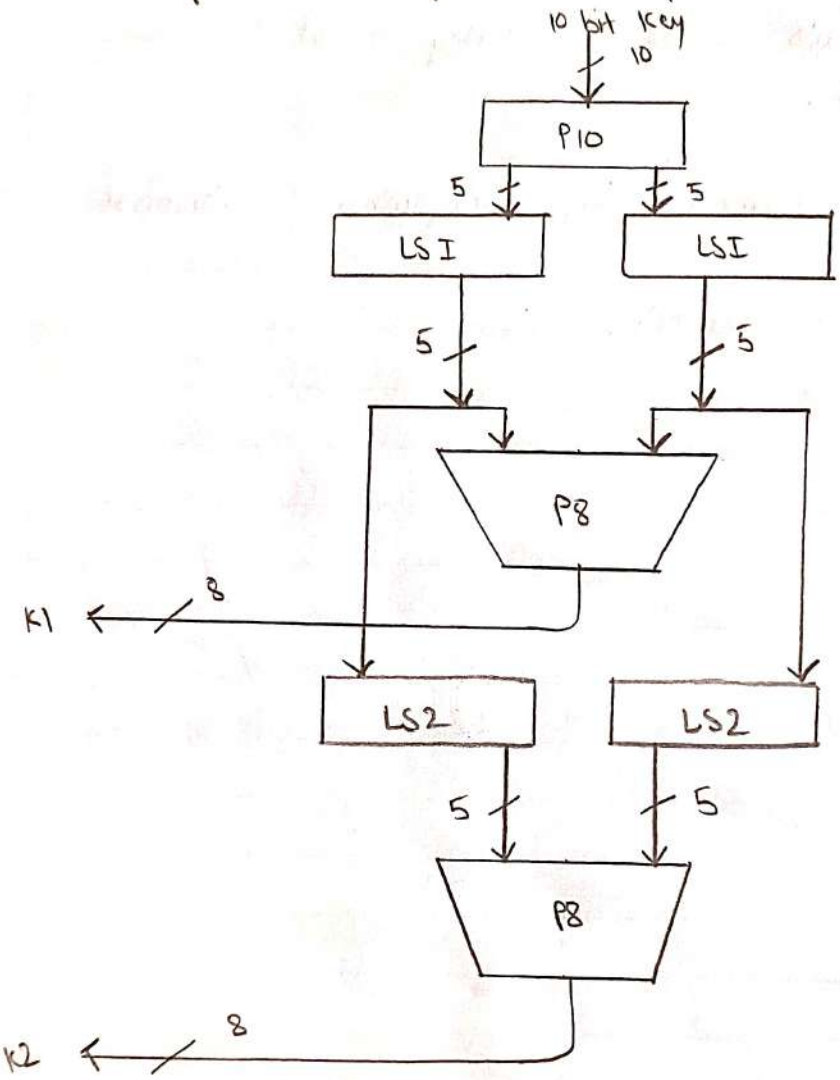
Practical No: 6

Aim: To implement simplified DES product cipher.



Date :

Practical No. 6

Aim: To implement simplified DES product cipher.

Theory:

S-DES is a simple version of DES algorithm. It is similar to the DES algorithm but is a smaller algorithm and has fewer parameter than DES. It was made for educational purposes so that understanding DES would become simpler. It is a block cipher that takes a block of plain text & converts it into ciphertext. It takes a block of 8 bit.

It is a symmetric key cipher ie. they use the same key for both encryption and decryption. In this article, we are going to demonstrate key generation for S-des encryption and decryption algorithm. we take a random 10-bit key and produce two 8 bit keys which will be used for encryption and decryption.

Key Generation Concept:

In the key generation algo., we accept the 10 bit key and convert it into two 8 bit keys. This key is shared betn sender and receiver

Step 1: we accepted a 10 bit key and permutated the bits by putting them in P10 table.

key = 1 0 1 0 0 0 0 0 1 0

$(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}) = (1, 0, 1, 0, 0, 0, 0, 0, 1, 0)$

P10 permutation is: $P10(K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}) =$

$(K_3, K_5, K_2, K_7, K_4, K_{10}, K_1, K_9, K_8, K_6)$

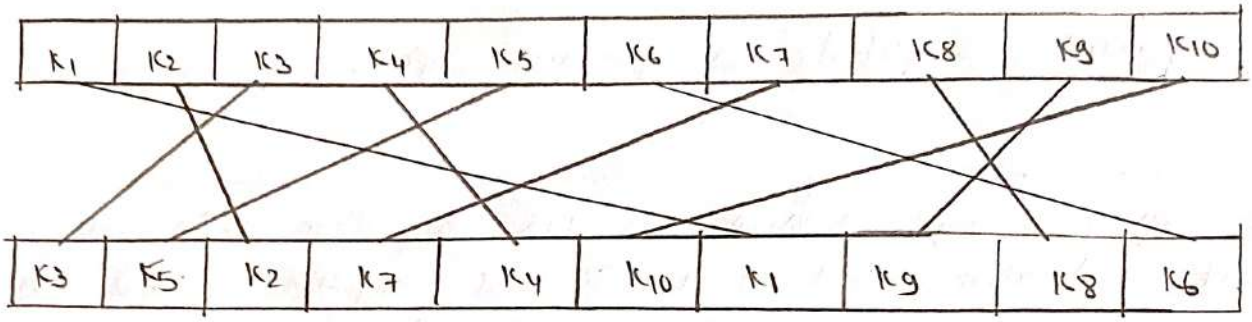
After P10, we get 1 0 0 0 0 0 1 1 0 0

Step 2: we divided the key into 2 halves of 5 bit each

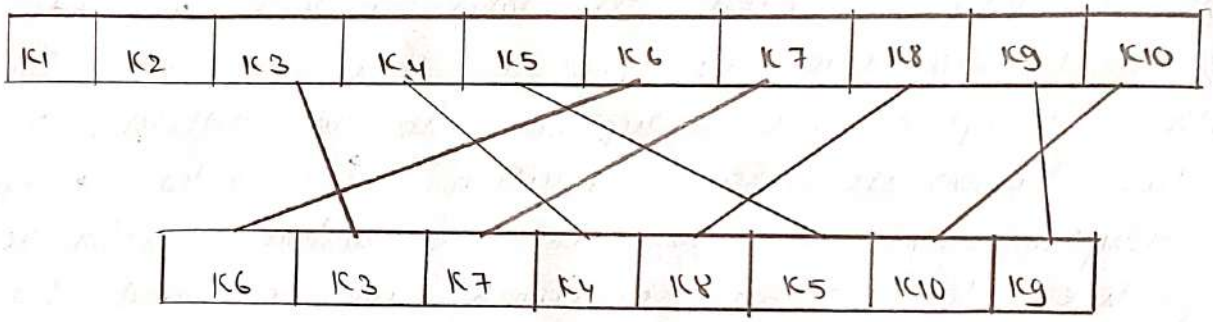
L = 1 0 0 0 0 , R = 0 1 1 0 0

Page No.

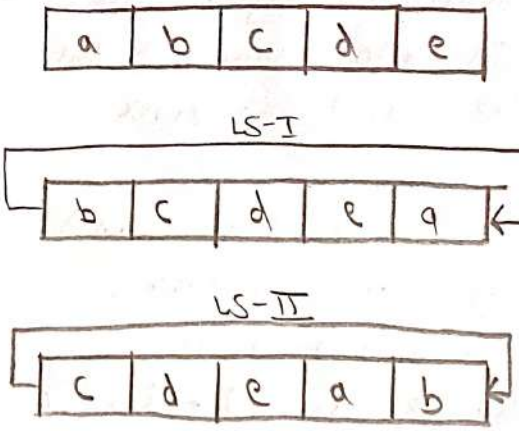
1. permutation P10:



2. permutation P8:



3. left shift:



Date :

Step 3 : Now we apply one bit left-shift on each key.
I = 0 0 0 0 1 , II = 1 1 0 0 0

Step 4 : combine both keys after step 3 and permute the bits by putting them in the P8 table. The output of the given table is the first key K1.
After LS-1 combined, we get 0 0 0 0 1 1 1 0 0 0
P8 permutation is : P8 (K1, K2, K3, K4, K5, K6, K7, K8, K9, K10) =
(K6, K3, K7, K4, K8, K5, K10, K9)
After P8, we get key-1 = 1 0 1 0 0 1 0 0

Step 5 : The o/p obtained from step 3 i.e. 2 halves after one bit left shift should again undergoes the process of two bit left shift.
Step 3 output - I = 0 0 0 0 1 , II = 1 1 0 0 0
After 2 bit shift I = 0 0 1 0 0 , II = 0 0 0 1 1

Step 6 : Combine the 2 halves obtained from step 5 and permute them by putting them in the P8 table. The o/p of the given table is the second key K2.
After LS-2 combined = 0 0 1 0 0 0 0 0 1.
P8 permutation is : P8 (K1, K2, K3, K4, K5, K6, K7, K8, K9, K10) =
(K6, K3, K7, K4, K8, K5, K10, K9)
After P8, we get key-2 : 0 1 0 0 0 0 1 1

Final Output :
key - 1 is : 1 0 1 0 0 1 0 0
key - 2 is : 0 1 0 0 0 0 1 1

Page No.

Date :

program :

```
package anproduct;
  public class anproduct {
     int key[] = {
          1,0,1,0,0,0,0,0,1,0
     };
     int P10[] = {3,5,2,7,4,10,1,9,8,6};
     int P8[] = {6,3,7,4,8,5,10,9};

     int key1[] = new int[8]
     int key2[] = new int[8]

     int[] IP = {2,6,3,1,4,8,5,7};
     int[] EP = {4,1,2,3,2,3,4,1};
     int[] P4 = {2,4,3,1};
     int[] IP-inv = {4,1,3,5,7,2,8,6};

     int[][] SO = {{1,0,3,2},
                   {3,2,1,0},
                   {0,2,1,3}};

     void key-generation()
     {
        int key-[] = new int[10];

        for (int j=0; j<10; j++) {
            key-[i] = key[P10[j]-1];
        }
        int LS[] = new int[5];
        int RS[] = new int[5];
        for (int i=0 ; i<5 ; i++) {
            LS[i] = key-[i];
```

Date :

```
    int RS[i] = key-[i+5];
}
int [] LS-1 = shift (LS, 1);
int [] RS-1 = shift (RS, 1);
for (int i=0; i<5; i++) {
    key-[i] = LS-1[i];
    key-[i+5] = RS-1[i];
}
for (int i=0 ; i<8; i++) {
    key1[i] = key-[PB[i] -1];
}
int [] LS-2 = shift (LS, 2);
int [] RS-2 = shift (RS, 2);
for (int i=0 ; int ; i<5 ; i++) {
    key-[i] = LS-2[i];
    key-[i+5] = RS-2[i];
}
for (int i = 0; i<8 ; i++) {
    key2[i] = key-[PB[i] -1];
}
System.out.println ("your key-1 : ");
for (int i=0 ; i<8 ; i++)
    System.out.print (key1[i] + " ");
System.out.println();
}
    while (n>0) {
        int temp = ar[0];
        for (int i=0 ; i < ar.length -1 ; i++) {
            ar[i] = ar[i+1];
        }
        ar[ar.length -1] = temp ;
```

Page No.

Date :

```
        n--;
    }
    return arr;
}
int [] encryption (int [] plaintext)
{
    int [] arr = new int [8];
    for (int i = 0; i < 8; i++) {
        arr[i] = plaintext [IP[i] - 1];
    }
    int [] arr1 = function_(arr, key1);
    int [] after_swap = swap (arr1, arr1.length/2);
    int [] arr2 = function_(after_swap, key2);
    int [] ciphertext = new int [8];
    for (int i=0; i<8; i++){
        ciphertext [i] = arr2 [IP-inv[i] - 1];
    }
    return ciphertext;

String binary_(int val)
{
    if (val == 0)
        return "00";
    else if (val == 1)
        return "01";
    else if (val == 2)
        return "10";
    else
        return "11";
}
int [] function_(int[] arr, int[] key_)
{
```

Page No.

Date :

```
int[] l = new int[4];
int[] r = new int[4];

for (int i=0; i<4; i++) {
    l[i] = or[i];
    r[i] = or[i+4];
}

int[] ep = new int[8];
for (int i=0; i<8; i++) {
    ep[i] = r[EP[i]-1];
}

for (int i=0; i<8; i++) {
    or[i] = key_[i] ^ ep[i];
}

int[] l_1 = new int[4];
int[] r_1 = new int[4];

for (int i=0; i<4; i++) {
    l_1[i] = or[i];
    r_1[i] = or[i+4];
}

int row, col, val;

row = Integer.parseInt("" + l_1[0] + l_1[3], 2);
col = Integer.parseInt("" + l_1[1] + l_1[2], 2);
val = S0[row][col];
String str_1 = binary_(val);
```

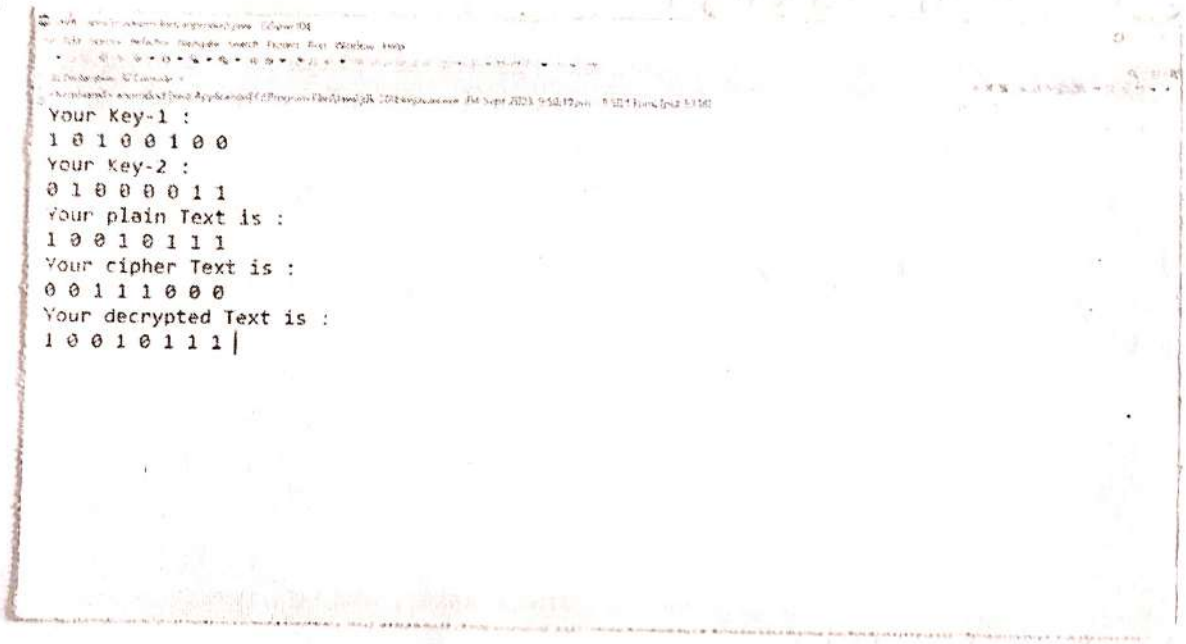
Date :

```
row = Integer.parseInt("" + x-1[0] + x-1[3] , 2 );
col = Integer.parseInt("" + x-1[1] + x-1[2] , 2 );
val = s1[row][col];
String str-x = binary-(val);

int[] x_ = new int[4];
for (int i=0 ; i<2; i++) {
    char c1 = str-1.charAt(i);
    char c2 = str-x.charAt(i);
    x-[i] = character.getNumericValue(c1);
    x-[i+2] = character.getNumbericValue(c2);
}
int[] x-p4 = new int[4];
for (int i=0 ; i<4 ; i++) {
    x-p4[i] = x-[p4[i]-1];
}

for (int i=0 ; i<4 ; i++) {
    d[i] = d[i] ^ x-p4[i];
}
int[] output = new int[8];
for (int i = 0 ; i<4 ; i++) {
    output [i] = d[i];
    output [i+4] = x[i];
}
return output;
}

int[] swap(int[] array, int n)
{
    int[] d = new int[n];
```

Date :

```
int [] r = new int [n];

for (int i = 0; i < n ; i++ ) {
    l[i] = array [i] ;
    r[i] = array [i+n];
}

int [] output = new int [2*n] ;
for (int i = 0; i < n ; i++) {
    output [i] = r[i];
    output [i+n] = l[i];
}

return output ;
}

int [] decryption (int [] ar)
{
int [] arr = new int [8];
for (int i = 0; i < 8; i++) {
    arr[i] = ar [IP[i] -1];
}

int [] arr1 = function - (arr, key2);
int [] after_swap = swap (arr1, arr1.length /2);
int [] arr2 = function - (after_swap , key1 );
int [] decrypted = new int [8];
for (int i = 0 ; i < 8 ; i++ ) {
    decrypted [i] = arr2 [IP_inv [i] -1];
}

return decrypted;
}
```

Output :



Conclusion :

Simplified - DES implemented successfully.

Date :

```
public static void main(String[] args )
{
    GfG obj = new GfG();
    obj.key_generation ();

    int [] plaintext = { 1, 0, 0, 1, 0, 1, 1, 1 } ;
    System.out.println ();
    System.out.println (" your plain Text is : " );
    for (int i= 0 ; i<8 ; i++ )
        System.out.print (plaintext [i] +" " );
    int [] ciphertext = obj. encryption (plaintext );

    System.out.println ();
    System.out.println ( " your cipher text is : " );

    for (int i = 0 ; i<8 ; i++ )
        System.out.print (ciphertext [i] +" " );
    int [] decrypted = obj. decryption (ciphertext );

    System.out.println ()
    System.out.println your decrypted text is" );

    for (int i = 0 ; i<8 ; i++ )
        System.out.print ( decrypted [i] +" " );
    }
}
```

Conclusion :

Simplified - DES implemented successfully.

Date :

Viva Voce :

(1) what is difference betn SDES and DES ?

→ SDES is simple version of DES algo. It is similar to the DES algo. but is smaller algo. and has few parameter than DES. It was made for educational purpose so that understanding DES would become simpler.

(2) What is key size in SDES ?

→ 10 bits

(3) What are disadvantages of S-DES ?

→ (1) It was never intended for a real-world use as it only for educational use.

(2) It lacks robustness and security features required to protect data against threats.

(3) May provide high level security due to simplicity & small key size.