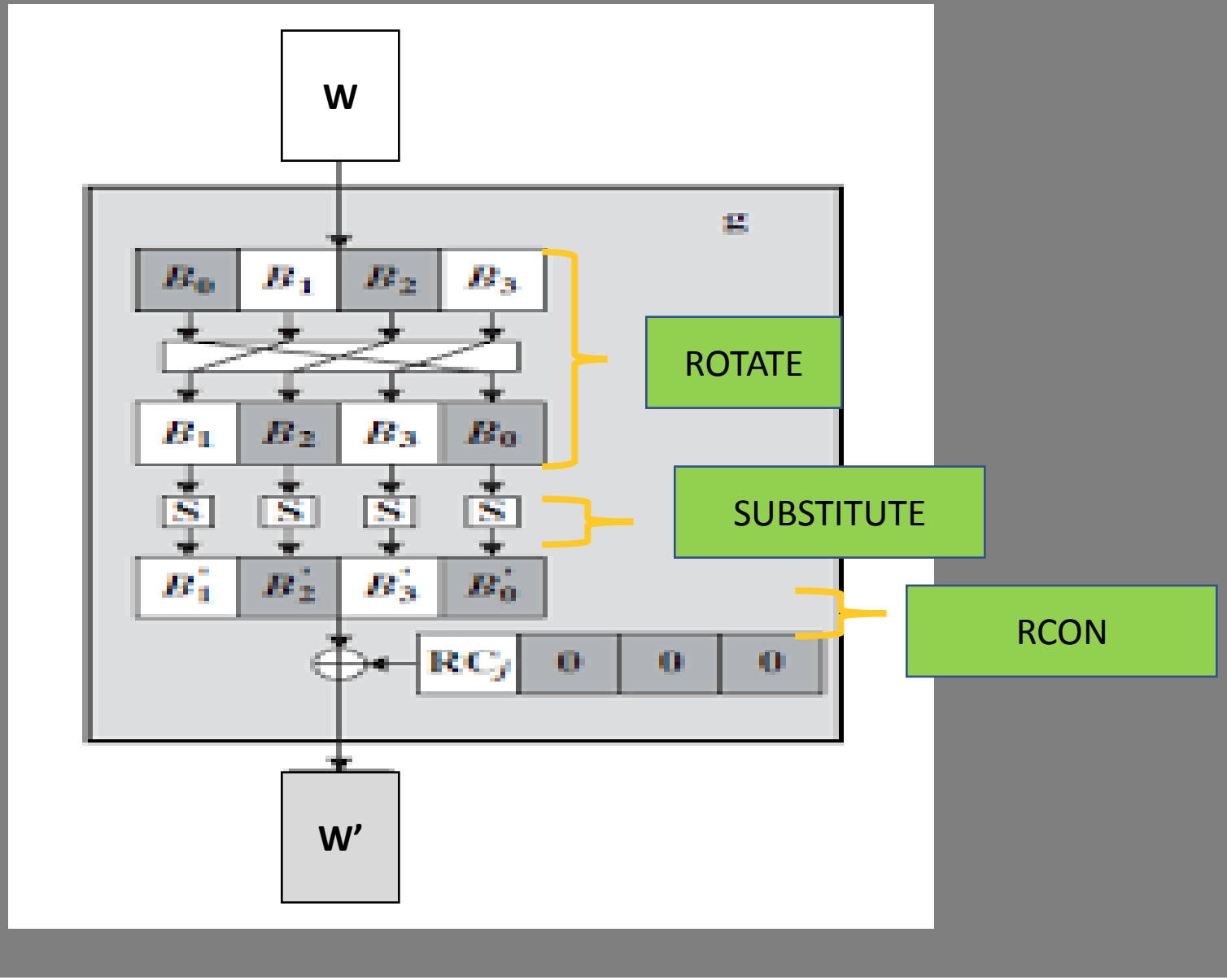
AES Key Expansion



	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

S-box table for key expansion

G-Function


w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B_1' B_2' B_3' B_0'
RC$_j$ 0 0 0
w'
ROTATE
SUBSTITUTE
RCON

Rcon Table

Rcon Constants (Base 16)			
Round	Constant(Rcon)	Round	Constant(Rcon)
1	01 00 00 00	6	20 00 00 00
2	02 00 00 00	7	40 00 00 00
3	04 00 00 00	8	80 00 00 00
4	08 00 00 00	9	1B 00 00 00
5	10 00 00 00	10	36 00 00 00

Powers of x = 0x02										
J	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

How leftmost byte is calculated dynamically?

RC_1	$\rightarrow x^{1-1}$	$= x^0$	mod *prime*	$= 1$	$\rightarrow 00000001$	$\rightarrow 01_{16}$
RC_2	$\rightarrow x^{2-1}$	$= x^1$	mod *prime*	$= x$	$\rightarrow 00000010$	$\rightarrow 02_{16}$
RC_3	$\rightarrow x^{3-1}$	$= x^2$	mod *prime*	$= x^2$	$\rightarrow 00000100$	$\rightarrow 04_{16}$
RC_4	$\rightarrow x^{4-1}$	$= x^3$	mod *prime*	$= x^3$	$\rightarrow 00001000$	$\rightarrow 08_{16}$
RC_5	$\rightarrow x^{5-1}$	$= x^4$	mod *prime*	$= x^4$	$\rightarrow 00010000$	$\rightarrow 10_{16}$
RC_6	$\rightarrow x^{6-1}$	$= x^5$	mod *prime*	$= x^5$	$\rightarrow 00100000$	$\rightarrow 20_{16}$
RC_7	$\rightarrow x^{7-1}$	$= x^6$	mod *prime*	$= x^6$	$\rightarrow 01000000$	$\rightarrow 40_{16}$
RC_8	$\rightarrow x^{8-1}$	$= x^7$	mod *prime*	$= x^7$	$\rightarrow 10000000$	$\rightarrow 80_{16}$
RC_9	$\rightarrow x^{9-1}$	$= x^8$	mod *prime*	$= x^4 + x^3 + x + 1$	$\rightarrow 00011011$	$\rightarrow 1B_{16}$
RC_{10}	$\rightarrow x^{10-1}$	$= x^9$	mod *prime*	$= x^5 + x^4 + x^2 + x$	$\rightarrow 00110110$	$\rightarrow 36_{16}$

Key Expansion Example

- **How the keys W04,W05,W06 W07 are calculated assuming that the 128-bit cipher key agreed upon by Alice and Bob is (24 75 A2 B3 | 34 75 56 88 | 31 E2 12 00 | 13 AA 54 87)$_{16}$?**

Key Expansion Example

- ***t4=SubWord(RotWord(W_{i-1})) ⊕ RCon1***
- ***t4=SubWord(RotWord(* **13 AA 54 87***)) ⊕ RCon1***
- ***t4=SubWord(* **AA 54 87 13***) ⊕ RCon1***
- ***t4=SubWord(* **AA 54 87 13***) ⊕ RCon1***

***= (*AC 20 17 7D) ⊕ (01 00 00 00)= AD 20 17 7D**

- **W04= t4 ⊕ W0= AD 20 17 7D ⊕ 24 75 A2 B3=** 89 55 B5 CE
- **W05= W04 ⊕ W01=**89 55 B5 CE ⊕**34 75 56 88=** bd20e346
- **W06= W05 ⊕ W02=** bd20e346⊕**31 E2 12 00 =**8cc2f146
- **W07= W06 ⊕ W03=** 8cc2f146⊕**13 AA 54 87=**9f68a5c1

Table 7.5 *Key expansion example*

Round	Values of t's	First word in the round	Second word in the round	Third word in the round	Fourth word in the round
—		w_{00} = 2475A2B3	w_{01} = 34755688	w_{02} = 31E21200	w_{03} = 13AA5487
1	AD20177D	w_{04} = 8955B5CE	w_{05} = BD20E346	w_{06} = 8CC2F146	w_{07} = 9F68A5C1
2	470678DB	w_{08} = CE53CD15	w_{09} = 73732E53	w_{10} = FFB1DF15	w_{11} = 60D97AD4
3	31DA48D0	w_{12} = FF8985C5	w_{13} = 8CFAAB96	w_{14} = 734B7483	w_{15} = 2475A2B3
4	47AB5B7D	w_{16} = B822deb8	w_{17} = 34D8752E	w_{18} = 479301AD	w_{19} = 54010FFA
5	6C762D20	w_{20} = D454F398	w_{21} = E08C86B6	w_{22} = A71F871B	w_{23} = F31E88E1
6	52C4F80D	w_{24} = 86900B95	w_{25} = 661C8D23	w_{26} = C1030A38	w_{27} = 321D82D9
7	E4133523	w_{28} = 62833EB6	w_{29} = 049FB395	w_{30} = C59CB9AD	w_{31} = F7813B74
8	8CE29268	w_{32} = EE61ACDE	w_{33} = EAFE1F4B	w_{34} = 2F62A6E6	w_{35} = D8E39D92
9	0A5E4F61	w_{36} = E43FE3BF	w_{37} = 0EC1FCF4	w_{38} = 21A35A12	w_{39} = F940C780
10	3FC6CD99	w_{40} = DBF92E26	w_{41} = D538D2D2	w_{42} = F49B88C0	w_{43} = 0DDB4F40

AES Decryption

- To decrypt, process must be invertible
- Inverse of MixAddRoundKey is easy, since "⊕" is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)

AES Security

- **Brute Force Attack:** With 128 bit attacker has to try 2^{128} and AES lacks weak keys.A PC that tries 255 keys per second needs 149.000 billion years to break AES
- 192 bit: 2^{192} and 256 bit: 2^{256} possible keys
- **Statistical Attack:** Strong confusion and diffusion provided by combination of Subbytes, ShiftRows and Mix Column operation removes any frequency pattern in plain text
- **Differential and Linear cryptanalysis:** AES is resistant to Linear and differential cryptanalysis. But some attacks like related key cryptanlysis and biclique cryptanlysis was successful but it was so complicated.

Encryption Key Power

Number of Bits in Key	Odds of Cracking: 1 in	Estimated Time to Crack*
8	256	.000032 seconds
16	65,536	.008192 seconds
24	16,777,216	2.097 seconds
32	4,294,967,296	8 minutes 56.87 seconds
56	72,057,594,037,927,900	285 years 32 weeks 1 day
64	18,446,744,073,709,600,000	8,090,677,225 years
128	3.40282E+38	5,257,322,061,209,440,000,000 years
256	1.15792E+77	2,753,114,795,116,330,000,000,000,000, 000,000,000,000,000,000,000,000 years
512	1.3408E+154	608,756,305,260,875,000,000,000,000,000, 000,000,000,000,000,000,000,000,000,000, 000,000,000,000,000,000,000,000,000,000, 000,000,000 years

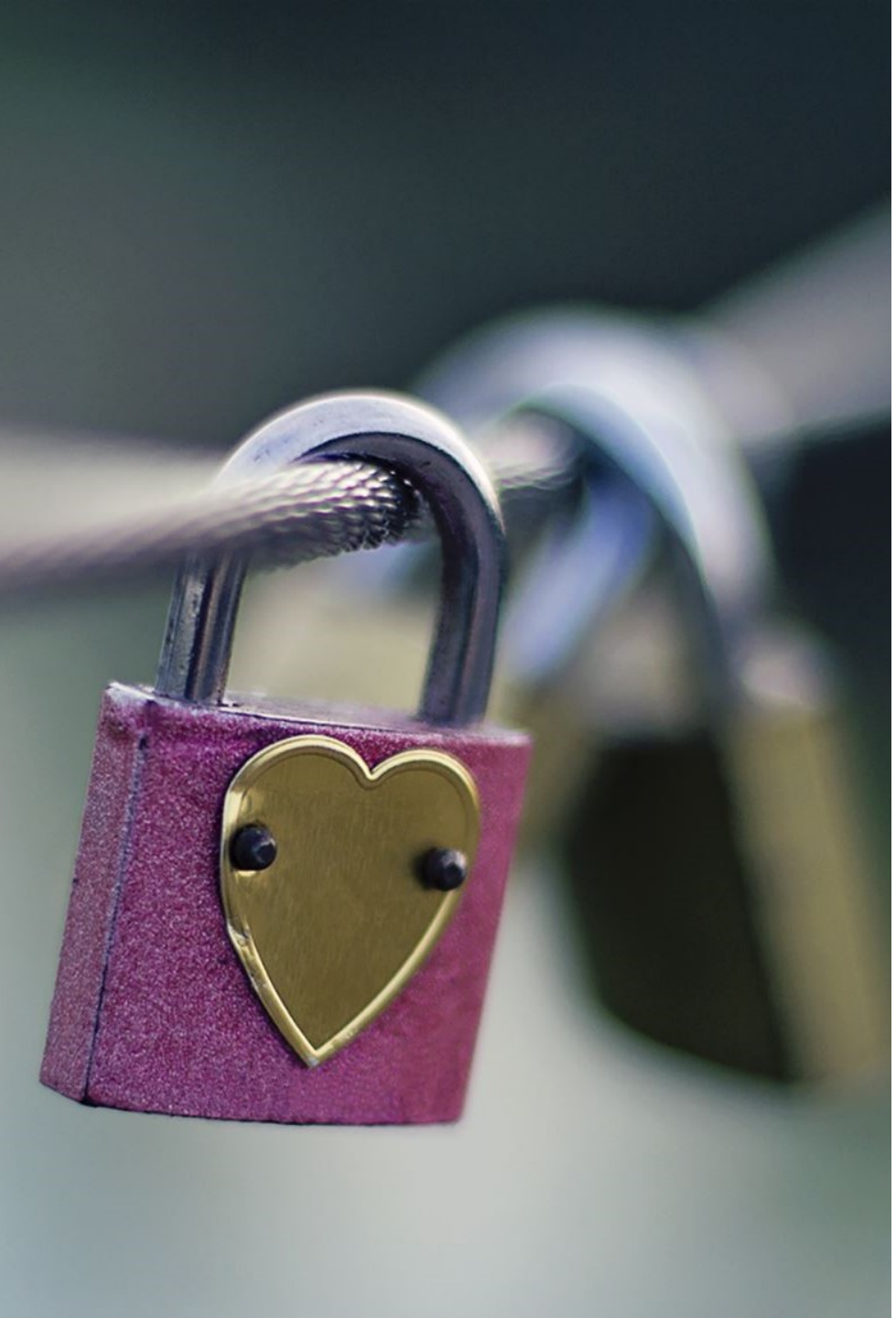
[NOTE] *Estimated Time to Crack is based on a general-purpose personal computer performing eight million guesses per second.

Problems with symmetric key cryptography

Problems with symmetric key cryptography

- **Sharing the Key securely**
- **More Damage if Compromised:** Both sides of the conversation get compromised as same key is used for encryption as well decryption
- Cannot provide digital signatures that cannot be repudiated

Key Exchange Protocols



Key Exchange Protocols

- A **"key exchange"** protocol is used for
  - To establish a shared symmetric key . ***Not*** for encrypting or signing
  - Requires no prior secrets
  - Real-time over an untrusted network
  - Exponential key agreement
- A protocol between two parties to establish a shared key ("session key") such that:
  1. **Authenticity:** they both know who the other party is
  2. **Secrecy:** only they know the resultant shared key
  3. **Consistency:** if two honest parties establish a common session key then both have a consistent view of who the peers to the session are

Shared key generation using Diffie-Hellman key exchange protocol

- Discovered by Whitfield **D**iffie and Martin **H**ellman
- **Diffie–Hellman key exchange** is a method of securely exchanging cryptographic keys over a pu channel
- Based on **difficulty in calculating discrete log problem**(primitive root of a prime number p g all prime numbers from 1 to p-1) in a finite field
  - **Given: g, p,** and g^k **mod p**
  - **Find:** exponent **k**
- Diffie-Hellman is currently used in many protocols, namely:
  - Secure Sockets Layer (SSL)/Transport Layer Security (TLS)
  - Secure Shell (SSH)
  - Internet Protocol Security (IPSec)
  - Public Key Infrastructure (PKI)

Discrete Logarithm Problem

- Discrete logarithms are logarithms defined with regard to **multiplicative cyclic groups(**discrete logarithm based crypto-systems is Z_p^* where p is a prime numbe)**.**
- If ***G*** is a **multiplicative cyclic group** and ***g*** is a **generator of** *G*, then from the definition of cyclic groups, we know **every element *h* in *G*** can be written as g^x **for some *x***.
- The discrete logarithm to the base *g* of *h* in the group *G* is defined to be *x* . For example, if the group is Z_5^*, and the generator is 2, then the discrete logarithm of 1 is 4 because $2^4 \equiv 1 \bmod 5$.
- The discrete logarithm problem is defined as: **given a group *G*, a generator *g* of the group and an element *h* of *G*, to find the discrete logarithm to the base *g* of *h* in the group *G*.**
- Discrete logarithm problem is not always hard. The hardness of finding discrete logarithms depends on the groups.
- Eg: discrete logarithm based crypto-systems is Z_p^* where p is a prime number.

Discrete Logarithm Problem

- It's easy to compute powers modulo a prime

Eg:Find **all** the powers of 2 up to 2^{10} , each modulo 11.

  - 2^0 Mod 11=**1**
  - 2^1 Mod11=**2**
  - 2^2 Mod11=**4**
  - 2^3 Mod11=**8**
  - 2^4 Mod11=**5**
  - 2^5 Mod11=**10**
  - 2^6 Mod11=**9**
  - 2^7 Mod11=**7**
  - 2^8 Mod11=**3**
  - 2^9 Mod11=**6**
  - 2^{10} Mod11=**1**

- Reverse the process: **which** power of 2 modulo 11 is 7,ie Find for which x 2^x Mod11=7.
- Powers of 2 run through all possible remainders where 2 can be called the **generator.** For a large **prime** modulus, then there is always **some** generator, which generates number from 0 to p-1

Discrete Logarithm Problem

If g is a generator of Z_n^* then the equation $g^x \equiv g^y \pmod{n}$ holds if and only if the equation $x \equiv y \pmod{\Phi(n)}$ holds

Given g, h ∈ G, find an x (if it exists) such that $g^x = h$.

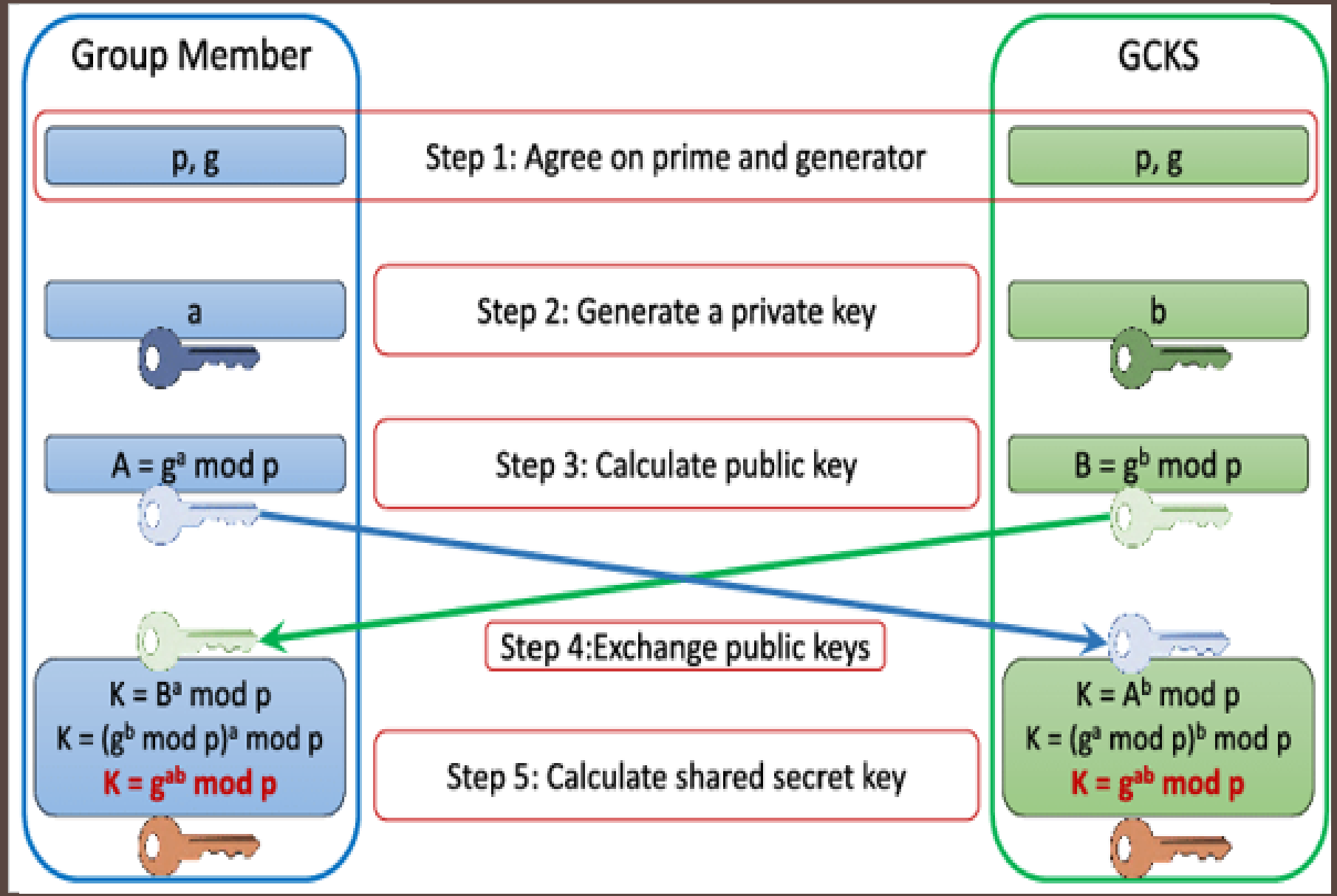
Eg: $3^n \equiv 17 \bmod 101$. Find n?

DIFFIE –HELLMAN ALGORITHM

- P and G are both publicly available numbers. Let p be prime, let g be a **generator** .g is a generator iff for all $0 < x \neq y < p$, $g^x \neq g^y \pmod p$.p is quite large, usually 2000 or more commonly 4000 bits long.
  - For any $x \in \{1,2,\ldots,p-1\}$ there is n s.t. $x = g^n \bmod p$ where P is at least 512 bits

1. Alice selects her private value a and Bob selects his private value b
2. Compute public values x and y where $1 < x < p-1$; $1 < y < p-1$, and exchange
   - Alice computes $x = g^a \bmod p$ and sends to Bob
   - Bob computes$y = g^b \bmod p$ and sends to Alice
3. Both compute shared Private key, $g^{ab} \bmod p$
   - $k_a = y^a \bmod p = g^{ab} \bmod p$
   - $k_b = x^b \bmod p = g^{ab} \bmod p$
4. Shared secret $g^{ab} \bmod p$ can be used as symmetric key

- Algebraically it can be shown that $k_a = k_b$ .Users now have a symmetric secret key to encrypt
- **Public Key:** g and p
- **Private Key:** Alice's exponent a, Bob's exponent b



Diffie-Hellman

Diffie Hellman Challenge

- In a Diffie-Hellman Key Exchange, Alice and Bob have chosen **prime value = 23**. If Alice's secret key is **3** and Bob's secret key is **6**, what is the secret key they exchanged?

EXAMPLE 1

Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume g = 7 and p = 23. The steps are as follows:

1. Alice chooses a = 3 and calculates $x = 7^3 \bmod 23 = 21$.
2. Bob chooses b = 6 and calculates $R_y = 7^6 \bmod 23 = 4$.
3. Alice sends the number 21 to Bob.
4. Bob sends the number 4 to Alice.
5. Alice calculates the symmetric key $K = 4^3 \bmod 23 = 18$.
6. Bob calculates the symmetric key $K = 21^6 \bmod 23 = 18$.

- The value of K is the same for both Alice and Bob; $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$.

Diffie Hellman Challenge

- In a Diffie-Hellman Key Exchange, Alice and Bob have chosen **prime value q = 17**. If Alice's secret key is **4** and Bob's secret key is **6**, what is the secret key they exchanged?

EXAMPLE 2

- Alice and Bob get public numbers
  - P = 17, G = 5
- Alice and Bob compute public values using a=4 and b=6 which are secret
  - $X = 5^4 \bmod 17 = 6561 \bmod 23 = 13$
  - $Y = 5^6 \bmod 17 = 729 \bmod 23 = 02$
- Alice and Bob exchange public numbers
- Alice and Bob compute symmetric keys
  - $k_a = y^a \bmod p = 2^4 \bmod 23 = 16$
  - $k_b = x^b \bmod p = 13^6 \bmod 23 = 16$
- Alice and Bob now can talk securely!

Diffie Hellman Challenge

- In a Diffie-Hellman Key Exchange, Alice and Bob have chosen **prime value q = 353.** If Alice's secret key is **97** and Bob's secret key is 223, what is the secret key they exchanged?

Security of Diffie-Hellman

Prone to 2 attacks:

1. **Discrete Logarithm Attack**
2. **Man in the middle attack**

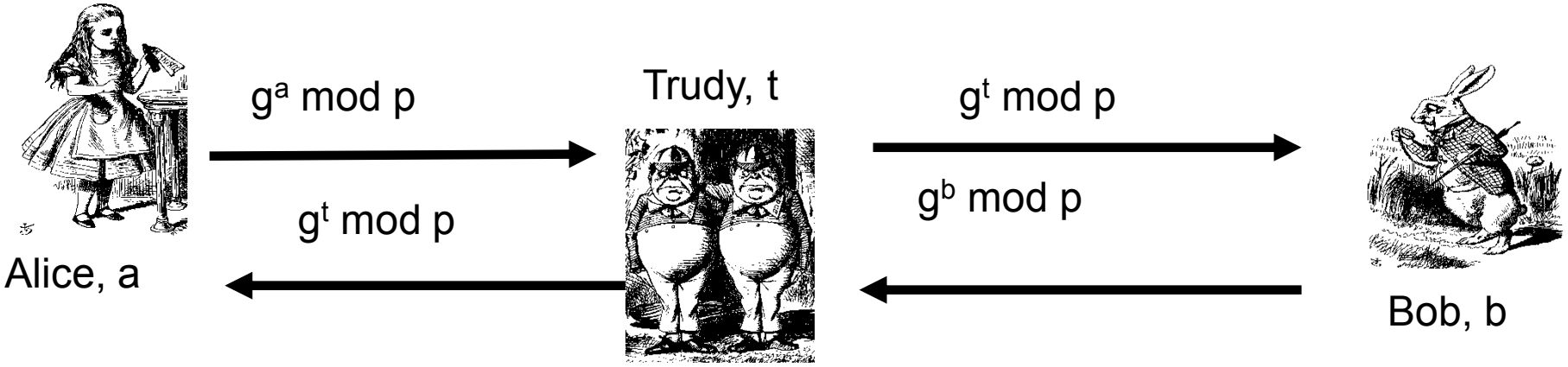
DISCRETE LOGARITHM ATTACK

- Trudy can see ga mod p and gb mod p, But… ga gb mod p = ga+b mod p ≠ gab mod p
- Security of DH-Key exchange depends on difficulty of discrete logarithm problem. If Trudy can find x from A and y from B then she can calculate symmetric key K. To prevent Discrete Logarithm Attack:

1. Chose p a prime number of around 300 digits, and a and b at least 100 digits each.
2. P should be selected in such a way p-1 has at least one large prime factor.
3. Bob and Alice must destroy x and y after calculation of symmetric key.
4. Generator must be chosen from group Z_{p*}

Man in the middle Attack on Diffie hellman

- Diffie hellman Key Exchange algorithm is susceptible Subject to man-in-the-middle (MiM) attack: Two keys instead of one. Also known as bucket brigade attack.
  - Trudy shares secret $g^{at} \bmod p$ with Alice
  - Trudy shares secret $g^{bt} \bmod p$ with Bob
  - Alice and Bob don't know Trudy is MiM



How to prevent MITM attack?

- Encrypt DH exchange with a shared symmetric key
- Encrypt DH exchange with a public key and
- Sign DH values with private keys.

Thank you

anooja@somaiya.edu