Experiment 6

Aim: Write a program to implement S-DES SubKey Generation algorithm.

Theory: Simplified-DES The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

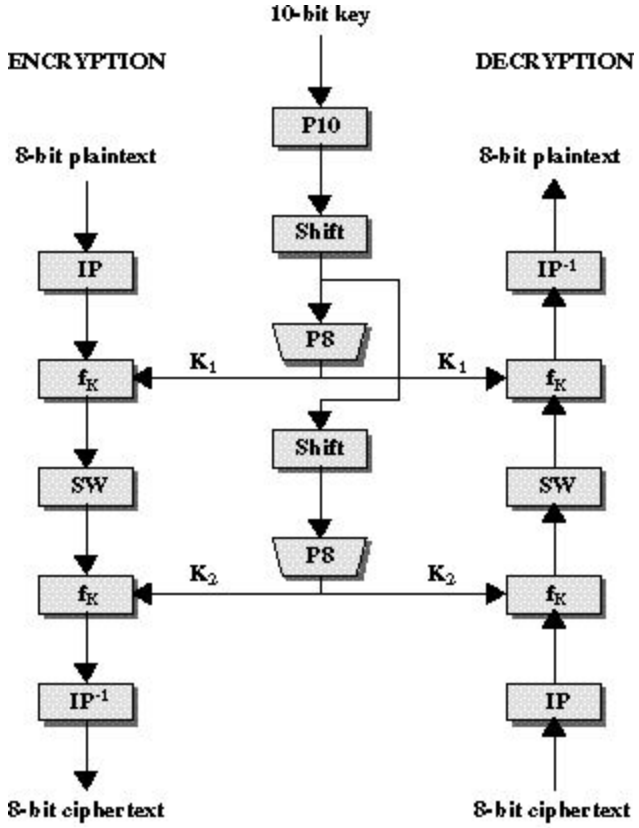

Figure 3.1 Simplified DES Scheme

The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled fk, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function fk again
- a permutation function that is the inverse of the initial permutation

The function fk takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the

shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition of functions, which can be written as where The decryption algorithm can be shown as:

Ciphertext = IP-1 (fK2 (SW (fk1 (IP (plaintext)))))

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

The decryption algorithm can be shown as:

Plaintext = IP-1 (fK1 (SW (fk2 (IP (ciphertext)))))

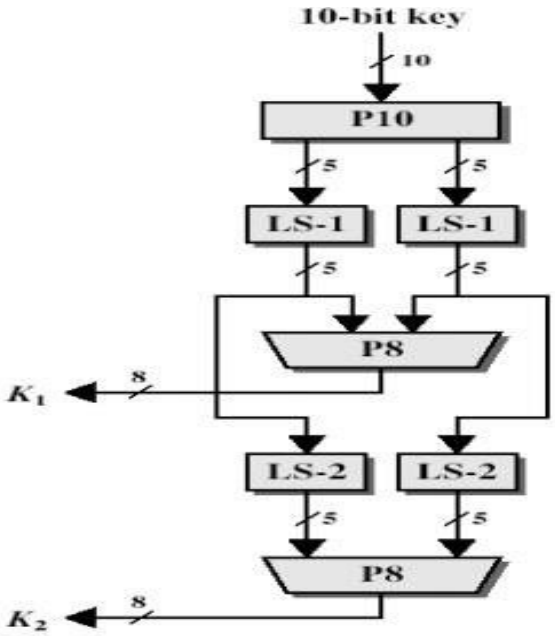

Figure: key generation for S-DES

S-DES key generation

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, k2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10

3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to

(10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8

6	3	7	4	8	5	10	9

The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

Source Code:

```
import java.io.*;
import java.lang.*;
class SDES
    {
     public int K1, K2;
     public static final int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
     public static final int P10max = 10;
     public static final int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9};
     public static final int P8max = 10;
     public static final int P4[] = { 2, 4, 3, 1};
     public static final int P4max = 4;
     public static final int IP[] = { 2, 6, 3, 1, 4, 8, 5, 7};
     public static final int IPmax = 8;
     public static final int IPI[] = { 4, 1, 3, 5, 7, 2, 8, 6};
     public static final int IPImax = 8;
     public static final int EP[] = { 4, 1, 2, 3, 2, 3, 4, 1};
     public static final int EPmax = 4;
     public static final int S0[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,
                                        3},{ 3, 1, 3, 2}};
     public static final int S1[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,
                                        2},{ 2, 1, 0, 3}};

    public static int permute( int x, int p[], int pmax)
    {
      int y = 0;
      for( int i = 0; i < p.length; ++i)
```

```
        {
          y <<= 1;
          y |= (x >> (pmax - p[i])) & 1;
       }
       return y;
     }

public static int F( int R, int K)
 {
        int t = permute( R, EP, EPmax) ^ K;
        int t0 = (t >> 4) & 0xF;
        int t1 = t & 0xF;
        t0 = S0[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];
        t1 = S1[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];
         t = permute( (t0 << 2) | t1, P4, P4max);
       return t;
 }

public static int fK( int m, int K)
      {
         int L = (m >> 4) & 0xF;
         int R = m & 0xF;
         return ((L ^ F(R,K)) << 4) | R;
      }

public static int SW( int x)
{
 return ((x & 0xF) << 4) | ((x >> 4) & 0xF);
}

 public byte encrypt( int m)
   {
       System.out.println("\nEncryption");
        m = permute( m, IP, IPmax);
       System.out.print("\nAfter Permutation : ");
        printData( m, 8);
        m = fK( m, K1);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
```

```
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore IP inverse : ");
        printData( m, 8);
        m = permute( m, IPI, IPImax);
        return (byte) m;
    }

  public byte decrypt( int m)
      {
        System.out.println("\nDecryption");
        System.out.print("\nReceived data : ");
        printData( m, 8);
        m = permute( m, IP, IPmax);
        System.out.print("\nAfter Permutation : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K1);
        System.out.print("\nBefore Extraction Permutation : ");
        printData( m, 4);
        m = permute( m, IPI, IPImax);
        System.out.print("\nAfter Extraction Permutation : ");
        printData( m, 8);
        return (byte) m;
      }

public static void printData( int x, int n)
       {
         int mask = 1 << (n-1);
         while( mask > 0)
         {
         System.out.print( ((x & mask) == 0) ? '0' : '1');
```

```
        mask >>= 1;
        }
      }

   public SDES( int K)
    {
      K = permute( K, P10, P10max);
      int t1 = (K >> 5) & 0x1F;
      int t2 = K & 0x1F;
      t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);
      t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);
      K1 = permute( (t1 << 5)| t2, P8, P8max);
      t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);
      t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);
      K2 = permute( (t1 << 5)| t2, P8, P8max);
    }
    }

public class SimplifiedDES
    {
     public static void main( String args[]) throws Exception
     {
      DataInputStream inp = new DataInputStream(System.in);
      System.out.println("Enter the 10 Bit Key :");
      int K = Integer.parseInt(inp.readLine(),2);
      SDES A = new SDES( K);
      System.out.println("Enter the 8 Bit message To be Encrypt  : ");
      int m = Integer.parseInt(inp.readLine(),2);
      System.out.print("\nKey K1: ");
      SDES.printData( A.K1, 8);
      System.out.print("\nKey K2: ");
      SDES.printData( A.K2, 8);
      System.out.print("\n");

      m = A.encrypt( m);
      System.out.print("\n\nEncrypted Message: ");
      SDES.printData( m, 8);
      System.out.print("\n");
      m = A.decrypt( m);
```

```
        System.out.print("\n");
        System.out.print("\nDecrypted Message: ");
        SDES.printData( m, 8);
        System.out.print("\n");
      }
   }
```

Output:

```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java SimplifiedDES
Enter the 10 Bit Key :
1100011110
Enter the 8 Bit message To be Encrypt  :
00101000

Key K1: 11101001
Key K2: 10100111

Encryption

After Permutation : 00100010
before Swap : 00110010
After Swap : 00100011
before IP inverse : 00010011

Encrypted Message: 10001010

Decryption

Received data : 10001010
After Permutation : 00010011
before Swap : 00100011
After Swap : 00110010
Before Extraction Permutation : 0010
After Extraction Permutation : 00101000

Decrypted Message: 00101000
```

Learning Outcomes:

DES ciphers are hard to break using the brute force approach. Although number of permutations is not too much, the use of 2 or 3 keys in DES makes it difficult to crack. Using differential cryptanalysis can however prove to be more efficient than brute force attack in DES , still from calculations and observations we see that DES is resistant to cryptanalysis.