USN

INTERNAL ASSESSMENT TEST – II

Sub:	CRYPTOGRAPHY							Code:	21EC642
Date:	11/ 07 / 2024	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	ECE

Answer any 5 full questions

		Marks	**CO**	**RBT**
1	a) Use the properties of discrete logarithms and solve $3 \cdot 5^x \equiv 6 \pmod{23}$. b) What is Euler's Totient function? Find the value of $\phi(144)$.	[6] [4]	CO3	L3
2	Use the matrix [17 17 5 21 18 21 2 2 19] and encrypt the message PAYMOREMONEY using Hill Cipher technique.	[10]	CO3	L3
3	a) Briefly explain the Vernam Cipher technique. b) State and Prove Fermat's Theorem.	[6] [4]	CO2 CO1	L1 L3
4	Explain the Feistel structure encryption and decryption with the neat diagram.	[10]	CO 2	L1
5	Briefly explain the DES Key generation process. Also explain initial and final permutation boxes used in DES.	[10]	CO 2	L1
6	With the help of neat diagram, explain the AES encryption and decryption process.	[10]	CO 2	L2
7	With suitable examples describe the AES MixColumn transformation.	[10]	CO 2	L1
8	Illustrate the following with necessary diagrams: (i) AddRoundKey and SubBytes in AES. (ii) Single DES encryption.	[5] [5]	CO 2	L2

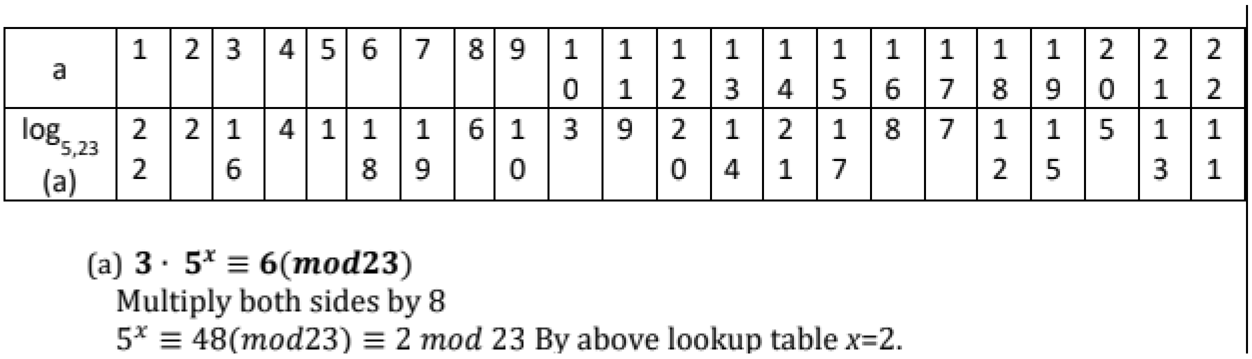
CCI HOD

IAT-2 Solution

1. a) Use the properties of discrete logarithms and solve $3 \cdot 5^x \equiv 6 \pmod{23}$.
   b) What is Euler's Totient function? Find the value of $\phi(144)$.

Solution:

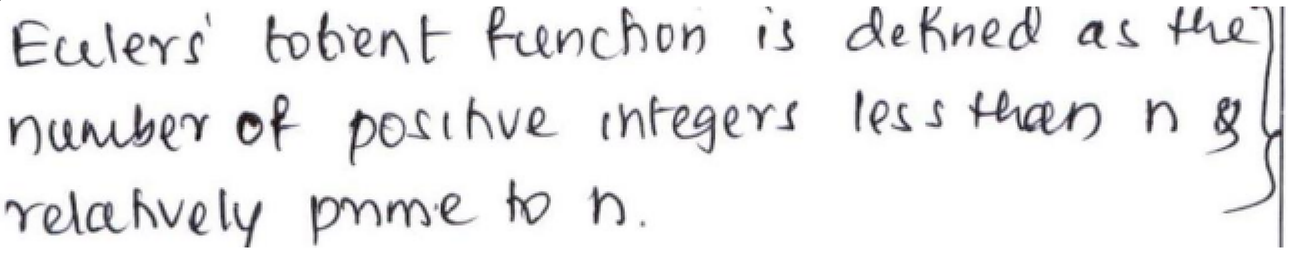
a)



a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
$\log_{5,23}$ (a)	22	2	16	4	1	18	19	6	10	3	9	20	14	21	17	8	7	12	15	5	13	11

(a) $\mathbf{3 \cdot 5^x \equiv 6(mod23)}$
Multiply both sides by 8
$5^x \equiv 48(mod23) \equiv 2 \; mod \; 23$ By above lookup table x=2.

b)



Eulers' totient function is defined as the number of positive integers less than n & relatively prime to n.

2. Use the matrix [17 17 5 21 18 21 2 2 19] and encrypt the message PAYMOREMONEY using Hill Cipher technique.

Plain text: PAY MORE MONEY

Key: $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

Step1: Divide the plain text into block of 3(as here key is a 3X3 matrix)

PAY MOR EMO NEY

P	A	Y	M	O	R	E	M	O	N	E	Y
15	0	24	12	14	17	4	12	14	13	4	24

If it is not possible to make a group then add some filler letters 'X' to complete the group.

Step-2:

$C = KP \; mod \; 26$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \times \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} mod\ 26$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} \times \begin{bmatrix} 15 & 12 & 4 & 13 \\ 0 & 14 & 12 & 4 \\ 24 & 17 & 14 & 24 \end{bmatrix} mod 26$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} 375 & 527 & 342 & 409 \\ 819 & 861 & 594 & 849 \\ 486 & 375 & 298 & 490 \end{bmatrix} mod\ 26 = \begin{bmatrix} 11 & 7 & 4 & 19 \\ 13 & 3 & 22 & 17 \\ 18 & 11 & 12 & 22 \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = C = \begin{bmatrix} L & H & E & T \\ N & D & W & R \\ S & L & M & W \end{bmatrix}$$

Plain Text: PAY MORE MONEY
Cipher Text: LNS HDLE WMTRW

3) a) Briefly explain the Vernam Cipher technique.
b) State and Prove Fermat's Theorem.

Vernam Cipher The ultimate defense against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT&T engineer named Gilbert Vernam in 1918.

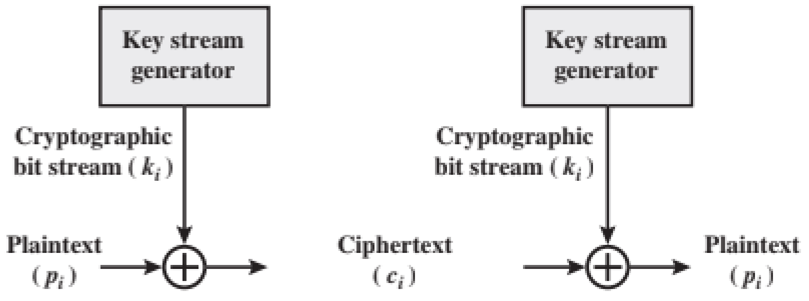


Figure 2.7 Vernam Cipher

His system works on binary data (bits) rather than letters. The system can be expressed succinctly as follows (Figure 2.7):

$$c_i = p_i \oplus k_i$$

where

$p_i =$ ith binary digit of plaintext

$k_i =$ ith binary digit of key

$c_i =$ ith binary digit of ciphertext

$\oplus$ = exclusive-or (XOR) operation

Compare this with Equation (2.3) for the Vigenère cipher.

Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$$p_i = c_i \oplus k_i$$

which compares with Equation (2.4).

The essence of this technique is the means of construction of the key. Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword. Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.

One-Time Pad

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that is as long as the message, so that the key need not be repeated. In addition, the key is to be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. Such a scheme, known as a **one-time pad**, is unbreakable. It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code.

An example should illustrate our point. Suppose that we are using a Vigenère scheme with 27 characters in which the twenty-seventh character is the space character, but with a one-time key that is as long as the message. Consider the ciphertext

```
ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
```

We now show two different decryptions using two different keys:

```
ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:        pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih
plaintext:  mr mustard with the candlestick in the hall

ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:        pftgpmiydgaxgoufhklllmhsqdqogtewbqfgyovuhwt
plaintext:  miss scarlet with the knife in the library
```

Suppose that a cryptanalyst had managed to find these two keys. Two plausible plaintexts are produced. How is the cryptanalyst to decide which is the correct decryption (i.e., which is the correct key)? If the actual key were produced in a truly random fashion, then the cryptanalyst cannot say that one of these two keys is more likely than the other. Thus, there is no way to decide which key is correct and therefore which plaintext is correct.

Fermat's Theorem[4]

Fermat's theorem states the following: If p is prime and a is a positive integer not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p} \qquad \textbf{(8.2)}$$

Proof: Consider the set of positive integers less than p: $\{1, 2, \ldots, p-1\}$ and multiply each element by a, modulo p, to get the set $X = \{a \bmod p, 2a \bmod p, \ldots, (p-1)a \bmod p\}$. None of the elements of X is equal to zero because p does not divide a. Furthermore, no two of the integers in X are equal. To see this, assume that $ja \equiv ka \pmod{p})$, where $1 \le j < k \le p-1$. Because a is relatively prime[5] to p, we can eliminate a from both sides of the equation [see Equation (4.3)] resulting in $j \equiv k \pmod{p}$. This last equality is impossible, because j and k are both positive integers less than p. Therefore, we know that the $(p-1)$ elements of X are all positive integers with no two elements equal. We can conclude the X consists of the set of integers $\{1, 2, \ldots, p-1\}$ in some order. Multiplying the numbers in both sets (p and X) and taking the result mod p yields

$$\begin{aligned} a \times 2a \times \cdots \times (p-1)a &\equiv [(1 \times 2 \times \cdots \times (p-1)] \pmod{p} \\ a^{p-1}(p-1)! &\equiv (p-1)! \pmod{p} \end{aligned}$$

We can cancel the $(p - 1)!$ term because it is relatively prime to p [see Equation (4.5)]. This yields Equation (8.2), which completes the proof.

$$\begin{aligned}
&a = 7, p = 19\\
&7^2 = 49 \equiv 11 \pmod{19}\\
&7^4 \equiv 121 \equiv 7 \pmod{19}\\
&7^8 \equiv 49 \equiv 11 \pmod{19}\\
&7^{16} \equiv 121 \equiv 7 \pmod{19}\\
&a^{p-1} = 7^{18} = 7^{16} \times 7^2 \equiv 7 \times 11 \equiv 1 \pmod{19}
\end{aligned}$$

An alternative form of Fermat's theorem is also useful: If p is prime and a is a positive integer, then

$$a^p \equiv a(\text{mod}\,p) \qquad \textbf{(8.3)}$$

Note that the first form of the theorem [Equation (8.2)] requires that a be relatively prime to p, but this form does not.

$$\begin{array}{ll}
p = 5, a = 3 & a^p = 3^5 = 243 \equiv 3(\text{mod}\,5) = a(\text{mod}\,p)\\
p = 5, a = 10 & a^p = 10^5 = 100000 \equiv 10(\text{mod}\,5) \equiv 0(\text{mod}\,5) = a(\text{mod}\,p)
\end{array}$$

4) Explain the Feistel structure encryption and decryption with the neat diagram.

Feistel Cipher Structure The left-hand side of Figure 3.3 depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K. The plaintext block is divided into two halves, L0 and R0.
The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs Li-1 and Ri-1 derived from the previous round, as well as a subkey Ki derived from the overall K. In general, the subkeys Ki are different from K and from each other. In Figure 3.3, 16 rounds are used, although any number of rounds could be implemented.
All rounds have the same structure. A substitution is performed on the left half of the data. This is done by applying a round function F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey Ki. Another way to express this is to say that F is a function of the right-half block of w bits and a subkey of y bits, which produces an output value of length w bits: F(REi , Ki+1). Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data.6

This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

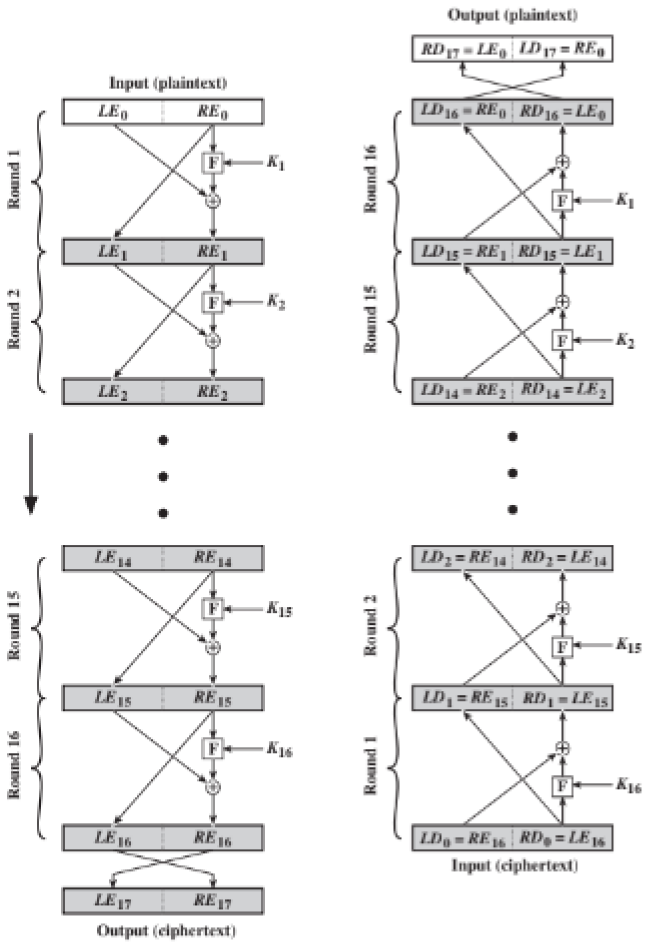


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

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

• Block size: Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.
• Key size: Larger key size means greater security but may decrease encryption/ decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
• Number of rounds: The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security.
A typical size is 16 rounds.

• Subkey generation algorithm: Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

• Round function F: Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:
• Fast software encryption/decryption: In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

- Ease of analysis: Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

FEISTEL DECRYPTION ALGORITHM The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on, until K_1 is used in the last round. This is a nice feature, because it means we need not implement two different algorithms; one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, Figure 3.3 shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm. For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the ith encryption round be $LE_i \| RE_i$ (LE_i concatenated with RE_i). Then the corresponding output of the $(16 - i)$ th decryption round is $RE_i \| LE_i$ or, equivalently, $LD_{16-i} \| RD_{16-i}$.

Let us walk through Figure 3.3 to demonstrate the validity of the preceding assertions. After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is $RE_{16} \| LE_{16}$. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is $RE_{16} \| LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that

$$\begin{aligned} LE_{16} &= RE_{15} \\ RE_{16} &= LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16}) \end{aligned}$$

On the decryption side,

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus \mathrm{F}(RD_0, K_{16}) \\ &= RE_{16} \oplus \mathrm{F}(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})] \oplus \mathrm{F}(RE_{15}, K_{16}) \end{aligned}$$

The XOR has the following properties:

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$D \oplus D = 0$$
$$E \oplus 0 = E$$

Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$. Therefore, the output of the first round of the decryption process is $RE_{15} \| LE_{15}$, which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the ith iteration of the encryption algorithm,

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

Rearranging terms:

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

Thus, we have described the inputs to the ith iteration as a function of the outputs, and these equations confirm the assignments shown in the right-hand side of Figure 3.3.

Finally, we see that the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.

Note that the derivation does not require that F be a reversible function. To see this, take a limiting case in which F produces a constant output (e.g., all ones) regardless of the values of its two arguments. The equations still hold.

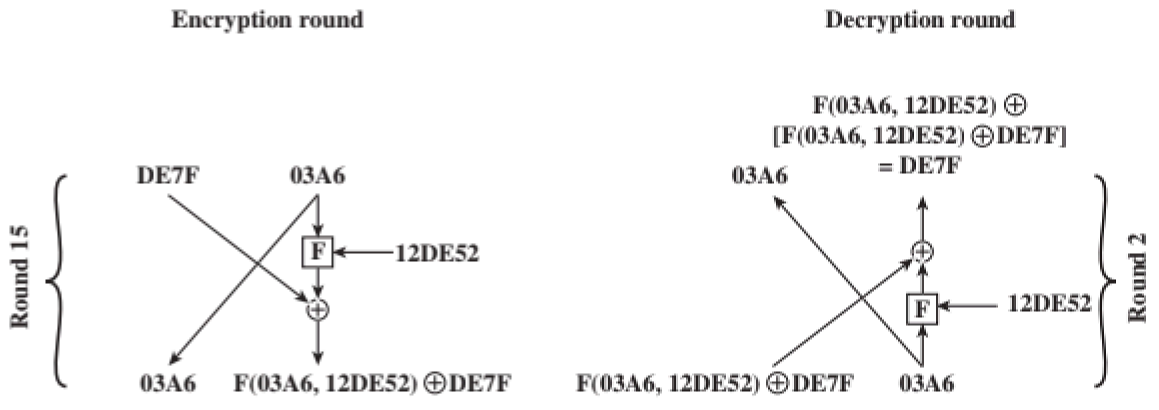


Figure 3.4 Feistel Example

To help clarify the preceding concepts, let us look at a specific example (Figure 3.4 and focus on the fifteenth round of encryption, corresponding to the second round of decryption. Suppose that the blocks at each stage are 32 bits (two 16-bit halves) and that the key size is 24 bits. Suppose that at the end of encryption round fourteen, the value of the intermediate block (in hexadecimal) is DE7F03A6. Then LE_{14} = DE7F and RE_{14} = 03A6. Also assume that the value of K_{15} is 12DE52. After round 15, we have LE_{15} = 03A6 and RE_{15} = F(03A6, 12DE52) $\oplus$ DE7F.

Now let's look at the decryption. We assume that $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$, as shown in Figure 3.3, and we want to demonstrate that $LD_2 = RE_{14}$ and $RD_2 = LE_{14}$. So, we start with LD_1 = F(03A6, 12DE52) $\oplus$ DE7F and RD_1 = 03A6. Then, from Figure 3.3, LD_2 = 03A6 = RE_{14} and RD_2 = F(03A6, 12DE52) $\oplus$ [F(03A6, 12DE52) $\oplus$ DE7F] = DE7F = LE14.

5) Briefly explain the DES Key generation process. Also explain initial and final permutation boxes used in DES.

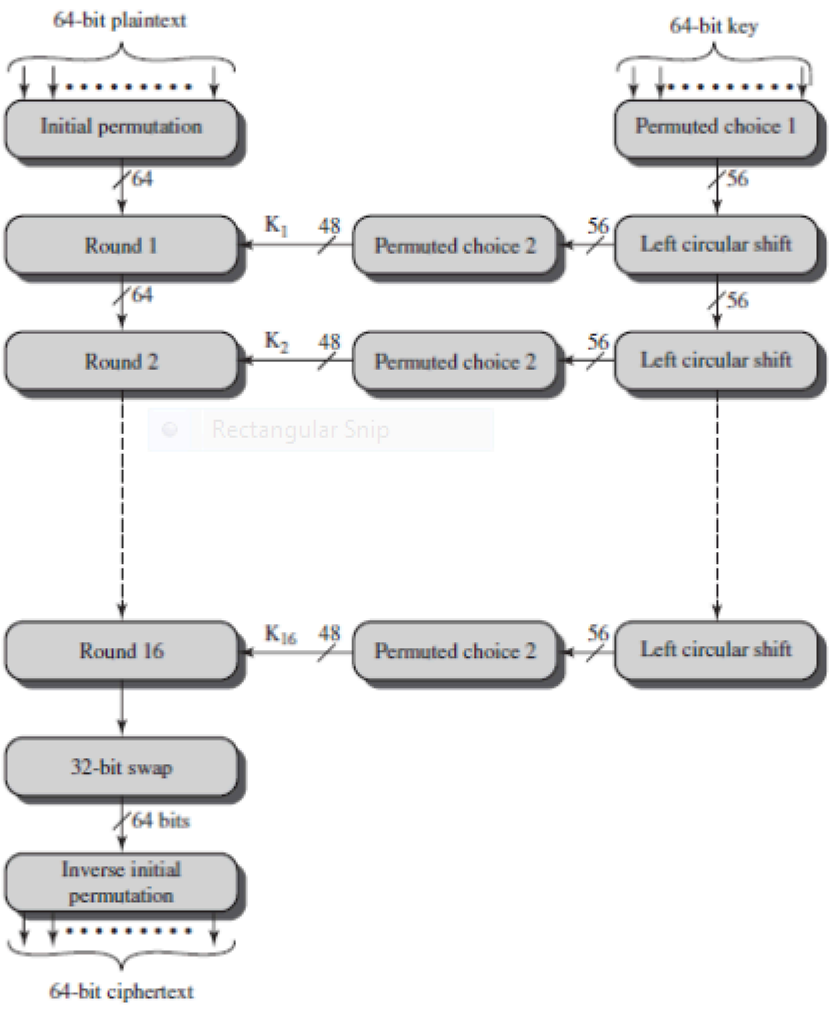


Figure: General Depiction of DES Encryption Algorithm

64 bit key is used but every 8th bit is the parity bit hence it is taken as 56 bit key. Initially the key is passed through the permutation function. For each 16 round, a sub key K_i is produced by the combination of left circular shift and permutation. The same permutation function is used in each round.

The plain text are processed through these phases

a) Initial Permutation
b) 16 rounds of same function
c) Swap
d) Final Permutation

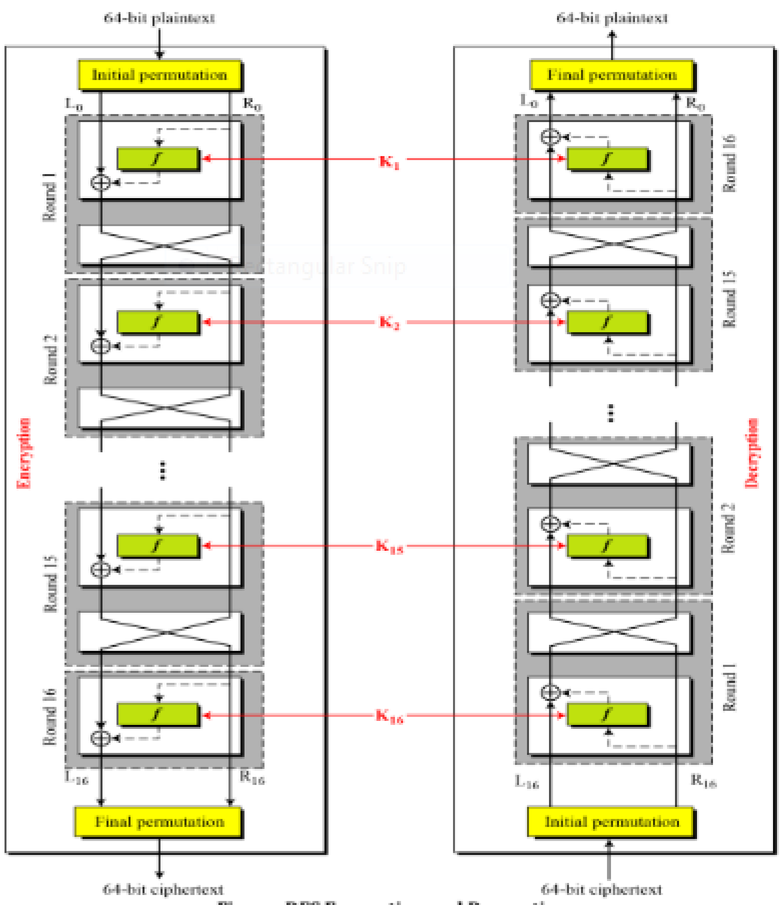
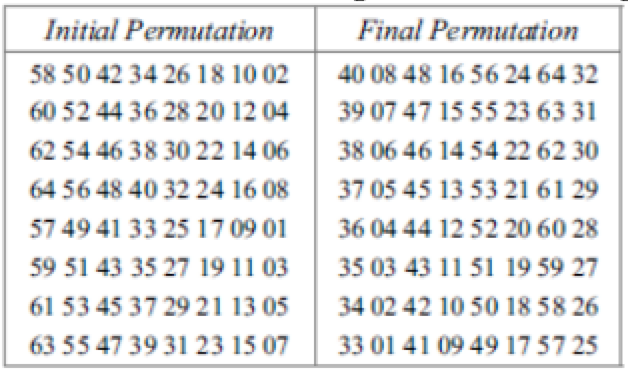


Figure: DES Encryption and Decryption

Initial Permutation and Final Permutation:

The input is 64 bit. These inputs are permuted according to a predefined rule. The permutation table contains a permutation of the number from 1 to 64. These permutation table and inverse permutation table can be designed such that the original bits can be restored.



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

DES Encryption:

a) In DES Encryption, there are two inputs to the encryption function:

i. the plaintext to be encrypted
ii. Key

b) In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.
c) The processing of the plaintext proceeds in three phases.
   i. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.
   ii. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions.
   iii. The left and right halves of the output are swapped to produce the **preoutput**.
   iv. Finally, the pre-output is passed through a permutation $[IP^{-1}]$ that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.
d) With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

Key Generation:

a) In DES, 56-bit key is used.
b) Initially, the key is passed through a permutation function.
a) Then, for each of the sixteen rounds, a *subkey* (K_i) is produced by the combination of a left circular shift and a permutation.
b) The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

DES Decryption:

a) As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.
b) Additionally, the initial and final permutations are reversed.

6) With the help of a neat diagram, explain the AES encryption and decryption process.

AES encryption and decryption with block diagram
AES doesn't use the Feistel structure. Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[*i*]. Four different stages are used, one of permutation and three of substitution:
a) Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
b) ShiftRows: A simple permutation
c) MixColumns: A substitution that makes use of arithmetic over GF(28)
d) AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key
The cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. AddRoundKey stage makes use of the key.The cipher begins and ends with an AddRoundKey stage. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the

result that $A \oplus B \oplus B = A$. In AES, the decryption algorithm is not identical to the encryption algorithm.

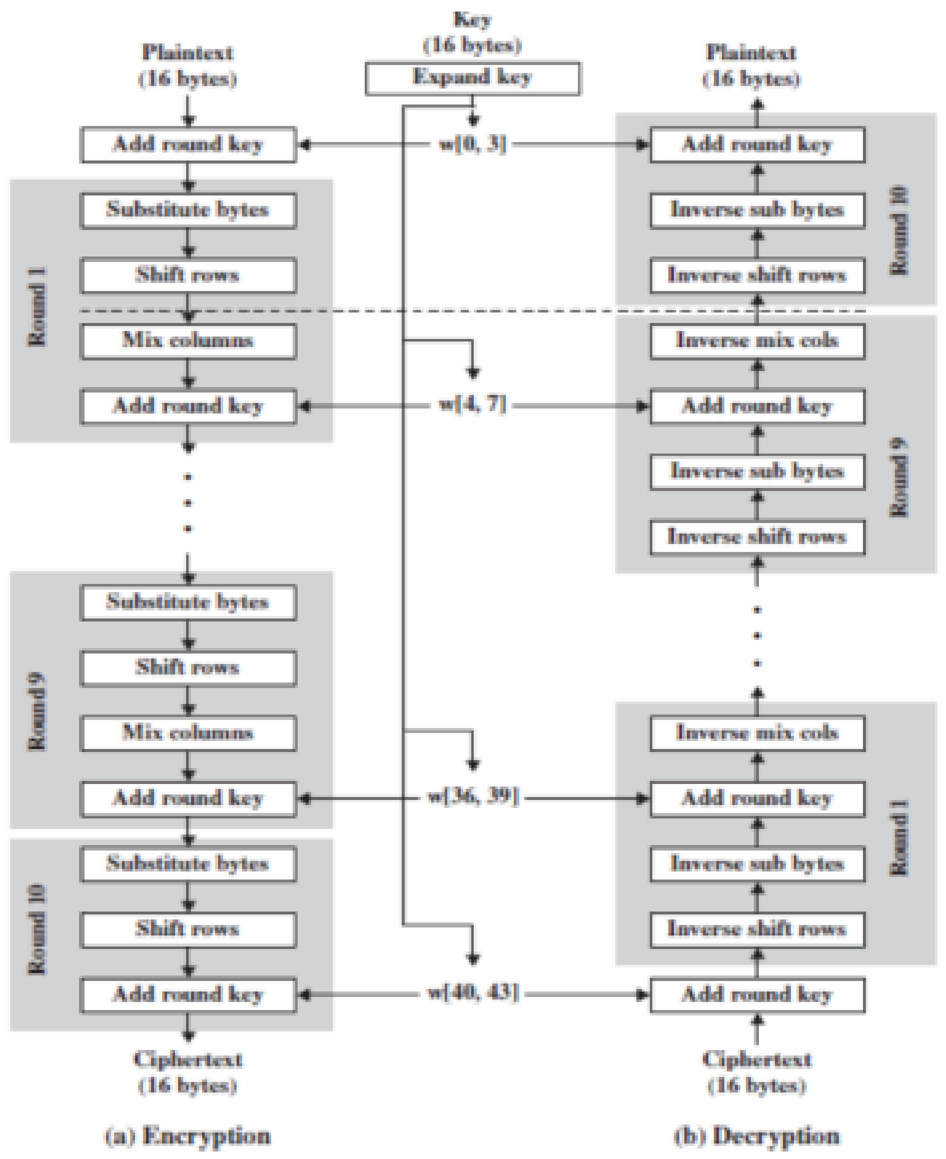


Figure: AES Encryption and Decryption

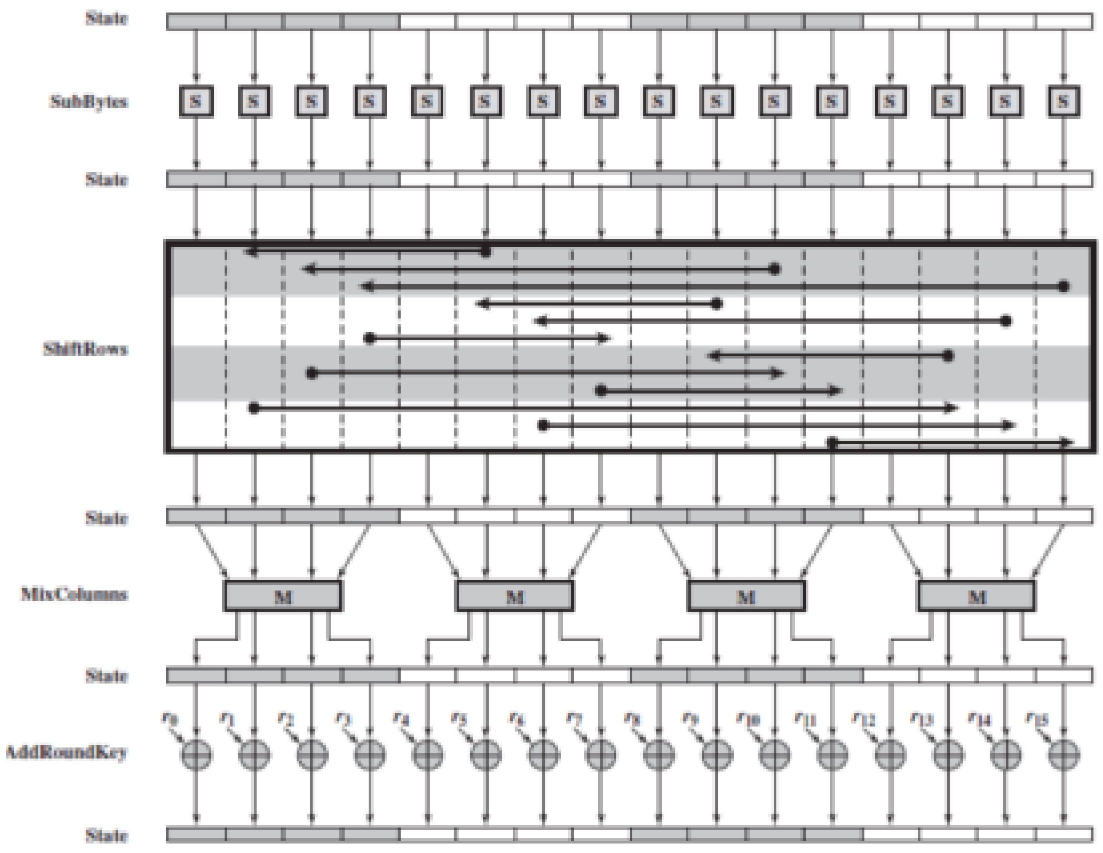

State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

Figure: AES Encryption Round

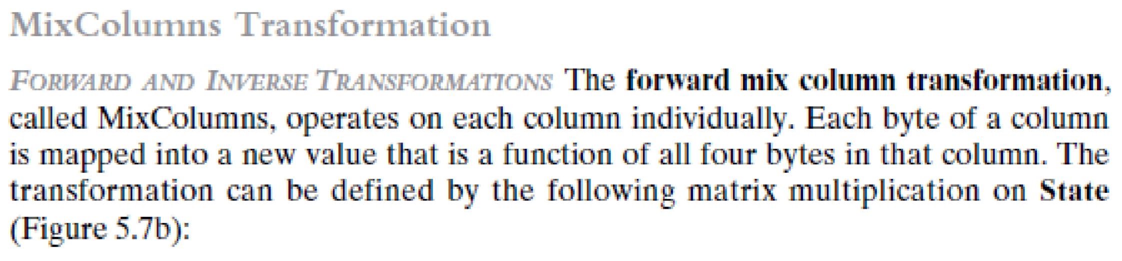
Figure: AES Encryption Round

As all stages are reversible, it is easy to perform decryption to recover the plain text. Encryption and decryption going in opposite vertical directions. The first N - 1 rounds consist of four distinct transformation functions:

- Sub Bytes,
- Shift Rows,
- Mix Columns,
- Add RoundKey

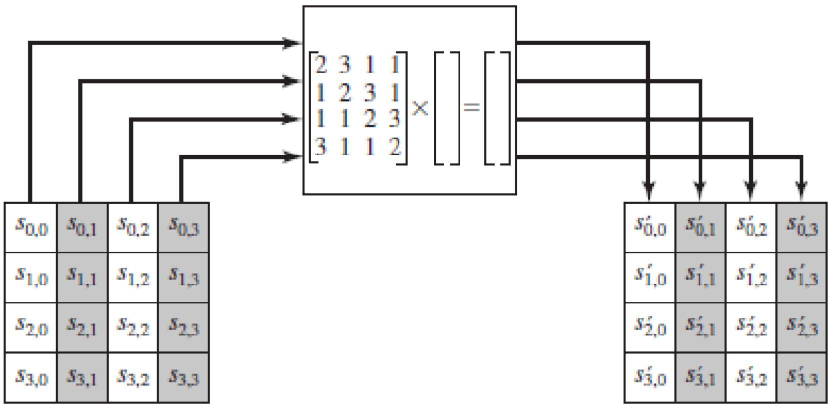
The final round contains only three transformations; those are SubBytes, ShiftRows and AddRoundKey, and there is an initial single transformation (AddRoundKey) before the first round, which can be considered Round 0.

7) With suitable examples describe the AES MixColumn transformation.



MixColumns Transformation

Forward and Inverse Transformations The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on **State** (Figure 5.7b):



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

(b) Mix column transformation

Figure 5.7 AES Row and Column Operations

$$\begin{bmatrix}02&03&01&01\\01&02&03&01\\01&01&02&03\\03&01&01&02\end{bmatrix}\begin{bmatrix}s_{0,0}&s_{0,1}&s_{0,2}&s_{0,3}\\s_{1,0}&s_{1,1}&s_{1,2}&s_{1,3}\\s_{2,0}&s_{2,1}&s_{2,2}&s_{2,3}\\s_{3,0}&s_{3,1}&s_{3,2}&s_{3,3}\end{bmatrix}=\begin{bmatrix}s'_{0,0}&s'_{0,1}&s'_{0,2}&s'_{0,3}\\s'_{1,0}&s'_{1,1}&s'_{1,2}&s'_{1,3}\\s'_{2,0}&s'_{2,1}&s'_{2,2}&s'_{2,3}\\s'_{3,0}&s'_{3,1}&s'_{3,2}&s'_{3,3}\end{bmatrix}\qquad\textbf{(5.3)}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications[5] are

performed in GF(2^8). The MixColumns transformation on a single column of **State** can be expressed as

$$\begin{aligned}s'_{0,j} &= (2\cdot s_{0,j})\oplus(3\cdot s_{1,j})\oplus s_{2,j}\oplus s_{3,j}\\ s'_{1,j} &= s_{0,j}\oplus(2\cdot s_{1,j})\oplus(3\cdot s_{2,j}\oplus s_{3,j}\\ s'_{2,j} &= s_{0,j}\oplus s_{1,j}\oplus(2\cdot s_{2,j})\oplus(3\cdot s_{3,j})\\ s'_{3,j} &= (3\cdot s_{0,j})\oplus s_{1,j}\oplus s_{2,j}\oplus(2\cdot s_{3,j})\end{aligned}\qquad\textbf{(5.4)}$$

The following is an example of MixColumns:

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

$\rightarrow$

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

Let us verify the first column of this example. In

GF(2^8), addition is the bitwise XOR operation and that multiplication can be per formed according to the rule established in Equation (4.14). In particular, multipli cation of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followec by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the origina value (prior to the shift) is 1. Thus, to verify the MixColumns transformation on the first column, we need to show that

It follows that multiplication by x (i.e., 00000010) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (00011011), which represents $(x^4 + x^3 + x + 1)$. To summarize,

$$x \times f(x) = \begin{cases} (b_6 b_5 b_4 b_3 b_2 b_1 b_0 0) & \text{if } b_7 = 0 \\ (b_6 b_5 b_4 b_3 b_2 b_1 b_0 0) \oplus (00011011) & \text{if } b_7 = 1 \end{cases} \qquad \textbf{(4.14)}$$

Multiplication by a higher power of x can be achieved by repeated application of Equation (4.14). By adding intermediate results, multiplication by any constant in GF(2^8) can be achieved.

$$\begin{array}{lllll}
(\{02\}\cdot\{87\}) \oplus (\{03\}\cdot\{6E\}) \oplus \{46\} & & \oplus \{A6\} & = & \{47\} \\
\{87\} \oplus (\{02\}\cdot\{6E\}) \oplus (\{03\}\cdot\{46\}) & & \oplus \{A6\} & = & \{37\} \\
\{87\} \oplus \{6E\} \oplus (\{02\}\cdot\{46\}) & & \oplus (\{03\}\cdot\{A6\}) & = & \{94\} \\
(\{03\}\cdot\{87\}) \oplus \{6E\} \oplus \{46\} & & \oplus (\{02\}\cdot\{A6\}) & = & \{ED\}
\end{array}$$

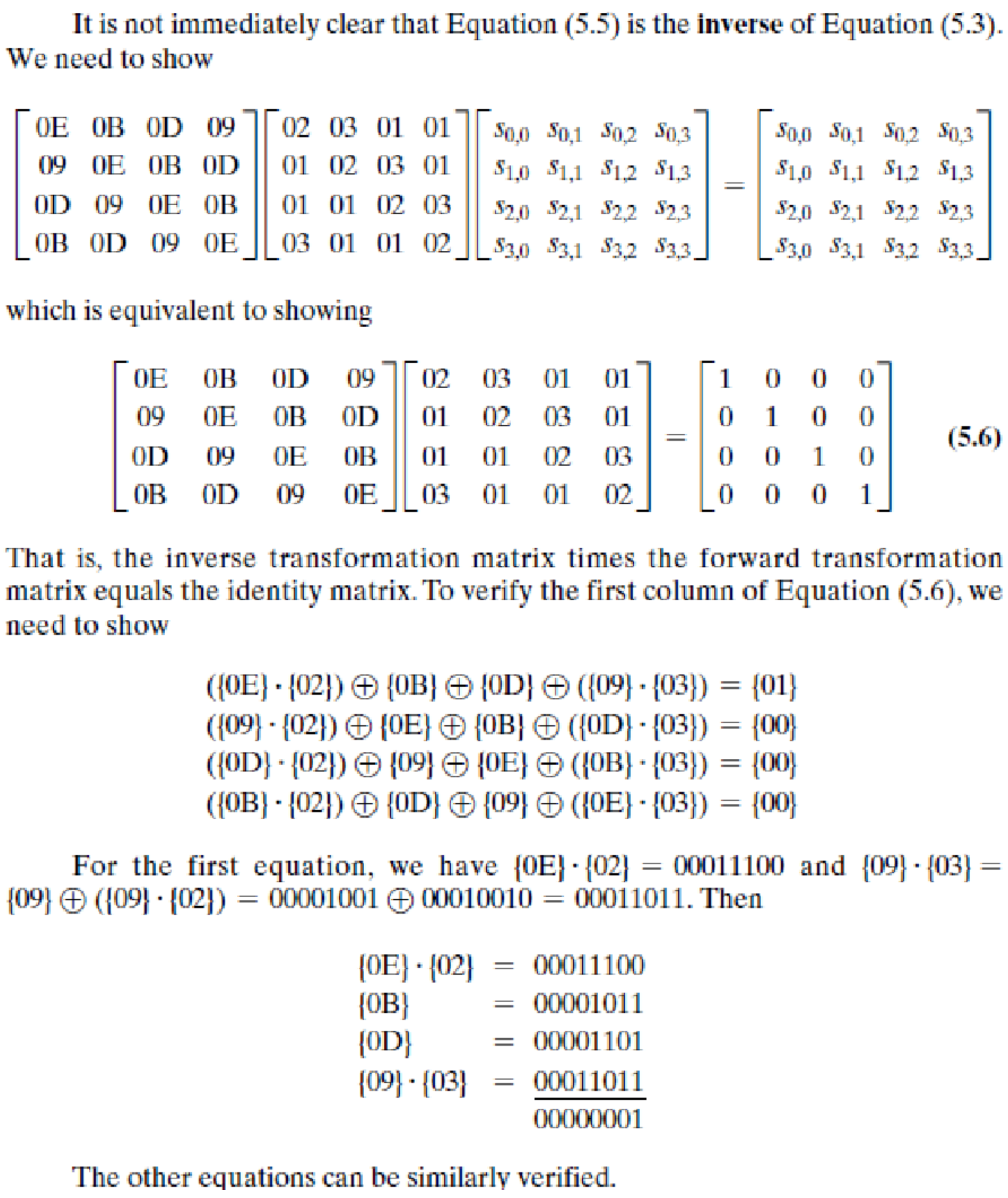
For the first equation, we have $\{02\}\cdot\{87\} = (0000\,1110) \oplus (0001\,1011) = (0001\,0101)$ and $\{03\}\cdot\{6E\} = \{6E\} \oplus (\{02\}\cdot\{6E\}) = (0110\,1110) \oplus (1101\,1100) = (1011\,0010)$. Then,

$$\begin{array}{lcl}
\{02\}\cdot\{87\} & = & 0001\,0101 \\
\{03\}\cdot\{6E\} & = & 1011\,0010 \\
\{46\} & = & 0100\,0110 \\
\{A6\} & = & \underline{1010\,0110} \\
 & & 0100\,0111 = \{47\}
\end{array}$$

The other equations can be similarly verified.

The **inverse mix column transformation**, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \qquad \textbf{(5.5)}$$



It is not immediately clear that Equation (5.5) is the **inverse** of Equation (5.3). We need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

which is equivalent to showing

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(5.6)}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of Equation (5.6), we need to show

$$\begin{aligned} (\{0E\}\cdot\{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\}\cdot\{03\}) &= \{01\} \\ (\{09\}\cdot\{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\}\cdot\{03\}) &= \{00\} \\ (\{0D\}\cdot\{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\}\cdot\{03\}) &= \{00\} \\ (\{0B\}\cdot\{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\}\cdot\{03\}) &= \{00\} \end{aligned}$$

For the first equation, we have $\{0E\}\cdot\{02\} = 00011100$ and $\{09\}\cdot\{03\} = \{09\} \oplus (\{09\}\cdot\{02\}) = 00001001 \oplus 00010010 = 00011011$. Then

$$\begin{aligned} \{0E\}\cdot\{02\} &= 00011100 \\ \{0B\} &= 00001011 \\ \{0D\} &= 00001101 \\ \{09\}\cdot\{03\} &= \underline{00011011} \\ & \ 00000001 \end{aligned}$$

The other equations can be similarly verified.

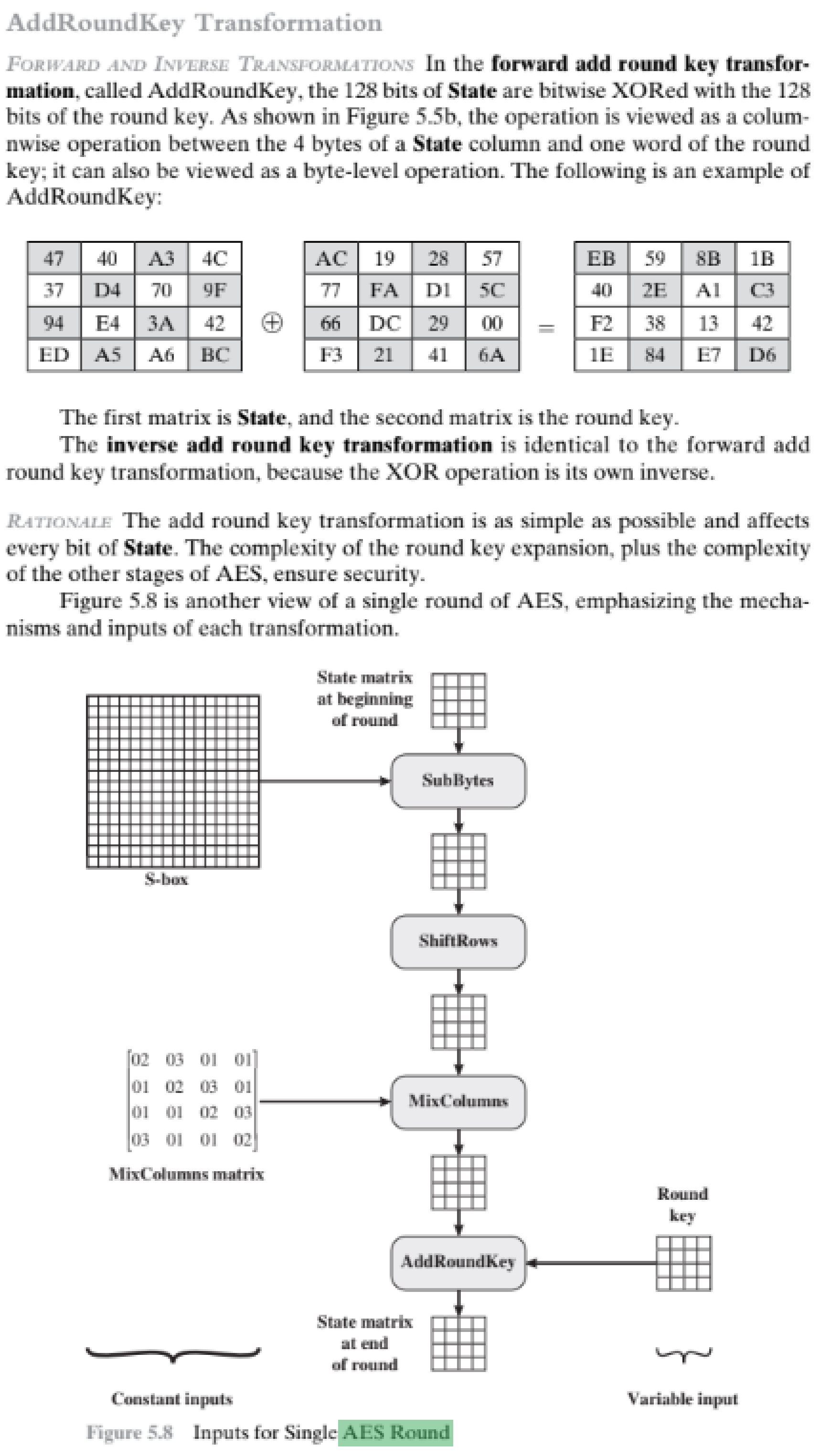
8) Illustrate the following with necessary diagrams:
   (i) AddRoundKey and SubBytes in AES.
   (ii) Single DES encryption.

Solution:

i) AddRoundKey and SubBytes in AES.

AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

Detailed Structure



AddRoundKey Transformation

Forward and Inverse Transformations In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. As shown in Figure 5.5b, the operation is viewed as a columnwise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:

$$\begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline AC & 19 & 28 & 57 \\ \hline 77 & FA & D1 & 5C \\ \hline 66 & DC & 29 & 00 \\ \hline F3 & 21 & 41 & 6A \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline EB & 59 & 8B & 1B \\ \hline 40 & 2E & A1 & C3 \\ \hline F2 & 38 & 13 & 42 \\ \hline 1E & 84 & E7 & D6 \\ \hline \end{array}$$

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

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

Rationale The add round key transformation is as simple as possible and affects every bit of **State**. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

Figure 5.8 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

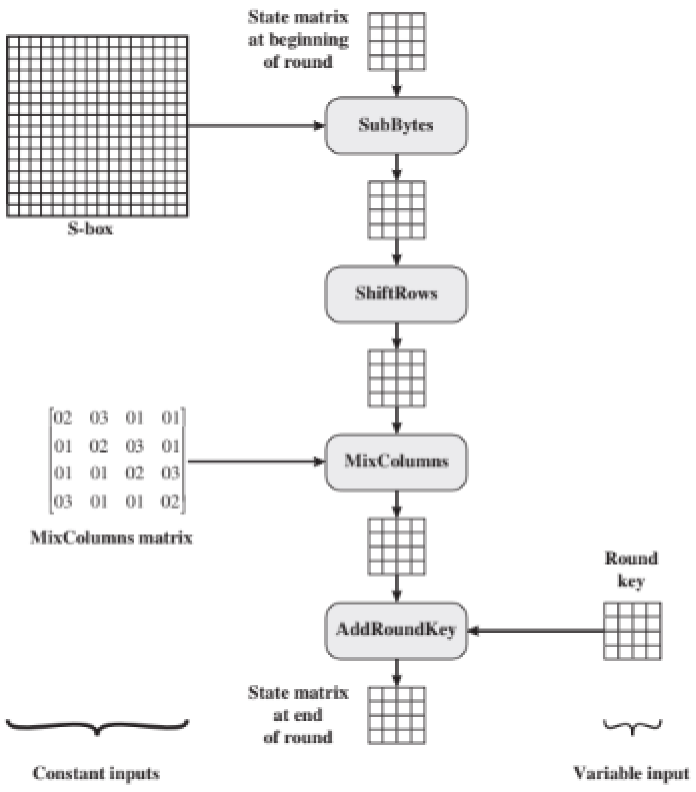
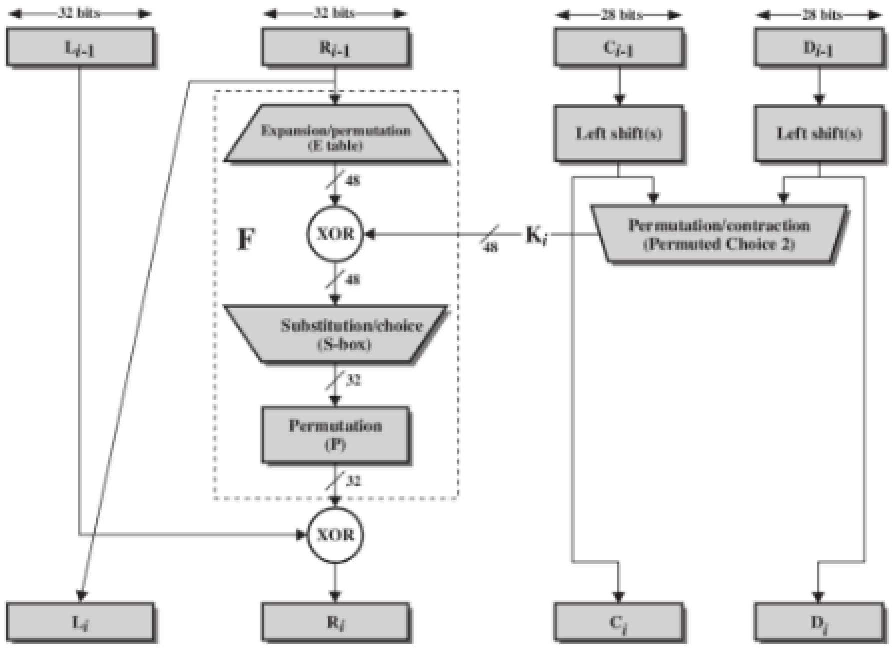


Figure 5.8 Inputs for Single AES Round

(ii) Single DES encryption.

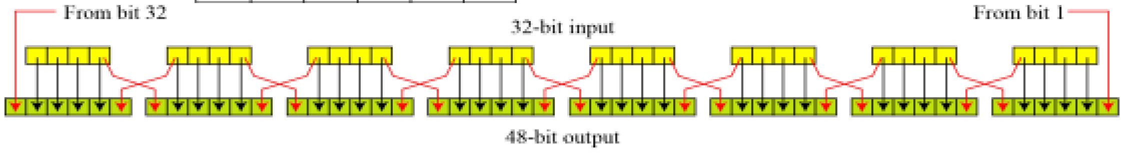
Figure below shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32- bit quantities, labelled L (left) and R (right). As in any classic Feistel cipher, the overall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

$$\boxed{\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{(i-1)} \oplus F\left(R_{(i-1)}, K_i\right) \end{aligned}}$$



Expansion: The round key K_i is 48 bits, and the R is 32 bits. The R is first expanded to 48 bits by using permutation plus expansion table as shown below.

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	28
24	25	26	27	28	29
28	29	30	31	32	1



XOR: The resulting 48 bits are XOR with K_i

Substitution Table: These 48 bits are passed through the substitution function that produces a 32 bits output which is permuted based on predefined rule as shown in table below.

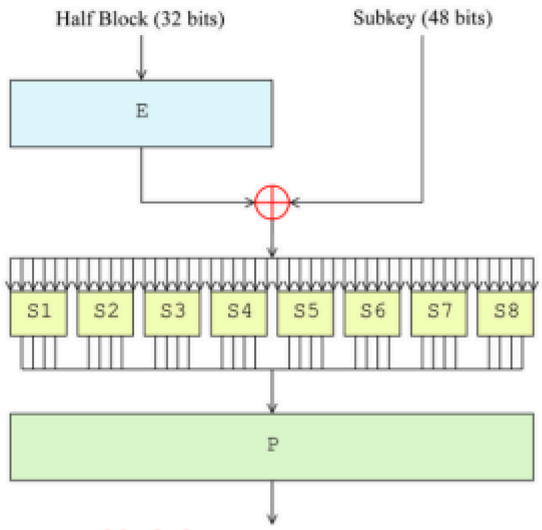
The round key K_i is 48 bits. The R input is 32 bits. This R input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the R bits. The resulting 48 bits are XORed with K_i. This 48-bit result passes through a substitution function that produces a 32-bit output. The role of the S-boxes in the function is illustrated in figure shown below. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output.

The substitution consists of 8 S-Boxes, which accepts 6 bits as input and produces 4 bits as output. The 1st and last bit of the input to S-Box S_i forms the row and the remaining 4 bits represents the column.

E.g. In S_1, for the input 011001, the row is 01 i.e. 1st row and 1100 i.e. 12th column, the value at 1st row and 12th column is 9 i.e. 1001.

The output of the S-Boxes is again permuted as

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25


Half Block (32 bits)
Subkey (48 bits)
E
S1
S2
S3
S4
S5
S6
S7
S8
P