# Module: 2

## *"Block Cipher and Stream Ciphers"*

--Dr.S.P.Anandaraj

*'Associate Professor'*

*Department-CSE'*

*Presidency University, Bangalore*

# Plaintext to Ciphertext

Objectives:

- one ‘character’ in ciphertext

  = function of a large number of ‘characters’ in the plaintext.

- Thus if e is the most commonly used character in English plaintext, it may not be so in the ciphertext.

In ciphertext all the characters should have ideally an equal frequency of occurrence.

# Frequency of Patterns

For every language: frequency of characters, digrams ( two letter sequences) and trigrams are known.→ statistical analysis to decipher encrypted information.

- English: e: the character with highest frequency
- C: #define and #include in the beginning
- Protocols and tcpdump: repetitive, fixed sized fields

# Block Ciphers

used for

- Fast encryption of large amount of data
- Creating a cryptographic checksum for guaranteeing the integrity of data
- Secrecy and authentication service

# Modern Block Ciphers

- will now look at modern block ciphers
- one of the most widely used types of cryptographic algorithms
- provide secrecy and/or authentication services
- in particular will introduce DES (Data Encryption Standard)

# Types of Cipher Algorithms

- Streaming Cipher: encrypts data unit by unit, where a unit is of certain number of bits (Example: If the unit be a bit, a stream cipher encrypts data unit by unit. Or if the unit be a byte, it encrypts byte by byte)
- Block cipher: encrypts a fixed- sized block of data at a time:
  - For a 64 bit block of plaintext, for encryption to a 64-bit ciphertext, may need a table of $2^{64}$ = 150 million terabytes = $15 \times 10^{19}$ bytes
  - For a block size of 128 bits, the table would require a memory of $5 \times 10^{39}$ bytes.

# Block vs Stream Ciphers

- block ciphers process messages in into blocks, each of which is then en/decrypted
- like a substitution on very big characters
  - 64-bits or more
- stream ciphers process messages a bit or byte at a time when en/decrypting
- many current ciphers are block ciphers
- hence are focus of course

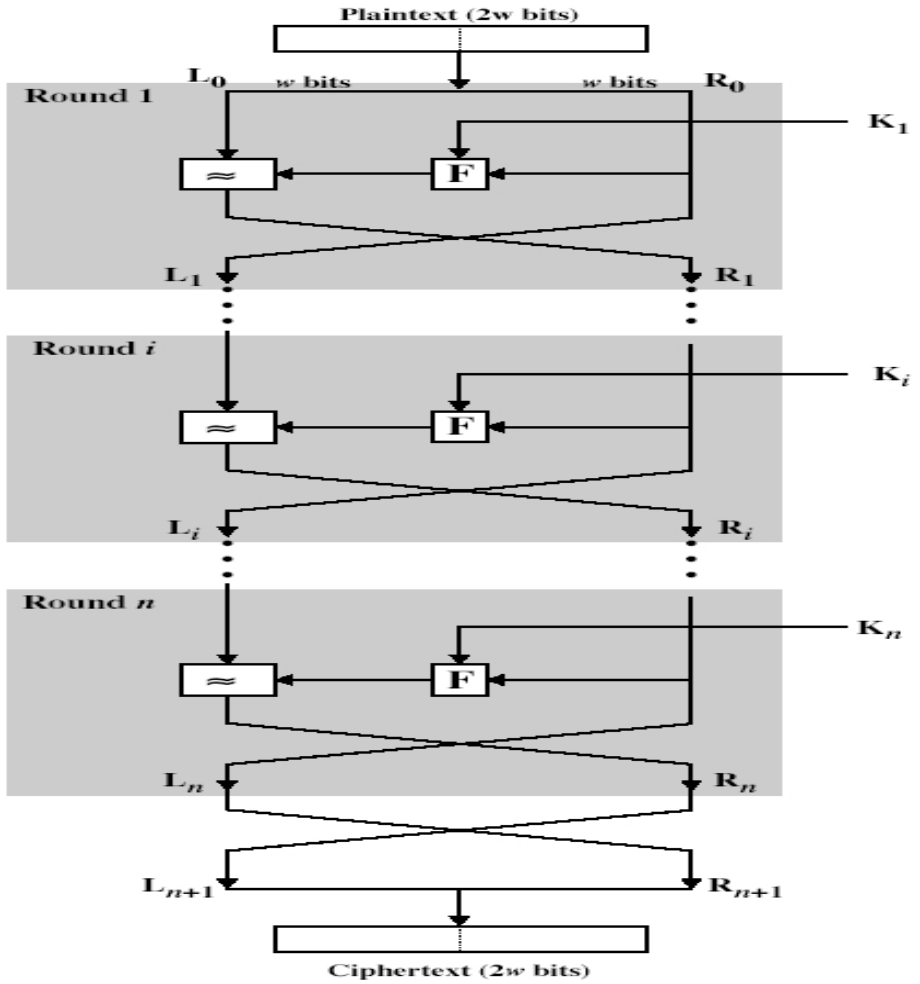
# Block Cipher Principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently
- block ciphers look like an extremely large substitution
- would need table of $2^{64}$ entries for a 64-bit block
- instead create from smaller building blocks
- using idea of a product cipher

# Feistel Cipher Structure

- Horst Feistel devised the **feistel cipher**
  - based on concept of invertible product cipher
- partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves
- implements Shannon's substitution-permutation network concept

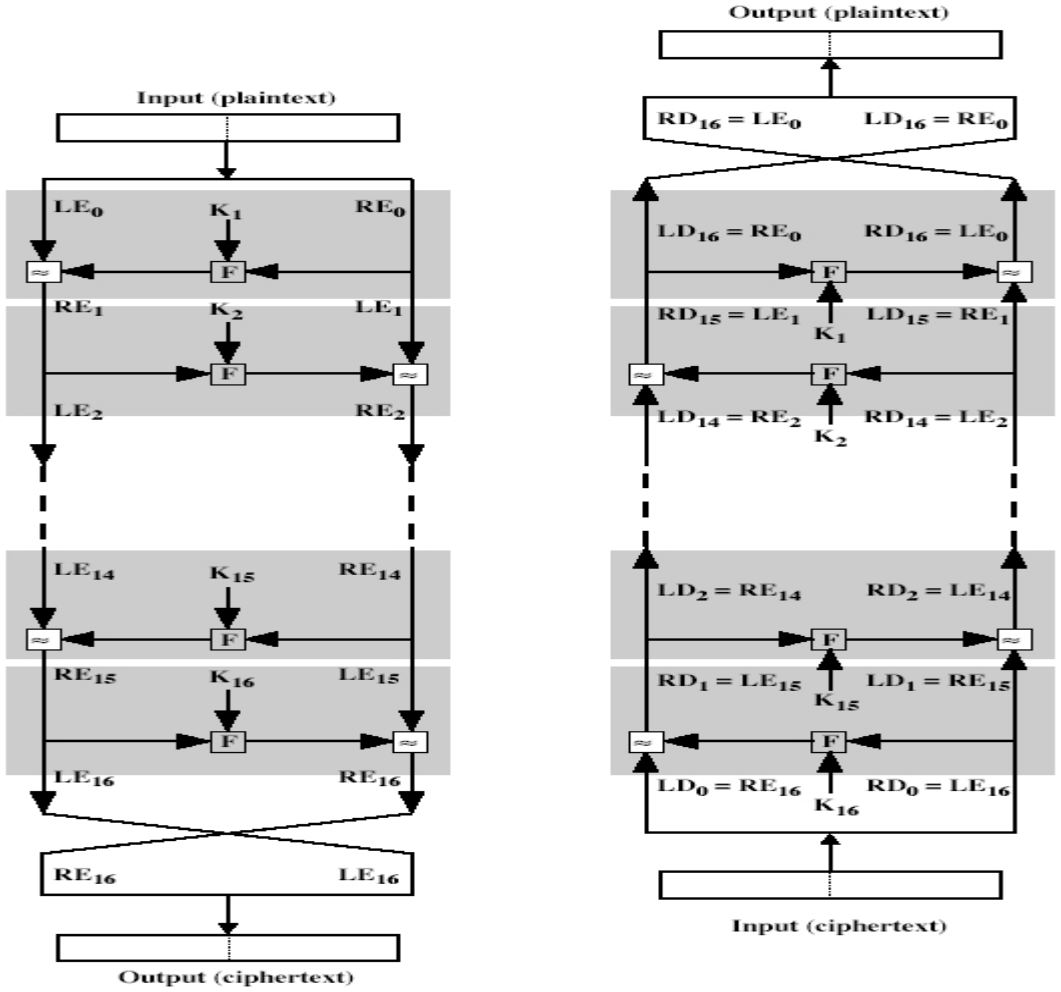
# Feistel Cipher Structure



# Feistel Cipher Design Principles

- **block size**
  - increasing size improves security, but slows cipher
- **key size**
  - increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **number of rounds**
  - increasing number improves security, but slows cipher
- **subkey generation**
  - greater complexity can make analysis harder, but slows cipher
- **round function**
  - greater complexity can make analysis harder, but slows cipher
- **fast software en/decryption & ease of analysis**
  - are more recent concerns for practical use and testing

# Feistel Cipher Decryption



# Diffusion & Confusion :

1945: “Introduce diffusion and confusion through cryptographic algorithms”, said CLAUDE SHANNON.

DIFFUSION:

- Use permutation followed by some functional transformation.
- seeks to make statistical relationship between the plaintext and ciphertext as complex as possible.
- Diffuses the structure of the plaintext over a large part of the ciphertext.

# CONFUSION

CONFUSION:

- makes the relationship between the statistics of the ciphertext and the encryption key as complex as possible.
- Achieved by using a complex substitution algorithm.

IMPORTANT: Substitution or Permutation: easy to break by using statistical analysis; strength due to non-linear functional transformation.

# Diffusion and Confusion

- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this
- more practically Shannon suggested combining elements to obtain:
- **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

# Kerckhoff's Rule

The strength of an encryption algorithm depends upon:

1. Design of the algorithm
2. Key length
3. Secrecy of the key ( requires proper management of key distribution)

1883: Jean Guillaumen Hubert Victor Fransois Alexandre Auguste Kerckhoff von Nieuwenhof: " Cryptosystems should rely on the secrecy of the key, but not of algorithm."

Advantages of Openness: 1994: A hacker published the source code of RC4, a secret encryption algorithm, designed by RSA Data security Inc. → attacks, that exposed several weaknesses of RC4

# Modern Encryption Techniques:

- DES: A rather complex encryption scheme.

- Simplified DES:
  - A teaching tool
  - Designed by Prof. Edward Schaeter, Santa Clara University, 1996
  - http://www.cs.binghamton.edu/~steflik/cs455/Simplified_DES.ppt, as of 29th Sept 2009

Given: plaintext 8-bit, Key 10-bit

Output: ciphertext 8-bit

# Simplified DES:

ciphertext = $IP^{-1}(f_{k2}(SW(f_{k1}(IP(plaintext)))))$

- <u>SDES 's five steps:</u>

1. Initial Permutation IP.
2. A complex function $f_k$ which requires key $K_1$.
3. A switch function SW which switches the left half and the right half of a data string.
4. The function $f_k$ again with a different key $K_2$.
5. A permutation function that is the inverse of IP –called $IP^{-1}$ .

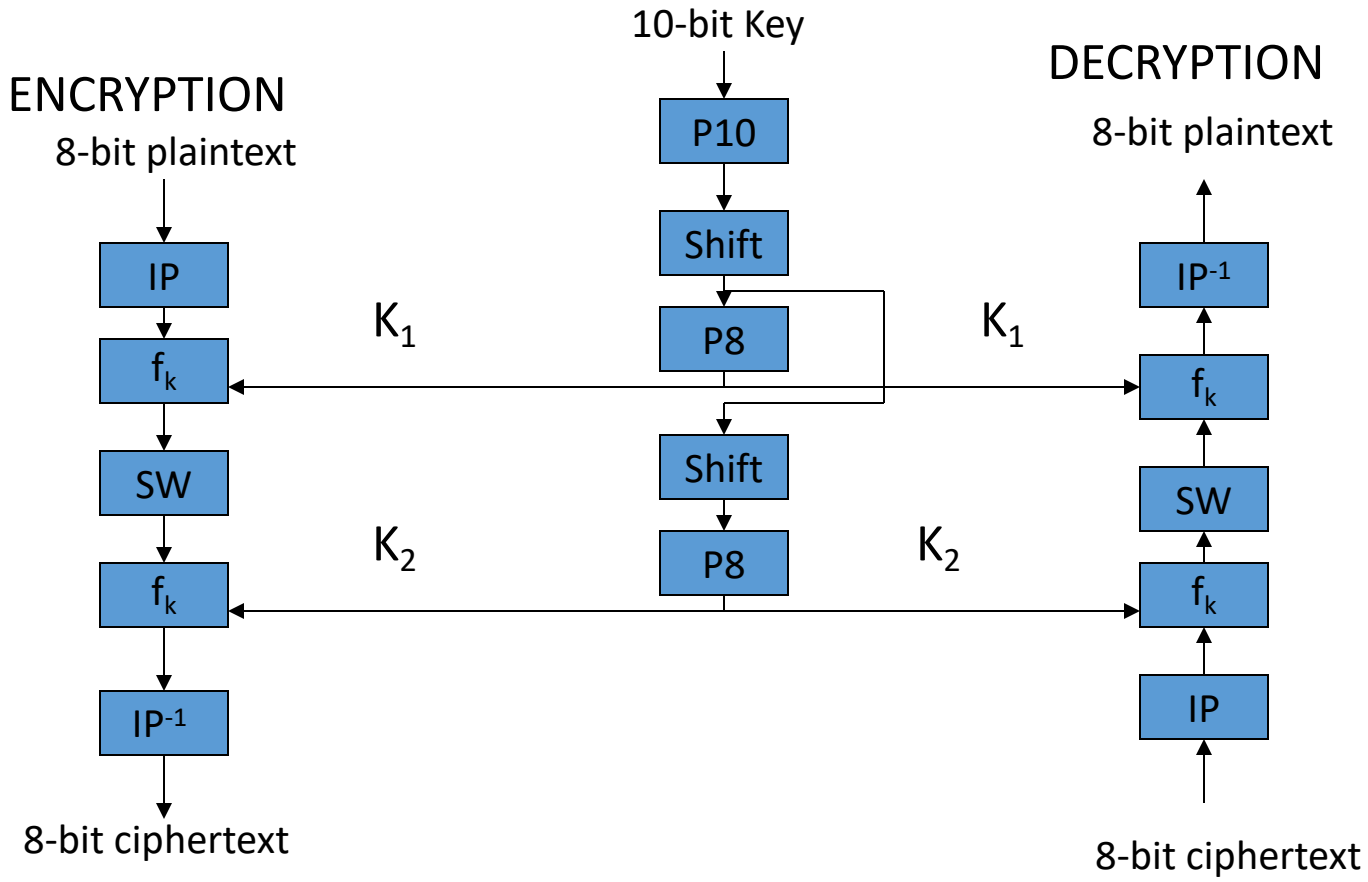
$(IP^{-1}(IP(X))) = X.$

SDES may be said to have two ROUNDS of the function $f_k$.

# Simplified DES scheme:

ciphertext = $IP^{-1}(f_{k2}(SW(f_{k1}(IP(plaintext)))))$

Plaintext = $IP^{-1}(f_{k1}(SW(f_{k2}(IP(ciphertext)))))$



# SDES (continued)

$K_1 = P8 ( Shift (P10 (Key)))$

Plaintext = $IP^{-1} (f_{k1} (SW (f_{k2} ( IP (ciphertext)))))$.

- To obtain $K_1$ and $K_2$ :
- Given: $K = (k_1\ k_2\ k_3\ k_4\ k_5\ k_6\ k_7\ k_8\ k_9\ k_{10})$
- Step1: Permutation P10

  P10 :

| 3 | 5 | 2 | 7 | 4 | 10 | 1 | 9 | 8 | 6 |
|---|---|---|---|---|---|---|---|---|---|

- Step2: Left shift (circular) by one bit
  - for the left half and
  - for the right half separately.

# To obtain K1 and K2

- Step3: Permutation for producing an 8 bit key $K_1$ from a 10 bit input.

  P8:

| 6 | 3 | 7 | 4 | 8 | 5 | 10 | 9 |
|---|---|---|---|---|---|---|---|

- Step4: Take the result of step2. On it use Left Shift (circular) by 2 bits
  - for the left half and
  - for the right half separately.
- Step5: Another instance of P8 is used to produce the second 8 bit key $K_2$ .

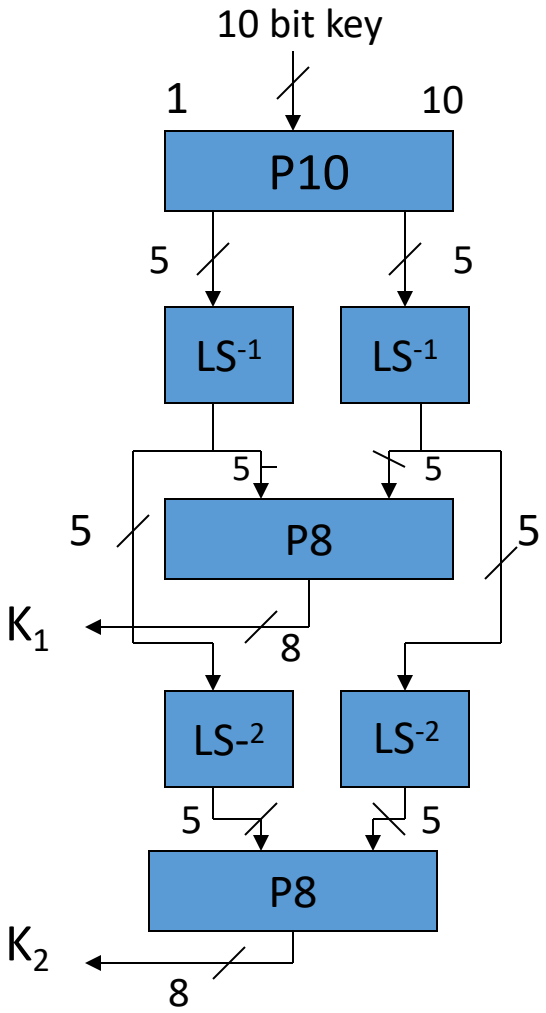
  $K_1$ = P8 ( Shift (P10 (Key)))

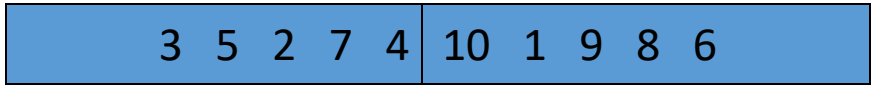
  $K_2$ = P8 ( Shift ( Shift (P10 (Key)))).

# Key generation for simplified DES:

$K_1$ = P8 ( Shift (P10 (Key)))

$K_2$ = P8 ( Shift ( Shift (P10 (Key))))




P10

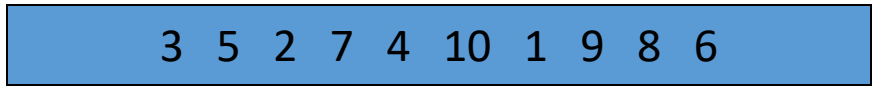
Circular left shift by 1, separately on the left and the right halves


P8

Circular left shift by 2 , separately on the left and the right halves


P8

# Example: Key Generation




P10


P8

10-bit key = 1 0 1 0 0 0 0 0 1 0

1 0 0 0 0 0 1 1 0 0 P1O

0 0 0 0 1 1 1 0 0 0 $LS^{-1}$ $LS^{-1}$

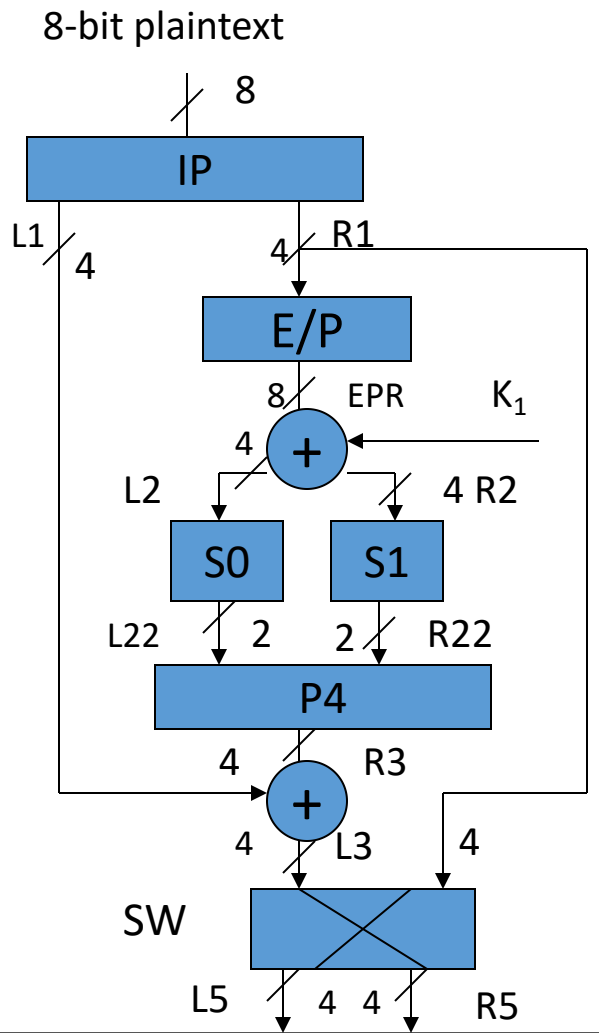
1 0 1 0 0 1 0 0 $= K_1$ P8

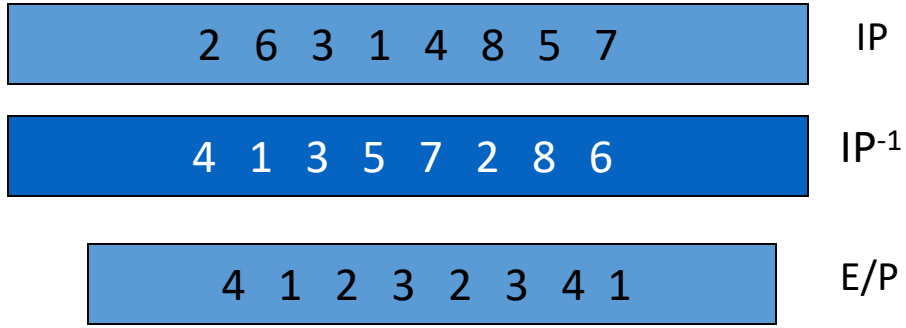
0 0 1 0 0 0 0 0 1 1 $LS^{-2}$ $LS^{-2}$

0 1 0 0 0 0 1 1 $= K_2$ P8

# Simplified DES Encryption:

ciphertext = $IP^{-1}(f_{k2}(SW(f_{k1}(IP(plaintext)))))$



# S0 and S1 boxes:

S0 =

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | 0 | 3 | 2 |
| 1 | 3 | 2 | 1 | 0 |
| 2 | 0 | 2 | 1 | 3 |
| 3 | 3 | 1 | 3 | 2 |

S1 =

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 2 | 0 | 1 | 3 |
| 2 | 3 | 0 | 1 | 0 |
| 3 | 2 | 1 | 0 | 3 |

| 2 | 4 | 3 | 1 |
|---|---|---|---|

P4

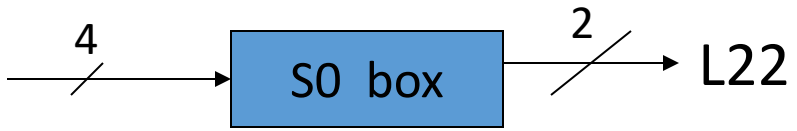
# The function $f_k$ :

Permutation IP is applied to the 8-bit plaintext to generate L1 and R1, the left and the right halves.

INPUT to $f_{k:}$

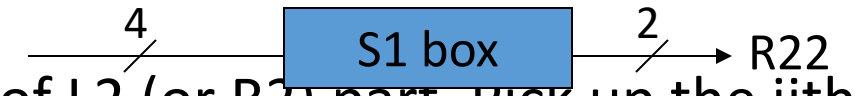
- 4-bit left half (L1) and
- 4-bit Right half (R1) of a data string.

- Step1: E/P: Expansion/Permutation on R1 to produce an 8 bit data string called EPR.
- Step2: XOR of EPR with key $K_1$ for $f_{k1}$ to produce the left half (L2) and right half (R2).
- Step3 (a): L2



# The function $f_k$ (continued):

- Step3(b): R2




Given the 4 bits of L2 (or R2) part. Pick up the ijth element of S0 (or of S1), where

i = 1st and 4th bits; j = 2nd and 3rd bits.

Then convert this element to a 2-bit binary number.

- AUTOKEYING (also called autoclaving): In step 3, the selection of the element in the S-box depends on both data & key. This feature is called autokeying.

# The function $f_k$ (continued)

Step4: (L22 : R22) goes through a permutation P4 to produce a 4-bit R3.

P4

| 2 | 4 | 3 | 1 |
|---|---|---|---|

- Step5: L3 = L1 $\oplus$ R3
- Step6: L3 : R1 is then the input to SW .
- The second instance of $f_k$ is similar to the first, except that the key $K_2$ is used.

# Example: SDES Encryption

- Example:

Plaintext = 1 0 1 1 1 1 0 1

0 1 1 1 1 1 1 0

L1 = 0 1 1 1

R1 = 1 1 1 0

EPR = 0 1 1 1 1 1 0 1

EPR $\oplus$ K1 = 1 1 0 1 1 0 0 1

Row : first and fourth bit

Column : 2nd and 3rd bit

# Example: SDES Encryption (continued)

For S0:

L2 = 1 1 0 1

Therefore Row = 3 Column = 2

L22 = 3 $\Rightarrow$ 11

For S1:

R2 = 1 0 0 1

Row = 3 Column = 0

R22 = 2 $\Rightarrow$ 1 0

L22 : R22 1 1 1 0

R3 = 1 0 1 1

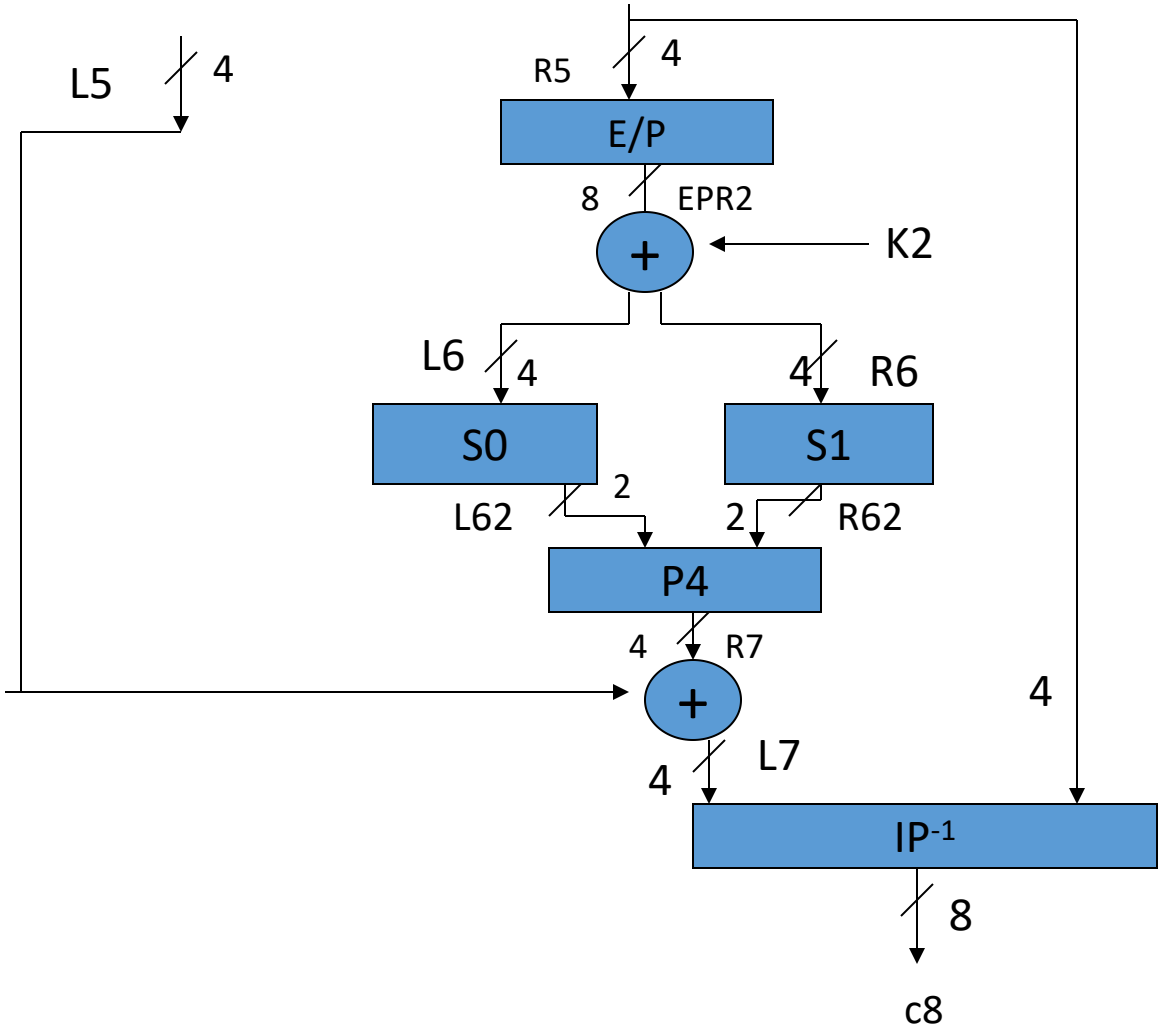
# Example: SDES Encryption (continued)

$$L3 = R3 \oplus L1$$

$$= 1\ 1\ 0\ 0$$

$$L5 = 1\ 1\ 1\ 0$$

$$R5 = 1\ 1\ 0\ 0$$


L5
4
R5
4
E/P
8
EPR2
+
K2
L6
4
4
R6
S0
S1
2
L62
2
R62
P4
4
R7
+
4
4
L7
IP-1
8
c8

# Example: SDES Encryption: $f_{k2}$

- EPR2 = 0 1 1 0  1 0 0 1

  EPR2 $\oplus$ K2  = 0 0 1 0 1 0 1 0

  L6  = 0 0 1 0

  R6  = 1 0 1 0

For S0:

  Row = 0  Column = 1

L62 = 0 $\Rightarrow$ 0 0

# Example: SDES Encryption: $f_{k2}$ (continued)

For S1:

Row = 2  Column = 1

R62 = 0 $\Rightarrow$ 0 0

R7 = 0 0 0 0

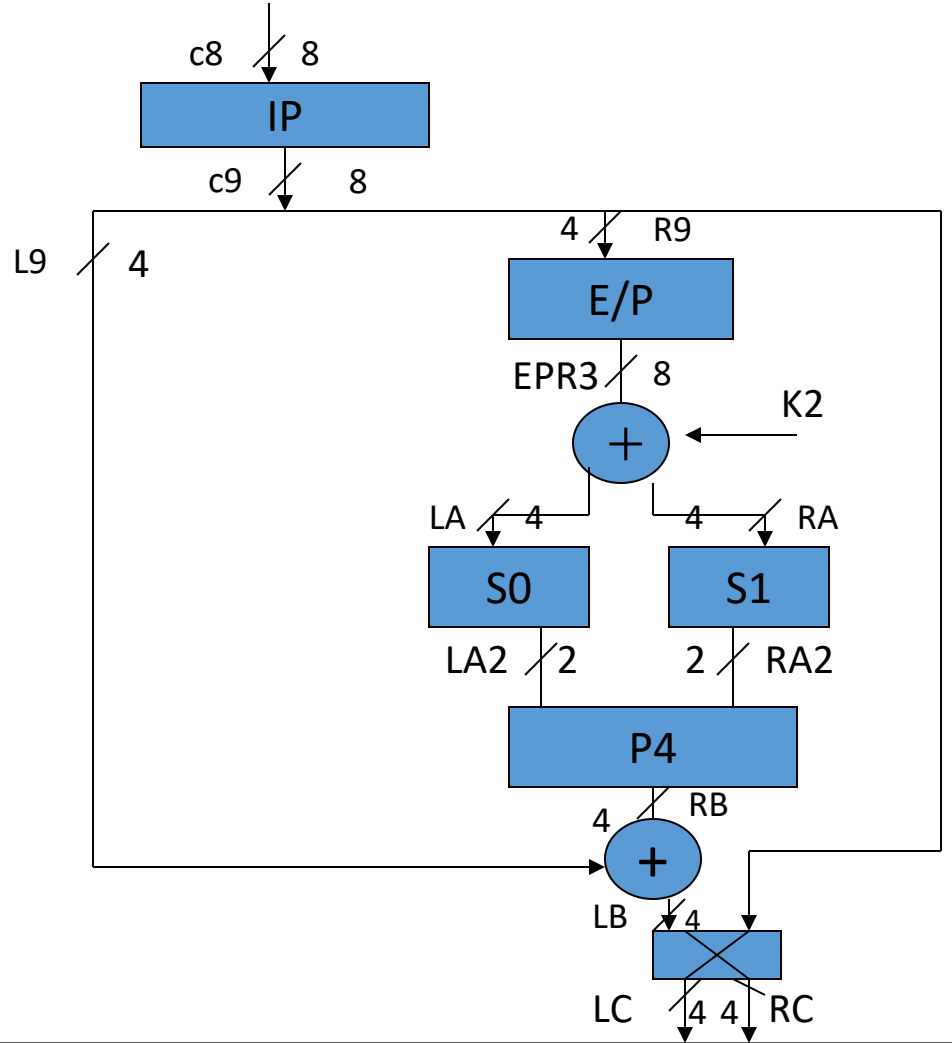
L7 = R7 $\oplus$ L5

= 1 1 1 0

L7 : L5 = 1 1 1 0  1 1 0 0

C8 = 0 1 1 1 0 1 0 1

# SDES Decryption:



# Decryption: Example

C9 = 1 1 1 0  1 1 0 0

EPR3 = 0 1 1 0  1 0 0 1

LA : RA = EPR3 $\oplus$ K2

= 0 0 1 0  1 0 1 0

S0:  Row = 0  Column = 1

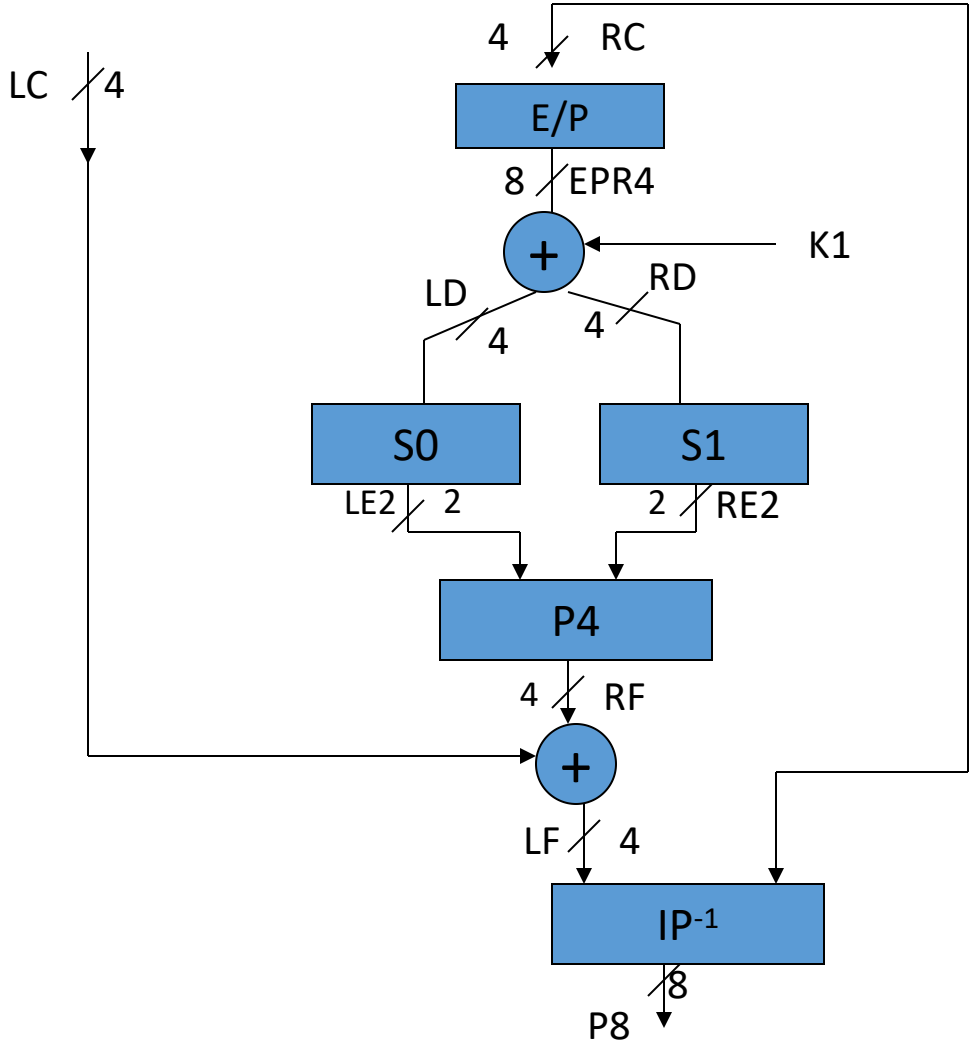
LA2 = 0  => 0 0

S1:  Row = 2  Column = 1

RA2 = 0  => 0 0

R7 = 0000  LB = 1110

LC = 1100  RC = 1110


4
RC
LC
4
E/P
8
EPR4
+
K1
LD
RD
4
4
S0
S1
LE2
2
2
RE2
P4
4
RF
+
LF
4
IP-1
8
P8

# Decryption: Example (continued)

- EPR4 = 0 1 1 1  1 1 0 1

  LD : RD = EPR4 ⊕ K1

  = 1 1 0 1  1 0 0 1

S0:  Row = 3  Column = 2

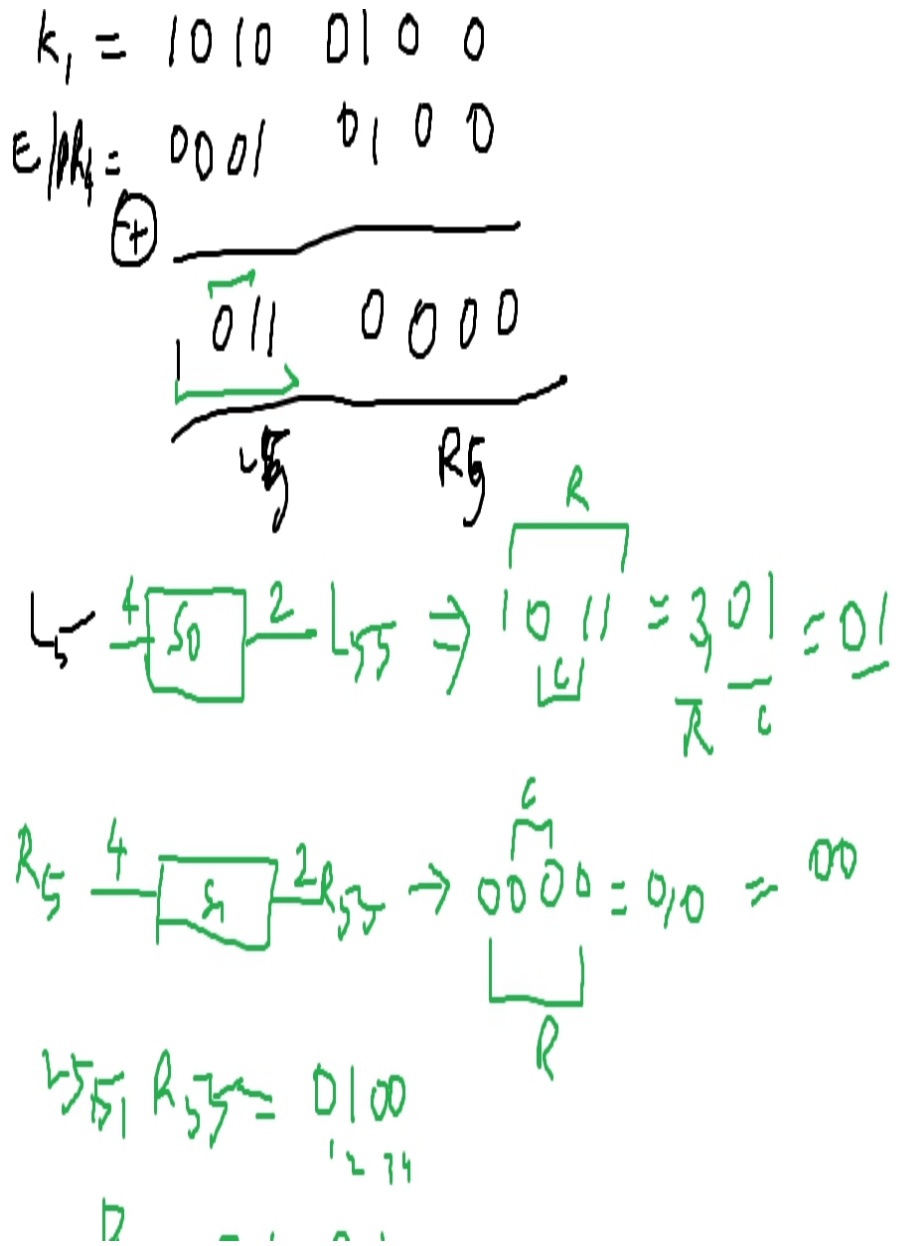
LE2 = 3  => 1 1

S1:  Row = 3  Column = 0

RE2 = 2  => 1 0

LE2 : RE2  = 1 1 1 0

RF = 1 0 1 1



# Decryption: Example (continued)

LF = LC $\oplus$ RF

= 0 1 1 1

LF : RC = 0 1 1 1 1 1 1 0

P8 = 1 0 1 1 1 1 0 1

"As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality."

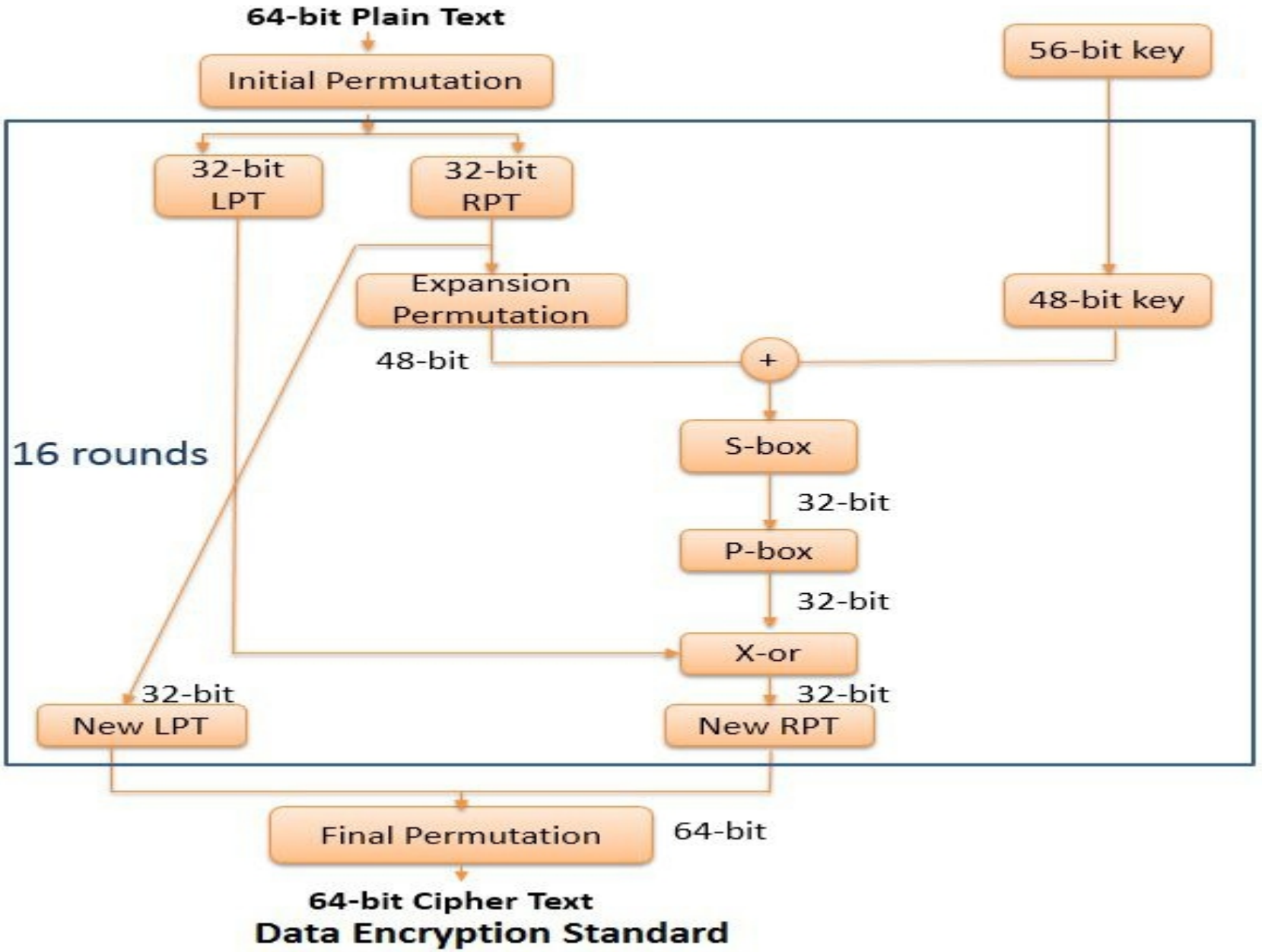
# DES Encryption:

DES: a public standard. But its design criterion has not been published.

**64 bit plaintext goes through**

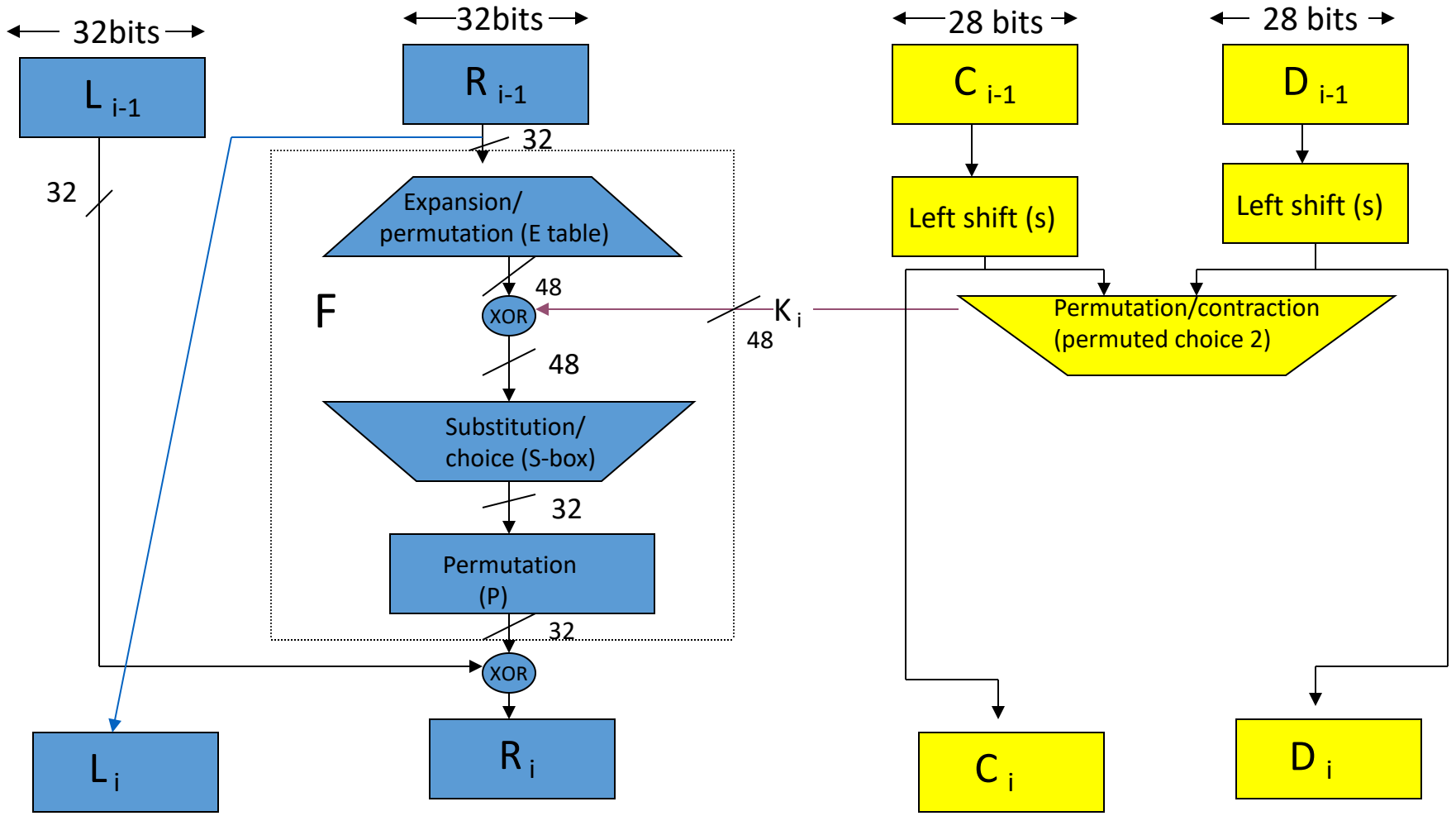
- an Initial Permutation (IP).
- 16 Rounds of a complex function $f_k$ as follows:
  - Round 1 of a complex function $f_k$ with sub key $K_1$ .
  - Round 2 of a complex function $f_k$ with sub key $K_2$.
  - Round 16 of a complex function $f_k$ with sub key $K_{16}$
- At the end of 16 rounds, the Left-half and Right-half are swapped..
- an Inverse Initial Permutation ($IP^{-1}$ )

**to produce 64 bit ciphertext**.




**Data Encryption Standard**

# Fig : single Round of DES Algorithm:



- Each round contains following functions:
- **Expansion Permutation**: Here the 32-bit right portion is expanded to form 48-bit right portion.
- **Xor**: The 48-bit right portion is Xor with 48-bit subkey obtained from the 56-bit key, which results in the 48-bit output.
- **S-box**: The 48-bit output obtained by Xor step is reduced to 32 bit again.
- **P-box**: Here the 32-bit result obtained from S-box is again permuted, which result in 32-bit permuted output.

# DES Round

- x: block of plaintext
- let $x_0 = IP(x) = L_0:R_0$
- 16 rounds with f: cipher function

  $K_i$: sub-key for the ith round

While $i \leq 16$,

$x_i = L_i:R_i$

$L_i = R_{i-1}$

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

# DES Encryption

Recapitulation:

- IP
- 16 rounds with 16 sub-keys
- Swapping
- Inverse Initial Permutation

# Initial Permutation (IP):

- IP and $IP^{-1}$ are defined by *8X8* tables T1 and T2.

Table T1: IP

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

# Inverse Initial Permutation($IP^{-1}$)

## Table T2: $IP^{-1}$

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

# IP and $IP^{-1}$

Input Permutation permutes the position of bits as follows:

Input 1 2 3 4 …… 61 62 63 64

Position

Output40 8 48 16 …… 49 17 57 25

Position

IP(plaintext) = L(0): R(0),

where L(0) and R(0) are the left –half and the right-half of the output after the permutation done through IP.

Inverse Input Permutation reverses it back.

Thus $IP^{-1}( IP( X)) = X$

# IP

Regular in structure, so that hardware is simple

- Even bits allocated to L(0)
- Odd bits allocated to R(0)

IP does not add to security. It makes the function a little complex.

- Example:

IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)

$IP^{-1}$(068dddcd 1d4ccebf) = 974affbf, 86022d1f

*Reference:Lawrie Brown, "Cryptography: lecture 8". available at http://www.cs.adfa.edu.au/archive/teaching/studinfo/ccs3/lectures/less08.html*

# Function

- E/P: to get 48 bits from 32 bits of $R_i$ : each input block of 4 bits *contributes* 2 bits to each output block → Avalanche Effect: A small difference in plaintext *causes* quite different ciphertext
- $E(R_{i-1}) \oplus Ki$
- Eight *4x16* S-boxes for converting 48 bits to 32 bits output: Non-linear; provide major part of the strength of the cipher: Divide 48 bits to 8 units of 6 bits each; the first and the last bit determine the row #; the middle 4 bits determine the column #; The element value is between 0 and 15 → 4 bits
- Straight permutation
- XOR with left half
- Switch the left half and the right half

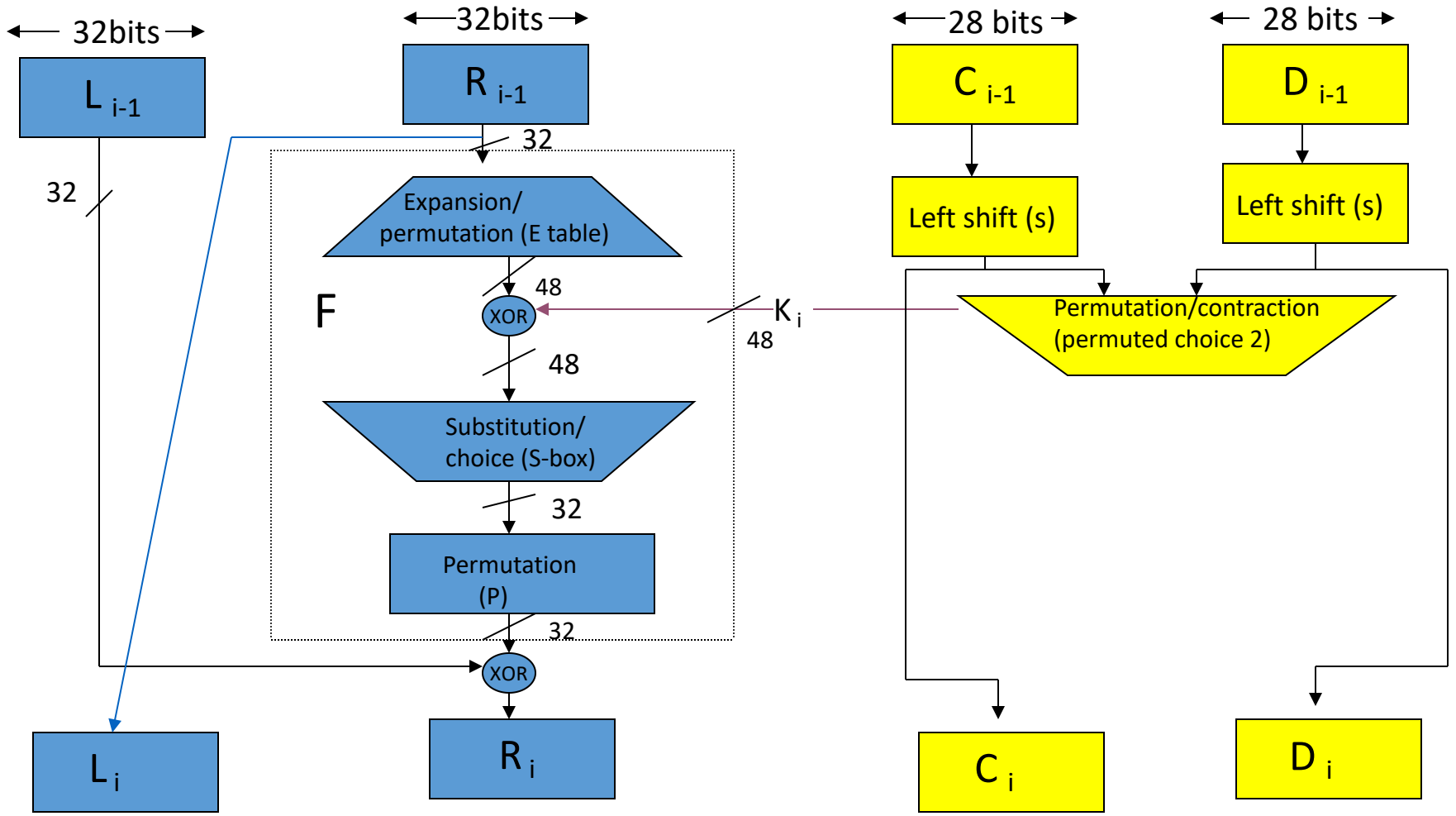
# Each round of encryption:

After IP, 64 bits are divided into left-half (L(0)) and Right-half (R(0)).

- L(0): R(0) is the input to Round 1 of encryption.
  During Round1, L(0):R(0) will be operated by $F_{k1}$ to produce L(1):R(1), where $F_{K1}$ is the function $F_k$ with subkey K1.
- .
- .
- Similarly for Round i, $L_{i-1}$:$R_{i-1}$ would be the input and $L_i$: $R_i$ will be the output.

Figure 2 shows the function $F_{Ki}$ .

# Fig : single Round of DES Algorithm:



# i-th Round

The part in yellow, in the previous slide, shows the sub key generation. After PC1, the circular rotations are independent for the left half and the right-half.

ENCRYPTION: In the i-th round,

$L_i = R_{i-1}$

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

$= L_{i-1} \oplus P(S(E(R_{i-1}) \oplus Ki))$

Where E: expansion from 32 bits to 48

S: Using 8 S-boxes to convert 48 bits to 32 bits – each S box converts 6 bits to 4 bits

P: permutation

# Expansion-Permutation (E/P):

- In figure 2, the E-table generates 48-bit output from 32 bit input by expansion-permutation by using table T6.

Table T6: E/P

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

# Expansion-Permutation (continued)

- Table T6 in fact divides the input of 32 bits into 8 units of 4 bit each.
- Each unit is converted to 6 bits by borrowing the two adjoining bits of each unit of 4 bits.
- Thus …… efgh ijkl mnop ……..

  is converted by Expansion/permutation through table T6 to …… defghi hijklm lmnopq ……

*AUTOKEYING/ AUTOCLAVING: The duplicated outside bits in each group of 6 act as a key to the following S-box, allowing the data (as well as the key) to influence the S-box operation.*

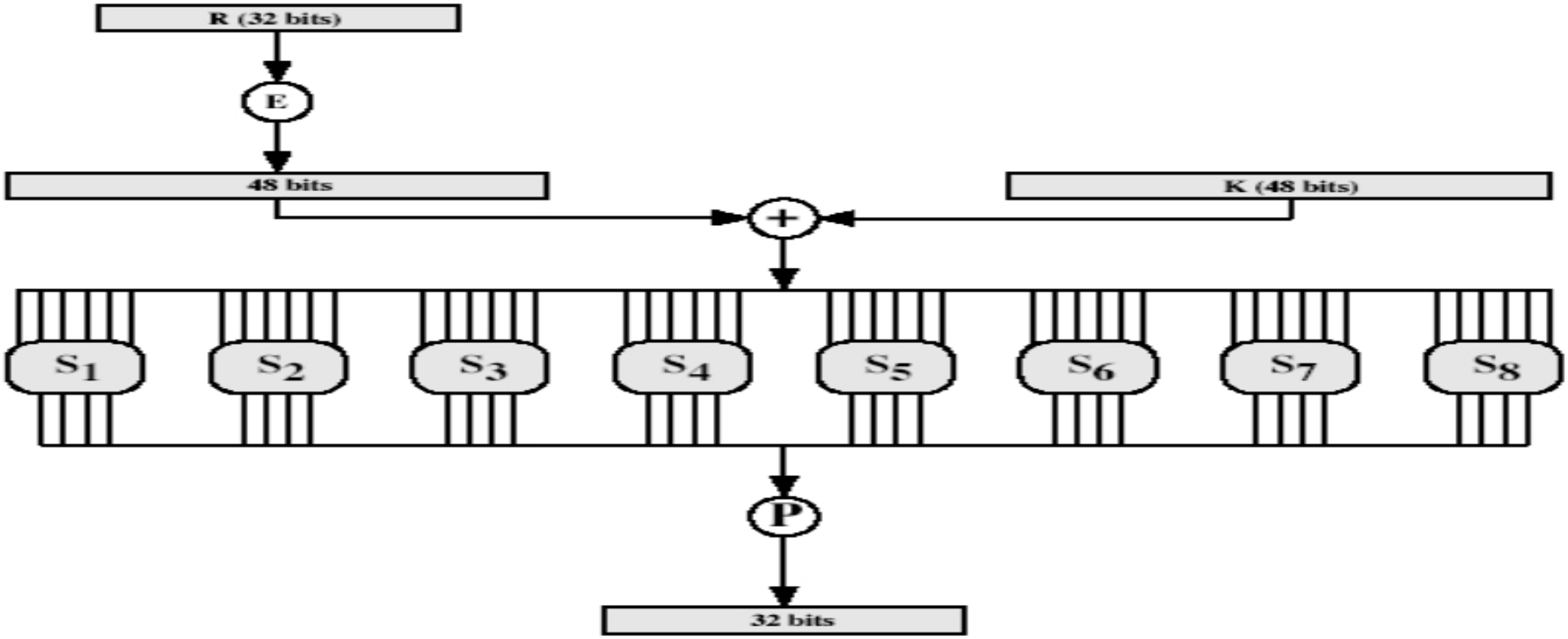
- This results in a 48 bit output from E/P module.

# Output of E/P and S-Boxes

- The output from the E/P module is divided into 8 groups of 6 bits each.
- Using 8 *4X16* S-boxes, each group of 6 bits is reduced to 4 bits as follows:
- For each S box: Row Number = Outermost 2 bits; Column Number = Inner 4 bits.
- Using the row and column number, the S-box yields a decimal number (lying between 0 and 15). Its 4 bit binary equivalent is the output of the S-box.

*Reference: A F Webster & S E Tavares "On the Design of S-boxes", in Advances in Cryptology - Crypto 85, Lecture Notes in Computer Science, No 218, Springer-Verlag, 1985, pp 523-534*

# DES Round Structure



# Definition of DES S-boxes:

- 8 S-boxes are shown in table T-7.

Table T7: S-Boxes

$S_1$

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

$S_2$

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

# Definition of DES S-boxes: S3 and S4

$S_3$

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

$S_4$

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

# Definition of DES S-boxes: S5 and S6

$S_5$

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

$S_6$

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

# Definition of DES S-boxes: S7 and S8

$S_7$

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

$S_8$

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

# Permutation Function (P):

- 8 S-boxes give 32 bit output, which is passed through Permutation (P).
- P is shown in table T-8.

Table T8: P

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

# Example: Evaluation of F:

Given $R_{i-1}$ and Ki: E, S boxes and P

$E(R_{i-1}) = E(004df6fb) = 20\ 00\ 09\ 1b\ 3e\ 2d\ 1f\ 36$

Converts 32 bits to 48 bits. The output is in 8 groups of 6bits – the first digit = 2bits; the second digit = 4 bits

Ki = 38 09 1b 26 2f 3a 27 0f

-- K1 from Slide 44

$E(R_{i-1}) \oplus Ki = 18\ 09\ 12\ 3d\ 11\ 17\ 38\ 39$

Input to S boxes: 48bits divided into 8 groups of 6 bits each.

S(18 09 12 3d 11 17 38 39) = 5fd25e03

P(5fd25e03) = 746fc91a

# Key Schedule Algorithm

- Each sub-key $K_i$ : 48 bits: obtained from a 56 bit key K
- Fixed Permutation: PC1(K) = $C_0$:$D_0$
- A left circular shift (of 1 or 2 bits) on the Left-half ($C_0$ ) and Right-half ($D_0$) *separately* (Output: $C_1$ of 28 bits and $D_1$ of 28 bits)
- 2 bits: for rounds 3-8 and 10-15
- Compression permutation PC2 to get 48 bit key $K_i$ from $C_i$:$D_i$
- Round-dependent left shifts → different parts of initial key create each sub-key

# Sub Key Generation

The input key: 56 bits

Hardware Design: the 8, 16, 24, 32, 40, 48, 56 and 64th bit is always the odd parity bit. → 64 bit key

Software design: the key is stated in ASCII code. Each character of 8 bits, with the first bit being zero plus 7 bits of code. (!)

Since DES was designed with the viewpoint of hardware implementation, the conversion to 56 bits is done by neglecting every $8^{th}$ bit.

PC1 converts to 56 bits and permutes.

# Key Schedule

- K: 64 bit key
- $C_0$: $D_0$ =PC1(K) , 56 bit key
- 16 steps for i = 1-15: A left circular

shift (of 1 or 2 bits) on the Left-half ($C_{i-1}$) and Right-half ($D_{i-1}$) *separately* (Output: $C_i$ of 28 bits and $D_i$ of 28 bits)

- 16 Subkeys for i = 1-15: $K_i$ = PC2($C_i$ : $D_i$ )

  of 48 bits each

# Input Key

odd parity bit: 8, 16, 24, 32, 40, 48, 56, 64

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

# Permuted Choice One (PC –1):

- Tables *8x7* T3 show PC-1.
- Table *6x8* T4 (slide 39) shows PC-2. Table T5 (slide 40) gives the number of shifts

Table T3: PC-1

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

# PC1: Obtaining $C_0$ and $D_0$

PC1 generates $C_0$ and $D_0$, the left and the right halves respectively.

$C_0$ Read the first column of the input 64-bit key from bottom up. Write it row-wise from left to right. Repeat for the second, the third and the lower-half of the fourth column respectively.

$D_0$ Read the seventh column of the input 64-bit key from bottom up. Write it row-wise from left to right. Repeat for the sixth, the fifth and the upper-half of the fourth column respectively.

Probably the conversion to the two halves was done due to the limitation of the hardware of seventies.

# Sub Key Generation: continued

Thus DES has a 56 bit key K consisting of $C_0$ and $D_0$.

All the sub keys K1 to K16 are of 48 bits.

To generate these keys, K goes through

- A Permuted Choice (PC-1) (output $C_0$ of 28 bits and $D_0$ of 28 bits).
  - A left circular shift (of 1 or 2 bits) on the Left-half ($C_0$ ) and Right-half ($D_0$) *separately* (Output: $C_1$ of 28 bits and $D_1$ of 28 bits)

    followed by a Permuted Choice (PC-2) which permutes as well as 'contracts' to produce a sub-key $K_1$ of 48 bits.

# Sub Key Generation (continued)

- A left circular shift (of 1 or 2 bits) on the Left-half ($C_1$ ) and Right-half ($D_1$) *separately* (Output: $C_2$ of 28 bits and $D_2$ of 28 bits)

  followed by a Permuted Choice (PC-2) which permutes as well as 'contracts' to produce a sub-key $K_2$ of 48 bits.
- .
- .
- .
- A left circular shift (of 1 or 2 bits) on the Left-half ($C_{15}$ ) and Right-half ($D_{15}$) *separately* (Output: $C_{16}$ of 28 bits and $D_{16}$ of 28 bits)

  followed by a Permuted Choice (PC-2) which permutes as well as 'contracts' to produce a sub-key $K_{16}$ of 48 bits.

# Key Schedule

- KA = PC1(K)
- KB1 = LS-j(KA);

LS-j is left circular shift by j bits, on the two halves of the 56 bits separately. j is given by Table 5.

KB2 = LS-j(KB1)

KB3 = LS-j(KB2)

.

$KB_i$ = LS-j($Kb_{i-1}$)

.

KB16 = LS-j(KB15)

- Ki = PC2($KB_i$)

# Permuted Choice Two (PC-2):

**Table T-4: PC-2:** The upper 3 rows (24bits) refer to the left half Ci. (It affects S-boxes 1 to 4.) Similarly the remaining 24 bits refer to Di (and affect S-boxes 5 to 8).

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

Schedule of left shifts:

- The number of circular left shift in each round is given in Table T5.
- 2 bits: for rounds 3-8 and 10-15:
- 1 bit: for rounds 1, 2, 9, 16 only. Total = 28

Table T-5 The number of circular Left Shifts

| Round number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bits rotated | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Schedule of left shifts:

- Co→C1 with 1 shift
- C1→C2 with 1 shift
- C2→C3 with 2 shifts
- .
- .
- C8→C9 with 2 shifts
- .
- .
- C15→C16 with 1 shift

Di follows the same shifts as Ci.

# Left Shifts in Key schedule

- The effect of left-shifts:
  - First, any bit would affect a different S-box in successive rounds.
  - Secondly if a  bit is not used in one sub-key, it would be in the next.
  - Thirdly after 16 rounds it comes back to the original value so that computing the sub-keys for decryption becomes easy.

# Example Trace of DES Key Schedule:

*Reference:Lawrie Brown, "Cryptography: lecture 8". available at http://www.cs.adfa.edu.au/archive/teaching/studinfo/ccs3/lectures/less08.html*

- keyinit(5b5a5767, 6a56676e)
- PC1(Key) C=00ffd82, D=ffec937

Key = PC2(C,D): Given in 16 HEX digits, in 8 pairs. The first digit is of 2bits (a value lying between 0 and 3), the second digit is of 4bits.

- KeyRnd01 C=01ffb04, D=ffd926f,
  PC2(C,D)=(38 09 1b 26 2f 3a 27 0f)
- KeyRnd02 C=03ff608, D=ffb24df,
  PC2(C,D)=(28 09 19 32 1d 32 1f 2f)
- KeyRnd03 C=0ffd820, D=fec937f,
  PC2(C,D)=(39 05 29 32 3f 2b 27 0b)
- KeyRnd04 C=3ff6080, D=fb24dff,
  PC2(C,D)=(29 2f 0d 10 19 2f 1d 3f)
- KeyRnd05 C=ffd8200, D=ec937ff,
  PC2(C,D)=(03 25 1d 13 1f 3b 37 2a)

# Example Trace of DES Key Schedule:

(continued-2)

- KeyRnd06 C=ff60803, D=b24dfff,
  PC2(C,D)=(1b 35 05 19 3b 0d 35 3b)
- KeyRnd07 C=fd8200f, D=c937ffe,
  PC2(C,D)=(03 3c 07 09 13 3f 39 3e)
- KeyRnd08 C=f60803f, D=24dfffb,
  PC2(C,D)=(06 34 26 1b 3f 1d 37 38)
- KeyRnd09 C=ec1007f, D=49bfff6,
  PC2(C,D)=(07 34 2a 09 37 3f 38 3c)
- KeyRnd10 C=b0401ff, D=26fffd9,
  PC2(C,D)=(06 33 26 0c 3e 15 3f 38)
- KeyRnd11 C=c1007fe, D=9bfff64,
  PC2(C,D)=(06 02 33 0d 26 1f 28 3f)

# Example Trace of DES Key Schedule:

(continued-3)

- KeyRnd12 C=0401ffb, D=6fffd92,
  PC2(C,D)=(14 16 30 2c 3d 37 3a 34)
- KeyRnd13 C=1007fec, D=bfff649,
  PC2(C,D)=(30 0a 36 24 2e 12 2f 3f)
- KeyRnd14 C=401ffb0, D=fffd926,
  PC2(C,D)=(34 0a 38 27 2d 3f 2a 17)
- KeyRnd15 C=007fec1, D=fff649b,
  PC2(C,D)=(38 1b 18 22 1d 32 1f 37)
- KeyRnd16 C=00ffd82, D=ffec937,
  PC2(C,D)=(38 0b 08 2e 3d 2f 0e 17)

Note that $C_{16} = C_0$ and $D_{16} = D_0$

# Example:

- Given:Slide 32: plaintext and IP(plaintext) = L0:R0
- $F_{k1}$(R0) = 746fc91a -- Slide 59
- L0=ffb2194d,
- R1 = ($F_{k1}$(R0) $\oplus$ L0) = 8bddd057
- L1 = R0 = 004df6fb

After 16 rounds, $IP^{-1}$ is the last step, for getting the encrypted block.

*Reference:Lawrie Brown, "Cryptography: lecture 8". available at http://www.cs.adfa.edu.au/archive/teaching/studinfo/ccs3/lectures/less08.html*

# DES Decryption:

Decryption uses the same algorithm as encryption except that the application of the sub-keys is reversed.:

- In the first round of decryption, sub-key K16 is used.
- .
- .
- .
- In the $16^{th}$ round of decryption, sub-key K1 is used .

# Decryption Relations

ENCRYPTION: (from slide 49)

$L_i = R_{i-1}$

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

$= L_{i-1} \oplus P(S(\ E(R_{i-1}) \oplus Ki\ ))$

Rewriting: DECRYPTION relations are:

$R_{i-1} = L_i$

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

On substituting the value of $R_{i-1}$ from the first decryption relation,

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

# Decryption Process

- First:  IP on ciphertext: undoes the final $IP^{-1}$ step of encryption
- 16 Rounds: First round with subkey 16 undoes 16th round of encryption

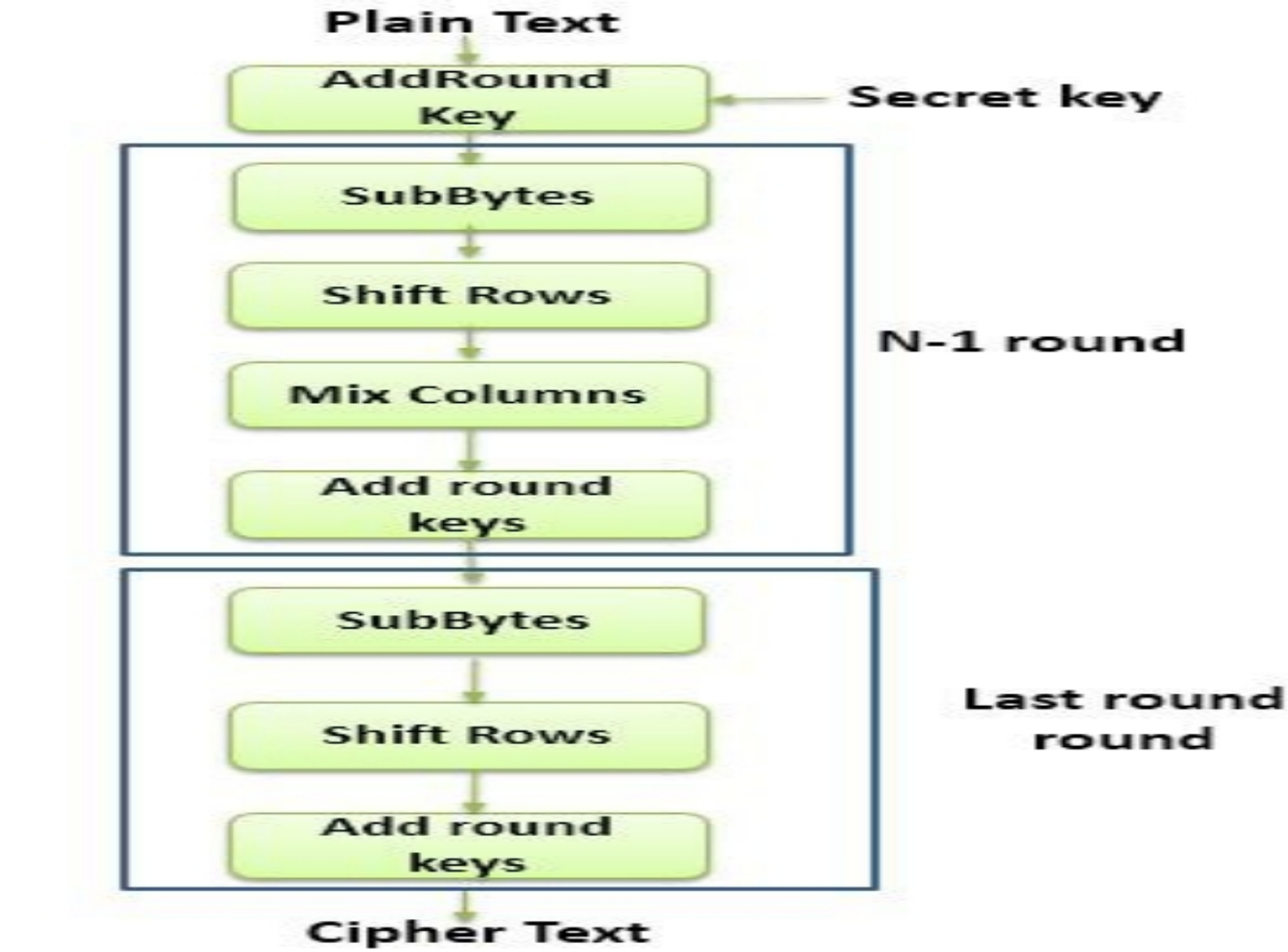
  .

  .

  Sixteenth round with subkey 1 undoes 1st encryption round
- Last: $IP^{-1}$ undoes the initial encryption IP

# Definition of AES (Advanced Encryption Standard)

- Advanced Encryption Standard (AES) is also a **symmetric key block cipher**. AES was published in **2001** by the **National Institute of Standards and Technology**.
- AES was introduced to replace DES as DES uses very small cipher key and the algorithm was quite slower.




Advanced Encryption Standard

# AES algorithm

- AES algorithm takes 128-bit plaintext and 128-bit secret key which together forms a 128-bit block which is depicted as 4 X 4 square matrix. This 4 X 4 square matrix undergoes an initial transformation. This step is followed by the 10 rounds. Among which 9 round contain following stages:
- **Sub-bytes**: It uses S-box by which it performs byte by byte substitution of the entire block (matrix).
- **Shift Rows**: Rows of the matrix are shifted.
- **Mix Columns**: Columns are of the matrix are shuffled from right to left.
- **Add round keys**: Here**,** the Xor of the current block and the expanded key is performed.
- And the last 10th round involves Subbytes, Shift Rows, and Add round keys stages only and provides 16 bytes (128-bit) ciphertext.

# Difference Between DES (Data Encryption Standard) and AES (Advanced Encryption Standard)

| BASIS FOR COMPARISON | DES (DATA ENCRYPTION STANDARD) | AES (ADVANCED ENCRYPTION STANDARD) |
|---|---|---|
| Basic | In DES the data block is divided into two halves. | In AES the entire data block is processed as a single matrix. |
| Principle | DES work on Feistel Cipher structure. | AES works on Substitution and Permutation Principle. |
| Plaintext | Plaintext is of 64 bits | Plaintext can be of 128,192, or 256 bits |
| Key size | DES in comparison to AES has smaller key size. | AES has larger key size as compared to DES. |
| Rounds | 16 rounds | 10 rounds for 128-bit algo<br>12 rounds for 192-bit algo<br>14 rounds for 256-bit algo |
| Rounds Names | Expansion Permutation, Xor, S-box, P-box, Xor and Swap. | Subbytes, Shiftrows, Mix columns, Addroundkeys. |
| Security | DES has a smaller key which is less secure. | AES has large secret key comparatively hence, more secure. |
| Speed | DES is comparatively slower. | AES is faster. |

# Key Differences Between DES and AES

1. The basic difference between DES and AES is that the block in DES is divided into two halves before further processing whereas, in AES entire block is processed to obtain ciphertext.
2. The DES algorithm works on the Feistel Cipher principle, and the AES algorithm works on substitution and permutation principle.
3. The key size of DES is 56 bit which is comparatively smaller than AES which has 128,192, or 256-bit secret key.
4. The rounds in DES include Expansion Permutation, Xor, S-box, P-box, Xor and Swap. On the other hands, rounds in AES include Subbytes, Shiftrows, Mix columns, Addroundkeys.
5. DES is less secure than AES because of the small key size.
6. AES is comparatively faster than DES

# Cryptography and Network Security

Third Edition

by William Stallings

Lecture slides by Lawrie Brown

# Chapter 7 – Confidentiality Using Symmetric Encryption

# Confidentiality using Symmetric Encryption

- traditionally symmetric encryption is used to provide message confidentiality
- consider typical scenario
  - workstations on LANs access other workstations & servers on LAN
  - LANs interconnected using switches/routers
  - with external lines or radio/satellite links
- consider attacks and placement in this scenario
  - snooping from another workstation
  - use dial-in to LAN or server to snoop
  - use external router link to enter & snoop
  - monitor and/or modify traffic one external links

# Confidentiality using Symmetric Encryption

- have two major placement alternatives
- **link encryption**
  - encryption occurs independently on every link
  - implies must decrypt traffic between links
  - requires many devices, but paired keys
- **end-to-end encryption**
  - encryption occurs between original source and final destination
  - need devices at each end with shared keys

# Traffic Analysis

- when using end-to-end encryption must leave headers in clear
  - so network can correctly route information
- hence although contents protected, traffic pattern flows are not
- ideally want both at once
  - end-to-end protects data contents over entire path and provides authentication
  - link protects traffic flows from monitoring

# Placement of Encryption

- can place encryption function at various layers in OSI Reference Model
  - link encryption occurs at layers 1 or 2
  - end-to-end can occur at layers 3, 4, 6, 7
  - as move higher less information is encrypted but it is more secure though more complex with more entities and keys

# Traffic Analysis

- is monitoring of communications flows between parties
  - useful both in military & commercial spheres
  - can also be used to create a covert channel
- link encryption obscures header details
  - but overall traffic volumes in networks and at end-points is still visible
- traffic padding can further obscure flows
  - but at cost of continuous traffic

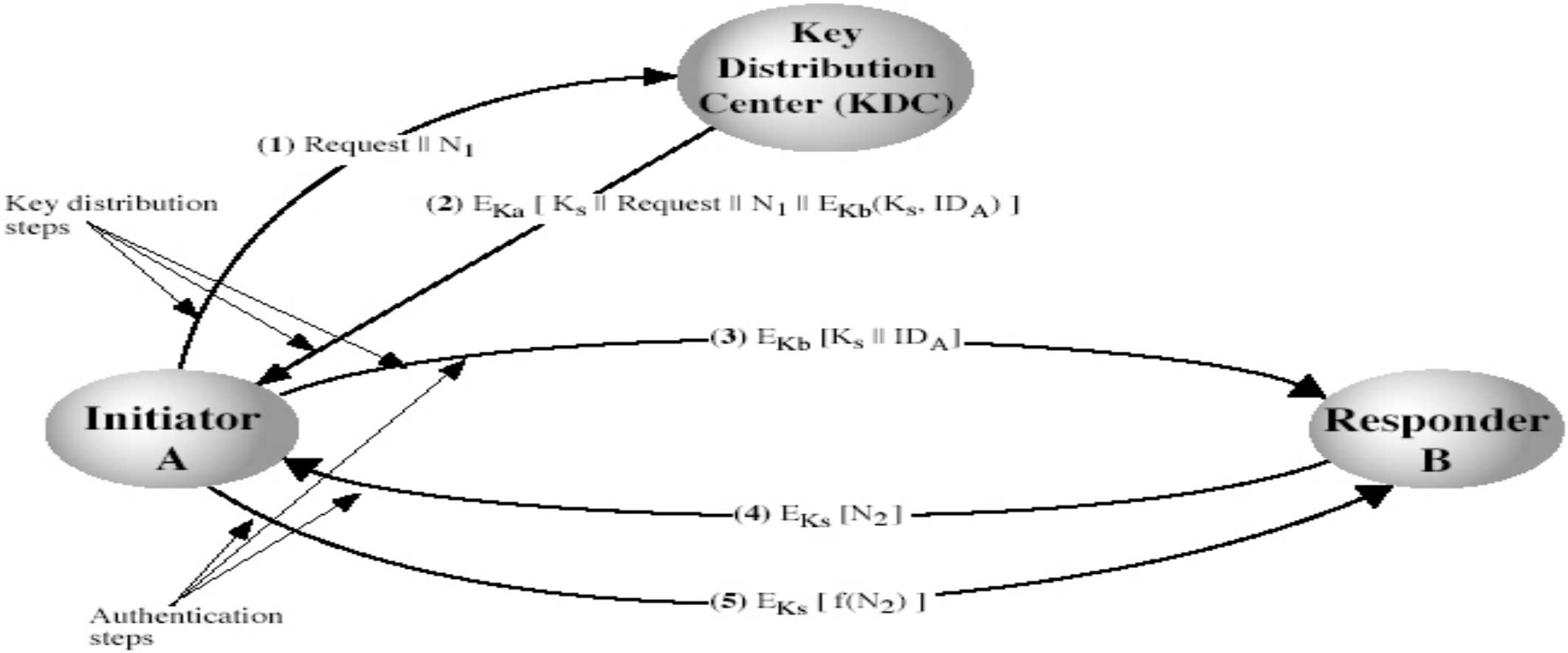
# Key Distribution

- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

# Key Distribution

- given parties A and B have various **key distribution** alternatives:
  1. A can select key and physically deliver to B
  2. third party can select & deliver key to A & B
  3. if A & B have communicated previously can use previous key to encrypt a new key
  4. if A & B have secure communications with a third party C, C can relay key between A & B

# Key Distribution Scenario



# Key Distribution Issues

- hierarchies of KDC’s required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling purposes keys are used for

# Summary

- have considered:
  - use of symmetric encryption to protect confidentiality
  - need for good key distribution
  - use of trusted third party KDC’s

# Chapter 8 – Introduction to Number Theory

# Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163
167 173 179 181 191 193 197 199
```

# Prime Factorisation

- to **factor** a number $n$ is to write it as a product of other numbers: $n = a \times b \times c$
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number $n$ is when its written as a product of primes
  - eg. $91 = 7 \times 13$ ; $3600 = 2^4 \times 3^2 \times 5^2$

$$a = \prod_{p \in P} p^{a_p}$$

# Relatively Prime Numbers & GCD

- two numbers $a$, $b$ are **relatively prime** if have **no common divisors** apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - eg. $300=2^1 \times 3^1 \times 5^2$ $18=2^1 \times 3^2$ hence $GCD(18,300)=2^1 \times 3^1 \times 5^0=6$

# Fermat's Theorem

- $a^{p-1} \bmod p = 1$
  - where $p$ is prime and $\gcd(a,p)=1$
- also known as Fermat's Little Theorem
- useful in public key and primality testing

# Euler Totient Function ø(n)

- when doing arithmetic modulo n
- **complete set of residues** is: `0..n-1`
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the **Euler Totient Function ø(n)**

# Euler Totient Function ø(n)

- to compute ø(n) need to count number of elements to be excluded
- in general need prime factorization, but
  - for p (p prime) ø(p) = p-1
  - for p.q (p,q prime) ø(p.q) = (p-1)(q-1)
- eg.
  - ø(37) = 36
  - ø(21) = (3-1)x(7-1) = 2x6 = 12

# Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{\phi(n)} \bmod N = 1$
  - where $\gcd(a,N)=1$
- eg.
  - $a=3; n=10; \phi(10)=4;$
  - hence $3^4 = 81 = 1 \bmod 10$
  - $a=2; n=11; \phi(11)=10;$
  - hence $2^{10} = 1024 = 1 \bmod 11$

# Primality Testing

- often need to find large prime numbers
- traditionally **sieve** using **trial division**
  - ie. divide by all numbers (primes) in turn less than the square root of the number
  - only works for small numbers
- alternatively can use statistical primality tests based on properties of primes
  - for which all primes numbers satisfy property
  - but some composite numbers, called pseudo-primes, also satisfy the property

# Miller Rabin Algorithm

- a test based on Fermat's Theorem
- algorithm is:

TEST (*n*) is:

1. Find integers *k*, *q*, *k* > 0, *q* odd, so that $(n-1)=2^k q$
2. Select a random integer $a$, $1<a<n-1$
3. **if** $a^q \bmod n = 1$ **then** return ("maybe prime");
4. **for** *j* = 0 **to** *k* – 1 **do**
   5. **if** ($a^{2^j q} \bmod n = n-1$)
      **then** return(" maybe prime ")
6. return ("composite")

# Probabilistic Considerations

- if Miller-Rabin returns “composite” the number is definitely not prime
- otherwise is a prime or a pseudo-prime
- chance it detects a pseudo-prime is < ¼
- hence if repeat test with different random a then chance n is prime after t tests is:
  - Pr(n prime after t tests) = $1-4^{-t}$
  - eg. for t=10 this probability is > 0.99999

# Prime Distribution

- prime number theorem states that primes occur roughly every (`ln n`) integers
- since can immediately ignore evens and multiples of 5, in practice only need test `0.4 ln(n)` numbers of size n before locate a prime
  - note this is only the “average” sometimes primes are close together, at other times are quite far apart

# Chinese Remainder Theorem

- used to speed up modulo computations
- working modulo a product of numbers
  - eg. mod $M = m_1 m_2 .. m_k$
- Chinese Remainder theorem lets us work in each moduli $m_i$ separately
- since computational cost is proportional to size, this is faster than working in the full modulus M

# Chinese Remainder Theorem

- can implement CRT in several ways
- to compute (A mod M) can firstly compute all ($a_i$ mod $m_i$) separately and then combine results to get answer using:

$$A \equiv \left( \sum_{i=1}^{k} a_i c_i \right) \bmod M$$

$$c_i = M_i \times \left( M_i^{-1} \bmod m_i \right) \quad \text{for } 1 \le i \le k$$

# Primitive Roots

- from Euler's theorem have $a^{\phi(n)} \bmod n = 1$
- consider $a^m \bmod n = 1$, $GCD(a,n) = 1$
  - must exist for m= ø(n) but may be smaller
  - once powers reach m, cycle will repeat
- if smallest is m= ø(n) then $a$ is called a **primitive root**
- if $p$ is prime, then successive powers of $a$ "generate" the group $\bmod p$
- these are useful but relatively hard to find

# Discrete Logarithms or Indices

- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find x where $a^x = b \bmod p$
- written as $x=\log_a b \bmod p$ or $x=\text{ind}_{a,p}(b)$
- if a is a primitive root then always exists, otherwise may not
  - $x = \log_3 4 \bmod 13$ (x st $3^x = 4 \bmod 13$) has no answer
  - $x = \log_2 3 \bmod 13 = 4$ by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem

# Summary

- have considered:
  - prime numbers
  - Fermat's and Euler's Theorems
  - Primality Testing
  - Chinese Remainder Theorem
  - Discrete Logarithms

## Modular Arithmetic

We begin by defining how to perform basic arithmetic modulo n, where n is a positive integer. ddition, subtraction, and multiplication follow naturally from their integer counterAparts, but we have complications with division.

## Euclid's Algorithm

We will need this algorithm to fix our problems with division. It was originally designed to find the greatest common divisor of two numbers.

## Divison

Once armed with Euclid's algorithm, we can easily compute divisions modulo n.

## The Chinese Remainder Theorem

We find we only need to study Zpk where p is a prime, because once we have a result about the prime powers, we can use the Chinese Remainder Theorem to generalize for all n.

## Units

While studying division, we encounter the problem of inversion. Units are numbers with inverses.

## Exponentiation

The behaviour of units when they are exponentiated is difficult to study. Modern cryptography exploits this.

## Order of a Unit

If we start with a unit and keep multiplying it by itself, we wind up with 1 eventually. The order of a unit is the number of steps this takes.

## The Miller-Rabin Test

We discuss a fast way of telling if a given number is prime that works with high probability.

## Generators

Sometimes powering up a unit will generate all the other units.

Cyclic Groups

> We focus only on multiplication and see if we can still say anything interesting.

Quadratic Residues

> Elements of Zn that are perfect squares are called quadratic residues.

**Bonus Material**

# Euclid's Algorithm

Given three integers a,b,c, can you write c in the form

$c=ax+by$

for integers x and y? If so, is there more than one solution? Can you find them all? Before answering this, let us answer a seemingly unrelated question:

How do you find the greatest common divisor (gcd) of two integers a,b?

We denote the greatest common divisor of a and b by gcd(a,b), or sometimes even just (a,b). If (a,b)=1 we say a and b are *coprime*.

The obvious answer is to list all the divisors a and b, and look for the greatest one they have in common. However, this requires a and b to be factorized, and no one knows how to do this efficiently.

A few simple observations lead to a far superior method: Euclid's algorithm, or the Euclidean algorithm. First, if d divides a and d divides b, then d divides their difference, a - b, where a is the larger of the two. But this means we've shrunk the original problem: now we just need to find gcd(a,a−b). We repeat until we reach a trivial case.

Hence we can find gcd(a,b) by doing something that most people learn in primary school: division and remainder. We give an example and leave the proof of the general case to the reader.

Suppose we wish to compute gcd(27,33). First, we divide the bigger one by the smaller one:

$33=1\times27+6$

Thus gcd(33,27)=gcd(27,6). Repeating this trick:

$27=4\times6+3$
and we see gcd(27,6)=gcd(6,3). Lastly,

$6=2\times3+0$
Since 6 is a perfect multiple of 3, gcd(6,3)=3, and we have found that gcd(33,27)=3.

This algorithm does not require factorizing numbers, and is fast. We obtain a crude bound for the number of steps required by observing that if we divide a by b to get $a=bq+r$, and $r>b/2$, then in the next step we get a remainder $r'\leq b/2$. Thus every two steps, the numbers shrink by at least one bit.

**Extended Euclidean Algorithm**

The above equations actually reveal more than the gcd of two numbers. We can use them to find integers m,n such that

$3=33m+27n$

First rearrange all the equations so that the remainders are the subjects:

$6=33-1\times27$

$3=27-4\times6$

Then we start from the last equation, and substitute the next equation into it:

$3=27-4\times(33-1\times27)=(-4)\times33+5\times27)$

And we are done: $m=-4,n=5$.

If there were more equations, we would repeat until we have used them all to find m and n.

Thus in general, given integers a and b, let d=gcd(a,b). Then we can find integer m and n such that

$d=ma+nb$

Using the extended Euclidean algorithm.

**The General Solution**

We can now answer the question posed at the start of this page, that is, given integers a,b,c find all integers x,y such that

$c=xa+yb$.

Let $d=\gcd(a,b)$, and let $b=b'd, a=a'd$. Since $xa+yb$ is a multiple of d for any integers x,y, solutions exist only when d divides c.

So say $c=kd$. Using the extended Euclidean algorithm we can find m,n such that $d=ma+nb$, thus we have a solution $x=km, y=kn$.

Suppose $x',y'$ is another solution. Then

$c=xa+yb=x'a+y'b$

Rearranging,

$(x'-x)a=(y-y')b$

Dividing by d gives:

$(x'-x)a'=(y-y')b'$

The numbers $a'$ and $b'$ are coprime since d is the greatest common divisor, hence $(x'-x)$ is some multiple of $b'$, that is:

$x'-x=tb/d$

for some integer t. Then solving for $(y-y')$ gives

$y'-y=ta/d$

Thus $x'=x+tb/d$ and $y'=y-ta/d$ for some integer t.

But if we replace t with any integer, $x'$ and $y'$ still satisfy $c=x'a+y'b$. Thus there are infinitely many solutions, and they are given by

$x=km+tb/d, y=kn+ta/d$.

for all integers t.

Later, we shall often wish to solve $1=xp+yq$ for coprime integers p and q. In this case, the above becomes

$x=m+tq, y=n+tp$.

# Division

Intuitively, to divide x by y means to find a number z such that y times z is x, but we had trouble adopting this defintion of division because sometimes there is more than one possibility for z modulo n.

We solved this by only defining division when the answer is unique. We stated without proof that when division defined in this way, one can divide by y if and only if $y-1$, the inverse of y exists.

We shall now show why this is the case. We wish to find all z such that $yz=x \pmod n$, which by definition means

$$x=zy+kn$$

for some integer k. But this is precisely the problem we encountered when discussing Euclid's algorithm! Let $d=\gcd(y,n)$.

Suppose $d>1$. Then no solutions exist if x is not a multiple of d. Otherwise the solutions for z,k are

$$z=r+tn/d, k=s-ty/d$$

for some integers r,s (that we get from Euclid's algorithm) and for all integers t. But this means z does not have a unique solution modulo n since $n/d<n$. (Instead z has a unique solution modulo n/d.)

On the other hand, if $d=1$, that is if y and n are coprime, then x is always a multiple of d so solutions exist. Recall we find them by using Euclid's algorithm to find r,s such that

$$1=ry+sn$$

Then the solutions for z,k are given by

$$z=xr+tn, k=zs-ty$$

for all integers t. Thus z has a unique solution modulo n, and division makes sense for this case.

Also, r satisfies $ry=1 \pmod n$ so in fact $y-1=r$. Thus our claims are correct: we can divide x by y if and only if y has an inverse.

We can also see that y has an inverse if and only if gcd(y,n)=1. (Actually, we have only proved some of these statements in one direction, but the other direction is easy.)

Note: even though we cannot always divide x by y modulo n, sometimes we still need to find all z such that yz=x.

**Computing Inverses**

We previously asked: given $y \in Z_n$, does $y^{-1}$ exist, and if so, what is it?

Our answer before was that since $Z_n$ is finite, we can try every possibility. But if n is large, say a 256-bit number, this cannot be done even if we use the fastest computers available today.

A better way is to use what we just proved: $y^{-1}$ exists if and only if gcd(y,n)=1 (which we can check using Euclid's algorithm), and $y^{-1}$ can be computed efficiently using the extended Euclidean algorithm.

**Example**: does $7^{-1} \pmod{19}$ exist, and if so, what is it? Euclid's algorithm gives

$19=2\times7+5$

$7=1\times5+2$

$5=2\times2+1$

Thus an inverse exists since gcd(7,19)=1. To find the inverse we rearrange these equations so that the remainders are the subjects. Then starting from the third equation, and substituting in the second one gives

$1=5-2\times2=5-2\times(7-1\times5)=(-2)\times7+3\times5$

Now substituting in the first equation gives

$1=(-2)\times7+3\times(19-2\times7)=(-8)\times7+3\times19$

from which we see that $7^{-1}=-8=11 \pmod{19}$.

# The Chinese Remainder Theorem

Suppose we wish to solve

$x=2 \pmod{5}$

$x=3 \pmod 7$

for x. If we have a solution y, then y+35 is also a solution. So we only need to look for solutions modulo 35. By brute force, we find the only solution is $x=17 \pmod{35}$.

For any system of equations like this, the Chinese Remainder Theorem tells us there is always a unique solution up to a certain modulus, and describes how to find the solution efficiently.

**Theorem**: Let p,q be coprime. Then the system of equations

$x=a \pmod p$

$x=b \pmod q$

has a unique solution for x modulo pq.

The reverse direction is trivial: given $x \in Z_{pq}$, we can reduce x modulo p and x modulo q to obtain two equations of the above form.

**Proof**: Let $p1=p-1 \pmod q$ and $q1=q-1 \pmod p$. These must exist since p,q are coprime. Then we claim that if y is an integer such that

$y=aqq1+bpp1 \pmod{pq}$

then y satisfies both equations:

Modulo p, we have $y=aqq1=a \pmod p$ since $qq1=1 \pmod p$. Similarly $y=b \pmod q$. Thus y is a solution for x.

It remains to show no other solutions exist modulo pq.
If $z=a \pmod p$ then z−y is a multiple of p. If $z=b \pmod q$ as well, then z−y is also a multiple of q. Since p and q are coprime, this implies z−y is a multiple of pq, hence $z=y \pmod{pq}$. ■

This theorem implies we can represent an element of $Z_{pq}$ by one element of $Z_p$ and one element of $Z_q$, and vice versa. In other words, we have a bijection between $Z_{pq}$ and $Z_p \times Z_q$.

**Examples**: We can write $17 \in Z_{35}$ as $(2,3) \in Z_5 \times Z_7$. We can write $1 \in Z_{pq}$ as $(1,1) \in Z_p \times Z_q$.

In fact, this correspondence goes further than a simple relabelling. Suppose $x,y \in Z_{pq}$ correspond to $(a,b),(c,d) \in Z_p \times Z_q$ respectively. Then a little thought shows $x+y$ corresponds to $(a+c,b+d)$, and similarly $xy$ corresponds to $(ac,bd)$.

A practical application: if we have many computations to perform on $x \in Z_{pq}$ (e.g. RSA signing and decryption), we can convert $x$ to $(a,b) \in Z_p \times Z_q$ and do all the computations on $a$ and $b$ instead before converting back. This is often cheaper because for many algorithms, doubling the size of the input more than doubles the running time.

**Example**: To compute $17 \times 17 \pmod{35}$, we can compute $(2 \times 2, 3 \times 3) = (4,2)$ in $Z_5 \times Z_7$, and then apply the Chinese Remainder Theorem to find that $(4,2)$ is $9 \pmod{35}$.

Let us restate the Chinese Remainder Theorem in the form it is usually presented.

**For Several Equations**

**Theorem**: Let $m_1,...,m_n$ be pairwise coprime (that is $\gcd(m_i,m_j)=1$ whenever $i \neq j$). Then the system of $n$ equations

$$x = a_1 \pmod{m_1}$$

...

$$x = a_n \pmod{m_n}$$

has a unique solution for $x$ modulo $M$ where $M = m_1 ... m_n$.

**Proof**: This is an easy induction from the previous form of the theorem, or we can write down the solution directly.

Define $b_i = M/m_i$ (the product of all the moduli except for $m_i$) and $b_i' = b_i^{-1} \pmod{m_i}$. Then by a similar argument to before,

$$x = \sum_{i=1}^{n} a_i b_i b_i' \pmod{M}$$

is the unique solution. ■

**Prime Powers First**

An important consequence of the theorem is that when studying modular arithmetic in general, we can first study modular arithmetic a prime power and

then appeal to the Chinese Remainder Theorem to generalize any results. For any integer n, we factorize n into primes n=p1k1...pmkm and then use the Chinese Remainder Theorem to get

Zn=Zp1k1×...×Zpmkm

To prove statements in Zpk, one starts from Zp, and inductively works up to Zpk. Thus the most important case to study is Zp.

# Lecture 10: Public Key Cryptography

Lecturer: Kurt Mehlhorn & He Sun

---

Today's lecture is about the Public Key Cryptography. We first discuss general ideas in designing cryptography protocols. Typically, the encryption scheme is a pair of algorithms, encryption algorithm and decryption algorithm. When sending a message, the sender fist uses the encryption algorithm to encode the message, and send the encoded messages, called *ciphertext*, over the channel. Upon receiving a ciphertext, the receiver applies the decryption algorithm to decode the message, and receive the original message.

Some possible approaches in designing cryptographic protocols:

- Use a dictionary as the key
- Use a random sequence as the key
- Use a pseudorandom generator and a random seed as the key

## 1 Modular Arithmetic

**Definition 1.** *The number $a$ is equivalent (congruent) to the number $b$ modulo $n$, expressed by $a \equiv b \pmod{n}$, if $a$ differs from $b$ by an exact multiple of $n$.*

**Lemma 2.** *The following statements hold:*

- *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$.*
- *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$.*

**Example.** $321 \times 741 \equiv 1 \times 1 \equiv 1 \pmod{5}$.

**Example.** $715^{10000} \equiv 1 \pmod{7}$.

**Example.** $321^3 \equiv 6^3 \pmod{7} = 36 \times 6 \pmod{7} \equiv 6 \pmod{7}$.

**Example.** $320^{984} \equiv 1 \pmod{7}$

Let us look at the solutions of the example above. We first write down the binary expression of 984, i.e.

$$\begin{aligned} 984 &= 512 + 256 + 128 + 64 + 16 + 8 \\ &= 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 2^3. \end{aligned}$$

Note that $320^9 84 \equiv 5^{984} \pmod{7}$. Moreover, we have the following:

- $5^2 = 25 \equiv 4 \pmod{7}$
- $5^4 = 4 \times 4 \pmod{7} \equiv 2 \pmod{7}$

- $5^8 = 2 \times 2 \pmod 7 \equiv 4 \pmod 7$
- $5^{16} = 4 \times 4 \pmod 7 = 2 \pmod 7$
- $5^{32} \equiv 4 \pmod 7$
- $5^{64} \equiv 2 \pmod 7$
- $5^{128} \equiv 4 \pmod 7$
- $5^{256} \equiv 2 \pmod 7$
- $5^{512} \equiv 4 \pmod 7$

Hence

$$\begin{aligned} 5^{984} &= 5^{512+256+128+64+16+8} \pmod 7 \\ &\equiv 4 \times 2 \times 4 \times 2 \times 2 \times 2 \times 4 \pmod 7 \\ &\equiv 1 \pmod 7 \end{aligned}$$

# 2 Fermat's Little Theorem

We call that $n$ is divisible by $m$ if $n = km$.

**Theorem 3** (Fermat's Little Theorem)**.** *If $p$ is a prime number, then $a^p \equiv a \pmod p$ for all $a$.*

An alternative formulation of the Fermat's Little Theorem is as follows: If $p$ is a prime number and $a$ is any integer not divisible by $p$, then $a^{p-1} \equiv 1 \pmod p$.

**Definition 4.** *If $ab \equiv 1 \pmod m$, then $b$ is called the multiplicative inverse of $a$ modulo $m$.*

# 3 The Euclidean Algorithm

Given two integers $r_0$ and $r_1$, the Euclidean Algorithm finds the greatest common divisor of $r_0$ and $r_1$, denoted by $\gcd(r_0, r_1)$.

Before present the algorithm, we first look at the following lemma.

**Lemma 5.** $\gcd(r_0, r_1) = \gcd(r_1, r_0 \mod r_1)$

**Theorem 6** (The Euclidean Algorithm)**.** *Given two integers $0 < b < a$, we make a repeated application of the division algorithm to obtain a series of division equations, which eventually terminate with a zero remainder:*

$$\begin{aligned} a &= bq_1 + r_1, 0 < r_1 < b \\ b &= r_1 q_2 + r_2, 0 < r_2 < r_1 \\ &\cdots \\ r_{j-2} &= r_{j-1} q_j + r_j, 0 < r_j < r_{j-1} \\ r_{j-1} &= r_j q_{j+1} \end{aligned}$$

*The greatest common divisor $\gcd(a, b)$ of $a$ and $b$ is $r_j$, the last nonzero remainder in the division process.*

Now we show that the Euclidean Algorithm can be used to compute a multiplicative inverse.

**Definition 7.** *If* $ab \equiv 1 \pmod{p}$*, then* $b$ *is called the multiplicative inverse of* $a$ *module* $p$*.*

**Theorem 8** (Multiplicative Inverse Algorithm)**.** *Given two integers* $0 < b < a$*, consider the Euclidean Algorithm equations which yield* $\gcd(a, b) = r_j$*. Rewrite all of these equations except the last one, by solving for the remainders:*

$$\begin{aligned} r_1 &= a - bq_1 \\ r_2 &= b - r_1 q_2 \\ r_3 &= r_1 - r_2 q_3 \\ &\cdots \\ r_{j-1} &= r_{j-3} - r_{j-2} q_{j-1} \\ r_j &= r_{j-2} - r_{j-1} q_j \end{aligned}$$

*Then, in the last of these equations,* $r_j = r_{j-2} - r_{j-1}q_j$*, replace* $r_{j-1}$ *with its expression in terms of* $r_{j-3}$ *and* $r_{j-2}$ *from the equation immediately above it. Continue this process successively, replacing* $r_{j-2}, r_{j-3}, \cdots$*, until we obtain the final equation*

$$r_j = ax + by,$$

*where* $x$ *and* $y$ *are integers. In the special case that* $\gcd(a, b) = 1$*, the integer equation reads*

$$1 = ax + by.$$

*Therefore we deduce*

$$1 \equiv by \mod a$$

*so that the residue of* $y$ *is the multiplicative inverse of* $b$*, mod* $a$*.*

# 4 The RSA Algorithm

- $p, q$ are two prime numbers
- Let $n = p \cdot q$
- Pick a positive integer $r$ that has no common factor with $(p-1) \cdot (q-1)$
- Find a multiplicative inverse of $r$ modulo $(p-1) \cdot (q-1)$, i.e. we find a number $s$ such that $rs \equiv 1 \pmod{(p-1) \cdot (q-1)}$

**Encryption:** We need to know $n, r$. Assume that the message is $x \leq n$. Let

$$y \triangleq x^r \pmod{n}.$$

The pair of values $n, r$ are **public encryption key**. This information is publicly available, and anyone can compute $y$ if they are given $x$.

**Decryption:** To decrypt, you need to know $s$, the **private decryption key**. With the value of $s$, you simply compute

$$z \triangleq y^s \pmod{n} \equiv x^{rs} \equiv x \pmod{n}.$$

That is, you need to know $s$ to decrypt. Now $s$ is the multiplicative inverse of $r$ modulo $(p-1)(q-1)$. The outsiders know $r$, and if they knew $(p-1)(q-1)$, then it would be easy (with the Euclidean Algorithm) to compute $s$. But they do not know $(p-1)(q-1)$. They know $n$, which is equal to $pq$, but they do not have $n$ factored into $p$ and $q$. To find $(p-1)(q-1)$, they need to know the prime factors $p$ and $q$ of $n$, and factoring large numbers is not easy.