
Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

# T.Y.B.Tech (CSE)

## Information Security

## Lab Assignment No – A2

**Name: Aniruddha Shende**

**Roll number: PE04**

**Batch: E1**

**Panel: E**

PE04- Aniruddha Shende

Name :- Aniruddha Arun Shende PE-04
Roll no:- PE04
Batch :- E1
Panel :- E

Subject :- INFORMATION SECURITY

## LAB ASSIGNMENT - A2

Implement simple DES symmetric key algorithm using Python or Java or C++.

Aim :- Implement Simplified DES symmetric key algo. using Python / Java / C++.

Objective :-
① Learn DES algorithm
② Implement it using Python / Java / C++.

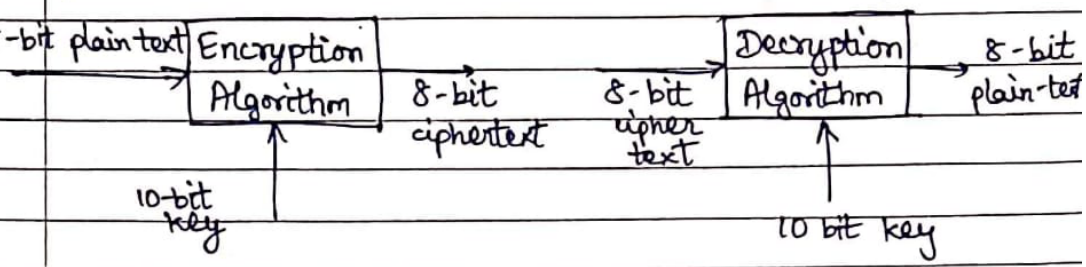
Theory :-

* Simplified DES

S-DES is kind of educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters.

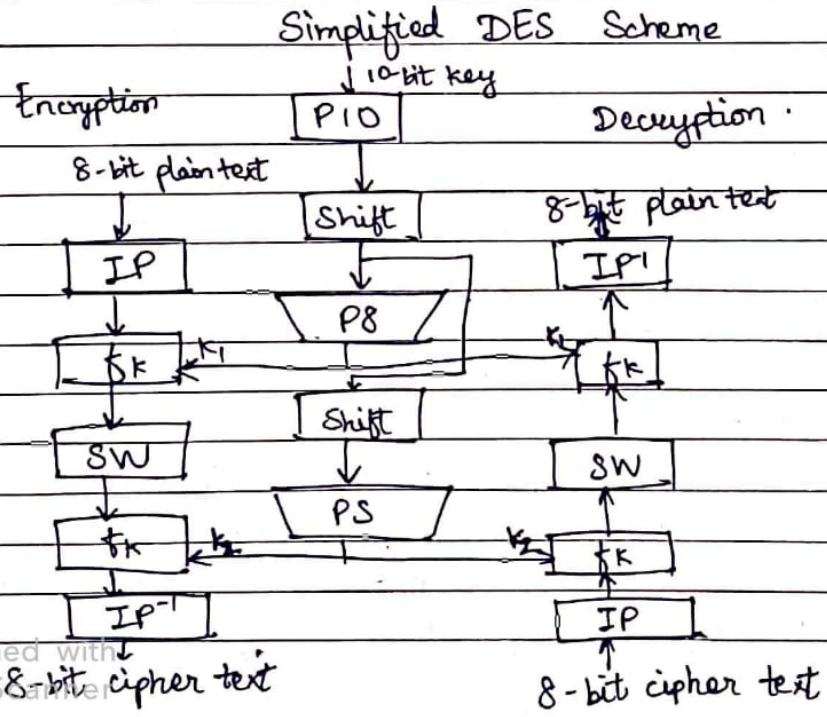
The S-DES encryption algorithm takes an 8-bit block of plain-text & 10 bit-key as input & produces an 8-bit block of cipher Text as output. And similarly, S-DES decryption algorithm takes an 8-bit block of ciphertext & same 10-bit key used to produce the ciphertext as input &

produces the original 8-bit block of plaintext as output.




The DES Algorithm involves 5 functions:-

1. An initial permutation (IP);
2. A complex function called $f_K$, which involves both permutation & substitution operations & depends on a key input.
3. A simple permutation function that switches 2 halves of the data.
4. The function $f_K$ again;
5. A permutation function that is the inverse of the initial permutation. ($IP^{-1}$).



Conclusion:-

Thus, we have successfully implemented simple DES symmetric key algorithm using C++.

FAQ's :-

1) What is the concept of Fiestal Cipher.

Ans1) Fiestal Cipher model is a structure or a design used to develop many block ciphers such as DES.

In general, Fiestal cipher may have invertible, non-invertible & self invertible components in its design. Same encryption as well as decryption is used. However same round keys are used for encryption & decryption as well.

2) Draw & describe ~~de~~ DES algorithm briefly.

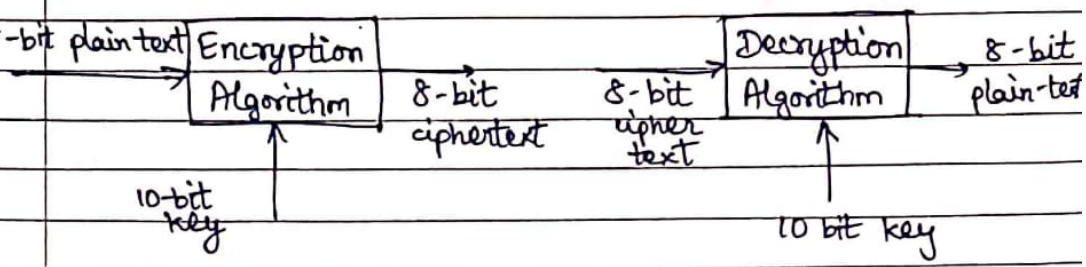
Ans2). Simplified DES algorithm was developed by Proffessor Edward Schaefer of Santa Clara University.

The S-DES encryption algorithm takes an 8-bit block of plaintext & a 10 bit key as a input & produces an 8-bit block of ciphertext as output.

The S-DES algorithm for decryption takes an 8-bit block of cipher text & produces a ciphertext using same 10-bit key & also

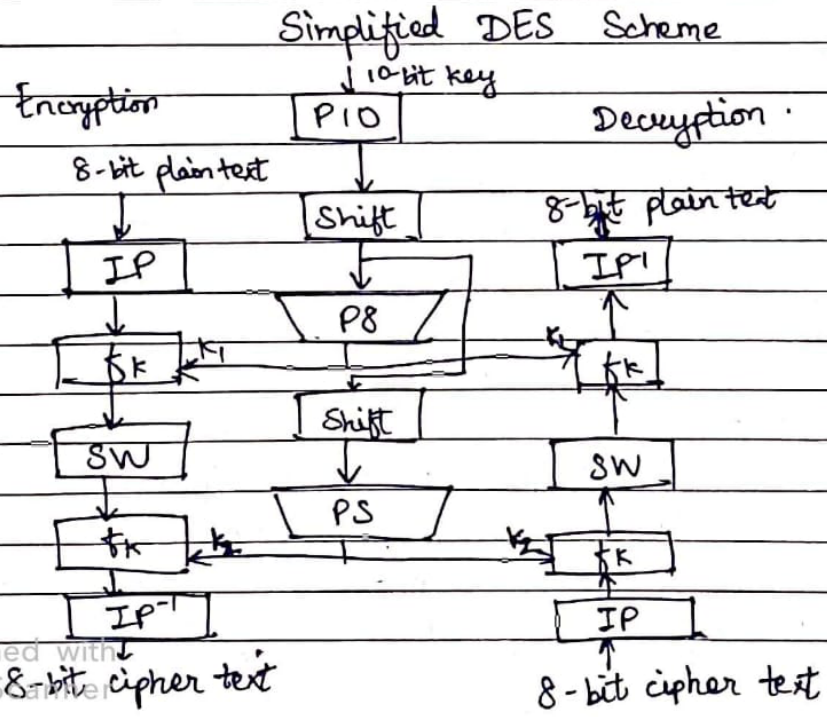
—

produces the original 8-bit block of plaintext as output.




The DES Algorithm involves 5 functions:-

1. An initial permutation (IP);
2. A complex function called $f_K$, which involves both permutation & substitution operations & depends on a key input.
3. A simple permutation function that switches 2 halves of the data.
4. The function $f_K$ again;
5. A permutation function that is the inverse of the initial permutation. ($IP^{-1}$).



3) List & state broad level operations used internally in DES algorithm.

Ans 3). DES algorithm internally makes use of multiple stages of permutation & substitution The function $f_K$ takes as input the data & 8-bit key.

The DES algorithm involves five functions:-

① Initial Permutation
② Complex function $f_K$
③ Simple Permutation
④ Again Complex $f^n$ $f_K$.
⑤ Inverse of initial permutation.

4) Compare various block ciphers such as DES, AES & Blowfish, etc.

Ans 4) ⓐ Blowfish algorithm is a better choice if time & memory are major constraint.
ⓑ If confidentiality & integrity are major factors, then AES algorithm can be selected.
ⓒ If the demand for application is the network bandwidth, then DES is the best option.

5) What are the Block cipher design guidelines.

Ans 5) ⓐ Number of Rounds:- It reflects the number of rounds to be suitable for an algorithm to make it more complex.
ⓑ Design of function F:- The complexity of cryptanalysis can be derived from the round function.
ⓒ Key schedule algorithm:- Greater complexity of in this algorithm, leads to greater difficulty of cryptanalysis.

# Program Code :

```
// Name : Aniruddha Shende
// Roll no : PE04
// Batch : E1
// Panel : E
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;

int S0[4][4] = {
    {1, 0, 3, 2},
    {3, 2, 1, 0},
    {0, 2, 1, 3},
    {3, 1, 3, 2}};
int S1[4][4] = {
    {0, 1, 2, 3},
    {2, 0, 1, 3},
    {3, 0, 1, 0},
    {2, 1, 0, 3}};

typedef std::vector<int> int_vec;
#define TWIDTH 25

int P10[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
int P8[8] = {6, 3, 7, 4, 8, 5, 10, 9};
int P4[4] = {2, 4, 3, 1};
int IP[8] = {2, 6, 3, 1, 4, 8, 5, 7};
int INVERSE_IP[8] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[8] = {4, 1, 2, 3, 2, 3, 4, 1};

string vector_to_string(int_vec v)
{
```

```
    string s = "";
    for (int i = 0; i < v.size(); i++)
    {
        s += to_string(v[i]);
        s += " ";
    }
    return s;
}

void print_vector(int_vec &v)
{
    for (int i = 0; i < v.size(); i++)
    {
        cout << v[i] << " ";
    }
    cout << endl;
}

void print_table_header()
{
    cout << "|| " << setw(TWIDTH) << left << "Operation"
         << " || " << setw(TWIDTH) << "O/P"
         << " ||" << endl;
    cout << "|| " << setw(TWIDTH) << left << " "
         << " || " << setw(TWIDTH) << " "
         << " ||" << endl;
}

void print_table_row(vector<string> row)
{
    cout << "->" << setw(TWIDTH) << row[0] << "-> "
         << setw(TWIDTH) << row[1] << "->"
         << endl;
}
```

```
class Bits
{
public:
    int size;
    int_vec bits, original_bits;
    Bits(int _size)
    {
        size = _size;
        bits.resize(size);
        original_bits.resize(8);
    }
    void set_bit(int_vec _bits)
    {
        if (_bits.size() != size)
        {
            cout << "Error: Size of bits is not equal to size of the
object" << endl;
            return;
        }
        bits = _bits;
    }
    Bits *RLshift()
    {
        int start = size / 2;
        int temp = bits[start];
        for (int i = start; i < size - 1; i++)
        {
            bits[i] = bits[i + 1];
        }
        bits[size - 1] = temp;
        return this;
    }
    Bits *LLshift()
```

```
{
    int end = (size / 2) - 1;
    int temp = bits[0];
    for (int i = 0; i < end; i++)
    {
        bits[i] = bits[i + 1];
    }
    bits[end] = temp;
    return this;
}
Bits *Lshift()
{
    this
        ->RLshift()
        ->LLshift();
    print_table_row({"LeftShift", vector_to_string(bits)});
    return this;
}
Bits *P10()
{
    if (size != 10)
    {
        cout << "Error: P10 on non 10-bit data" << endl;
        return this;
    }
    int_vec temp(10);
    for (int i = 0; i < 10; i++)
    {
        temp[i] = bits[::P10[i] - 1];
    }
    bits = temp;
    print_table_row({"P10", vector_to_string(bits)});
    return this;
}
```

```
Bits *P8()
{
    int_vec temp(8);
    for (int i = 0; i < 8; i++)
    {
        temp[i] = bits[::P8[i] - 1];
    }
    bits = temp;
    size = 8;
    print_table_row({"P8", vector_to_string(bits)});
    return this;
}
Bits *P4()
{
    if (size != 8)
    {
        cout << "Error: Size of bits is not equal to 8 bit" << endl;
        return this;
    }
    int_vec temp(8);
    for (int i = 0; i < 8; i++)
    {
        if (i < 4)
            temp[i] = bits[i];
        else
            temp[i] = bits[::P4[i - 4] - 1 + 4];
    }
    bits = temp;
    print_table_row({"P4", vector_to_string(bits)});
    return this;
}
Bits *R_xor_with(Bits *b)
{
    if (size != 12)
```

```
    {
        cout << "Size of bits is not equal to 12 bit" << endl;
        return this;
    }
    int_vec temp(size);
    for (int i = 0; i < 12; i++)
    {
        if (i < 4)
            temp[i] = bits[i];
        else
            temp[i] = bits[i] == b->bits[i - 4] ? 0 : 1;
    }
    bits = temp;
    print_table_row({"R_xor_with", vector_to_string(bits)});
    return this;
}
Bits *R_xor_with_L()
{
    if (size != 8)
    {
        cout << "Size of bits not equal to 8" << endl;
        return this;
    }
    int_vec temp(size);
    for (int i = 0; i < 8; i++)
    {
        if (i < 4)
            temp[i] = bits[i];
        else
            temp[i] = bits[i] == bits[i - 4] ? 0 : 1;
    }
    bits = temp;
    for (int i = 0; i < 4; i++)
    {
```

```
            temp[i] = bits[i + 4];
            temp[i + 4] = original_bits[i + 4];
        }
        bits = temp;
        print_table_row({"R_xor_with_L", vector_to_string(bits)});
        return this;
    }
    Bits *IP()
    {
        if (size != 8)
        {
            cout << "Size of bits is not equal to 8 bit" << endl;
            return this;
        }
        int_vec temp(8);
        for (int i = 0; i < 8; i++)
        {
            temp[i] = bits[::IP[i] - 1];
        }
        original_bits = bits = temp;
        print_table_row({"IP", vector_to_string(bits)});
        return this;
    }
    Bits *INVERSE_IP()
    {
        if (size != 8)
        {
            cout << "Size of bits is not equal to 8 bit" << endl;
            return this;
        }
        int_vec temp(8);
        for (int i = 0; i < 8; i++)
        {
            temp[i] = bits[::INVERSE_IP[i] - 1];
```

```
    }
    bits = temp;
    print_table_row({"INVERSE_IP", vector_to_string(bits)});
    return this;
}
Bits *EP()
{
    int_vec temp(12);
    for (int i = 0; i < 12; i++)
    {
        if (i < 4)
            temp[i] = bits[i];
        else
            temp[i] = bits[::EP[i - 4] - 1 + 4];
    }
    size = 12;
    bits = temp;
    print_table_row({"EP", vector_to_string(bits)});
    return this;
}
Bits *shrink()
{
    if (size != 12)
    {
        cout << "Size of bits is not equal to 12 bit" << endl;
        return this;
    }
    int_vec temp(8);
    for (int i = 0; i < 4; i++)
    {
        temp[i] = bits[i];
    }
    // for s0
```

```
        int a = bits[4], b = bits[7], c = bits[5], d = bits[6]; // ab =
row , cd = col
        int row = a * 2 + b, col = c * 2 + d;
        int corresponding_value = S0[row][col];
        temp[4] = corresponding_value / 2;
        temp[5] = corresponding_value % 2;

        // for s1
        a = bits[8], b = bits[11], c = bits[9], d = bits[10]; // ab = row
, cd = col
        row = a * 2 + b;
        col = c * 2 + d;
        corresponding_value = S1[row][col];
        temp[6] = corresponding_value / 2;
        temp[7] = corresponding_value % 2;

        size = 8;
        bits = temp;
        print_table_row({"Shrink", vector_to_string(bits)});
        return this;
    }
    Bits *swap()
    {
        int_vec temp = bits;
        for (int i = 0; i < 4; i++)
        {
            bits[i] = temp[4 + i];
            bits[4 + i] = temp[i];
        }
        original_bits = bits;
        print_table_row({"Swap", vector_to_string(bits)});
        return this;
    }
    Bits *complex(Bits *key)
```

```
    {
        return this->EP()->R_xor_with(key)->shrink()->P4()-
>R_xor_with_L();
    }
    void print()
    {
        print_vector(bits);
    }
};

class SDES
{
public:
    Bits *plaintext;
    Bits *key, *key1, *key2;
    Bits *ciphertext;
    SDES()
    {
        plaintext = new Bits(8);
        key = new Bits(10);
        key1 = new Bits(10);
        key2 = new Bits(10);
        ciphertext = new Bits(8);
    }
    void set_key(int_vec _key)
    {
        key->set_bit(_key);
        key1->set_bit(_key);
    }
    void set_plaintext(int_vec _plaintext)
    {
        plaintext->set_bit(_plaintext);
    }
    void encrypt()
```

```
    {
        cout << endl
             << "KEY 1:" << endl;
        print_table_header();
        Bits *temp_plaintext = new Bits(8);
        temp_plaintext->set_bit(plaintext->bits);
        // generate key1 and key2
        key1->P10()
            ->Lshift();
        key2->set_bit(key1->bits); // copy key1 to key2
        key1->P8();
        cout << endl
             << "KEY 2:" << endl;
        print_table_header();
        key2
            ->Lshift()
            ->Lshift()
            ->P8();

        // now start encryption
        cout << endl
             << "PLain Text:" << endl;
        print_table_header();
        ciphertext = temp_plaintext->IP()->complex(key1)->swap()-
>complex(key2)->INVERSE_IP();
    }
    void decrypt()
    {
        cout << endl
             << "KEY 1:" << endl;
        print_table_header();
        Bits *temp_plaintext = new Bits(8);
        temp_plaintext->set_bit(plaintext->bits);
        key1->P10()
```

```
            ->Lshift();
        key2->set_bit(key1->bits);
        key1->P8();
        cout << endl
             << "KEY 2:" << endl;
        print_table_header();
        key2
            ->Lshift()
            ->Lshift()
            ->P8();
        cout << endl
             << "PLain Text:" << endl;
        print_table_header();
        ciphertext = temp_plaintext->IP()->complex(key2)->swap()-
>complex(key1)->INVERSE_IP();
    }
    void display(int is_decrypt = 0)
    {
        cout << endl
             << (is_decrypt ? "Ciphertext: " : "Plaintext: ");
        plaintext->print();
        cout << "Key: ";
        key->print();
        cout << "Sub Key1: ";
        key1->print();
        cout << "Sub Key2: ";
        key2->print();
        cout << (is_decrypt ? "Plaintext: " : "Ciphertext: ");
        ciphertext->print();
        cout << endl;
    }
};

int main()
```

```
{
    int choice = 0;
    int temp_int = 0;
    int_vec temp;
    SDES encrypt_object, decrypt_object;
    do
    {
        cout << "\n"
             << "1. Enter value of  Key"
             << "\n"
             << "2. SDES Encrypt"
             << "\n"
             << "3. SDES Decrypt"
             << "\n"
             << "4. End program"
             << "\n"
             << "Enter choice: " << endl;
        cin >> choice;
        cout << endl;
        switch (choice)
        {
        case 1:
            cout << "Enter key: ";
            temp.clear();
            for (int i = 0; i < 10; i++)
            {
                cin >> temp_int;
                temp.push_back(temp_int);
            }
            encrypt_object.set_key(temp);
            decrypt_object.set_key(temp);
            break;
        case 2:
            cout << "Enter plaintext: ";
```

```
            temp.clear();
            for (int i = 0; i < 8; i++)
            {
                cin >> temp_int;
                temp.push_back(temp_int);
            }
            encrypt_object.set_plaintext(temp);
            encrypt_object.encrypt();
            encrypt_object.display();
            break;
        case 3:
            cout << "Enter ciphertext: ";
            temp.clear();
            for (int i = 0; i < 8; i++)
            {
                cin >> temp_int;
                temp.push_back(temp_int);
            }
            decrypt_object.set_plaintext(temp);
            decrypt_object.decrypt();
            decrypt_object.display(1);
            break;
        case 4:
            break;
        default:
            cout << endl
                 << "Invalid Choice!";
        }
        cout << endl;
    } while (choice != 4);
    cout << endl;
    return 0;
}
```

# Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

cd "/Users/ani/Desktop/Tri-8/IS LAbs/p1/" && g++ --std=c++17 a.cpp -o a && "/Users/ani/Desktop/Tri-8/IS LAbs/p1/"a
[3] 82436
ani@Aniruddhas-MacBook-Pro IS LAbs % cd "/Users/ani/Desktop/Tri-8/IS LAbs/p1/" && g++ --std=c++17 a.cpp -o a && "/Use
Enter the string you want to Encrypt or Decrypt : MITWPU
Enter Key : 3
1. Encrypt
2. Decrypt

Enter your choice
1

Encrypted str is: plwzsx

Do you want to continue ?? (1->yes)1
Enter the string you want to Encrypt or Decrypt : plwzsx
Enter Key : 3
1. Encrypt
2. Decrypt

Enter your choice
2

Decrypted str is: MITWPU

Do you want to continue ?? (1->yes)0
ani@Aniruddhas-MacBook-Pro p1 % cd "/Users/ani/Desktop/Tri-8/IS LAbs/IS Assign 2vDES Final/" && g++ --std=c++17 copy1
S Assign 2vDES Final/"copy1

1. Enter value of  Key
2. SDES Encrypt
3. SDES Decrypt
4. End program
Enter choice:
1

Enter key: 1 0 1 0 0 0 0 0 1 0


1. Enter value of  Key      _____

1. Enter value of  Key
2. SDES Encrypt
3. SDES Decrypt
4. End program
Enter choice:
2

Enter plaintext: 1 1 1 1 0 0 1 1

KEY 1:
|| Operation                || O/P                      ||
||                          ||                          ||
->P10                      -> 1 0 0 0 0 0 1 1 0 0      ->
->LeftShift                -> 0 0 0 0 1 1 1 0 0 0      ->
->P8                       -> 1 0 1 0 0 1 0 0          ->

KEY 2:
|| Operation                || O/P                      ||
||                          ||                          ||
->LeftShift                -> 0 0 0 1 0 1 0 0 0 1      ->
->LeftShift                -> 0 0 1 0 0 0 0 0 1 1      ->
->P8                       -> 0 1 0 0 0 0 1 1          ->

PLain Text:
|| Operation                || O/P                      ||
||                          ||                          ||
->IP                       -> 1 0 1 1 1 1 0 1          ->
->EP                       -> 1 0 1 1 1 1 1 0 1 0 1 1  ->
->R_xor_with               -> 1 0 1 1 0 1 0 0 1 1 1 1  ->
->Shrink                   -> 1 0 1 1 1 1 1 1          ->
->P4                       -> 1 0 1 1 1 1 1 1          ->
->R_xor_with_L             -> 0 1 0 0 1 1 0 1          ->
->Swap                     -> 1 1 0 1 0 1 0 0          ->
->EP                       -> 1 1 0 1 0 0 1 0 1 0 0 0  ->
->R_xor_with               -> 1 1 0 1 0 1 1 0 1 0 1 1  ->
->Shrink                   -> 1 1 0 1 1 0 0 1          ->
->P4                       -> 1 1 0 1 0 1 0 1          ->
->R_xor_with_L             -> 1 0 0 0 0 1 0 0          ->
->INVERSE_IP               -> 0 1 0 0 0 0 0 1          ->
```

```
                          ______
->Shrink                   -> 1 1 0 1 1 0 0 1          ->
->P4                       -> 1 1 0 1 0 1 0 1          ->
->R_xor_with_L             -> 1 0 0 0 0 1 0 0          ->
->INVERSE_IP               -> 0 1 0 0 0 0 0 1          ->

Plaintext: 1 1 1 1 1 0 0 1 1
Key: 1 0 1 0 0 0 0 0 1 0
Sub Key1: 1 0 1 0 0 1 0 0
Sub Key2: 0 1 0 0 0 0 1 1
Ciphertext: 0 1 0 0 0 0 0 1



1. Enter value of  Key
2. SDES Encrypt
3. SDES Decrypt
4. End program
Enter choice:
3

Enter ciphertext: 0 1 0 0 0 0 0 1

KEY 1:
|| Operation                || O/P                       ||
||                           ||                           ||
->P10                      -> 1 0 0 0 0 0 1 1 0 0      ->
->LeftShift                -> 0 0 0 0 1 1 1 0 0 0      ->
->P8                       -> 1 0 1 0 0 1 0 0          ->

KEY 2:
|| Operation                || O/P                       ||
||                           ||                           ||
->LeftShift                -> 0 0 0 1 0 1 0 0 0 1      ->
->LeftShift                -> 0 0 1 0 0 0 0 0 1 1      ->
->P8                       -> 0 1 0 0 0 0 1 1          ->

PLain Text:
|| Operation                || O/P                       ||
||                           ||                           ||
```

```
|| Operation                || O/P                       ||
||                           ||                           ||
->LeftShift                -> 0 0 0 1 0 1 0 0 0 1      ->
->LeftShift                -> 0 0 1 0 0 0 0 0 1 1      ->
->P8                       -> 0 1 0 0 0 0 1 1          ->

PLain Text:
|| Operation                || O/P                       ||
||                           ||                           ||
->IP                       -> 1 0 0 0 0 1 0 0          ->
->EP                       -> 1 0 0 0 0 0 1 0 1 0 0 0  ->
->R_xor_with               -> 1 0 0 0 0 1 1 0 1 0 1 1  ->
->Shrink                   -> 1 0 0 0 1 0 0 1          ->
->P4                       -> 1 0 0 0 0 1 0 1          ->
->R_xor_with_L             -> 1 1 0 1 0 1 0 0          ->
->Swap                     -> 0 1 0 0 1 1 0 1          ->
->EP                       -> 0 1 0 0 1 1 1 0 1 0 1 1  ->
->R_xor_with               -> 0 1 0 0 0 1 0 0 1 1 1 1  ->
->Shrink                   -> 0 1 0 0 1 1 1 1          ->
->P4                       -> 0 1 0 0 1 1 1 1          ->
->R_xor_with_L             -> 1 0 1 1 1 1 0 1          ->
->INVERSE_IP               -> 1 1 1 1 0 0 1 1          ->

Ciphertext: 0 1 0 0 0 0 0 1
Key: 1 0 1 0 0 0 0 0 1 0
Sub Key1: 1 0 1 0 0 1 0 0
Sub Key2: 0 1 0 0 0 0 1 1
Plaintext: 1 1 1 1 1 0 0 1 1



1. Enter value of  Key
2. SDES Encrypt
3. SDES Decrypt
4. End program
Enter choice:
4
```