BCSE309P – Cryptography and Network Security Lab

Exercise 3

Slot: L37+L38
Date: 30-01-2024

Name: *Deepansh Parmar*
Reg No: *21BCE1812*

DES Encryption

Consider a sender and receiver who need to exchange data confidentially using symmetric encryption. Write program that implements DES encryption and decryption using a 64 bit key size and 64 bit block size.

Code:

Sender side:

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<vector>
#include <bitset>
#include <sstream>

using namespace std;

vector<string> subkeys;
string subkey1 = "", subkey2 = "";

string binaryToHex(const string& binaryString) {
    bitset<128> binaryValue(binaryString); // Assuming a maximum length of 128 bits
```

```
    // Convert binary to hexadecimal
    stringstream hexStream;
    hexStream << hex << binaryValue.to_ullong();

    return hexStream.str();
}

string convertDecimalToBinary(int decimal)
{
    string binary;
    while(decimal != 0) {
        binary = (decimal % 2 == 0 ? "0" : "1") + binary;
        decimal = decimal/2;
    }
    while(binary.length() < 4){
        binary = "0" + binary;
    }
    return binary;
}

int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for(int i = size-1; i >= 0; i--)
    {
        if(binary[i] == '1'){
            decimal += pow(2, counter);
        }
        counter++;
    }
    return decimal;
}

string HexBin(string hexdec)
{
     //Skips "0x" if present at beggining of Hex string
    size_t i = (hexdec[1] == 'x' || hexdec[1] == 'X')? 2 : 0;
    string hextobin = "";

    while (hexdec[i]) {
        switch (hexdec[i]) {
        case '0':
```

```
    hextobin += "0000";
    break;
case '1':
    hextobin += "0001";
    break;
case '2':
    hextobin += "0010";
    break;
case '3':
    hextobin += "0011";
    break;
case '4':
    hextobin += "0100";
    break;
case '5':
    hextobin += "0101";
    break;
case '6':
    hextobin += "0110";
    break;
case '7':
    hextobin += "0111";
    break;
case '8':
    hextobin += "1000";
    break;
case '9':
    hextobin += "1001";
    break;
case 'A':
case 'a':
    hextobin += "1010";
    break;
case 'B':
case 'b':
    hextobin += "1011";
    break;
case 'C':
case 'c':
    hextobin += "1100";
    break;
case 'D':
case 'd':
    hextobin += "1101";
```

```
            break;
        case 'E':
        case 'e':
            hextobin += "1110";
            break;
        case 'F':
        case 'f':
            hextobin += "1111";
            break;
        case '.':
            hextobin += ".";
            break;
        default:
            cout << "\nInvalid hexadecimal digit "<< hexdec[i];
        }
        i++;
    }
    return hextobin;
}

string permute(const string& input, const int* permutation, int size) {
    string output;
    for (int i = 0; i < size; ++i) {
        output += input[permutation[i] - 1];
    }
    return output;
}


string Xor(string a, string b)
{
    string result = "";
    int size = b.size();
    for(int i = 0; i < size; i++)
    {
        if(a[i] != b[i])
        {
            result += "1";
        }
        else
        {
            result += "0";
        }
    }
```

```
    return result;
}

/*
int toDecimal(string bin)
{
    int dec = 0, c = 0;
    for(int i = bin.size()-1; i >= 0; --i)
    {
        dec += atoi(bin[i]) * pow(2,c++);
    }

    return dec;
}
*/
string shift(string half, int index)
{
        string str = half.substr(index, 28-index) + half.substr(0, index);
        return str;
}

void printSubKeys(string key1){
    string key = HexBin(key1);
        int perm56[56] = {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};
        int perm48[48] =
{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};
        string perm_key = "";
        string subkey1_left, subkey1_right, subkey2_left, subkey2_right, left_half,
right_half, sk1, sk2;

        for(int i = 0; i < 56; ++i)
        {
                perm_key += key[perm56[i] - 1];
                if(i < 28)
                {
                        left_half += perm_key[i];
                }
                else
                {
                        right_half += perm_key[i];
                }
```

```
        }
        //vector<string> subkeys;
        int lcs[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
        for(int i = 0; i < 16; ++i)
        {
                string skey1 = shift(left_half, lcs[i]);
                string skey2 = shift(right_half, lcs[i]);
                left_half = skey1;
                right_half = skey2;
                string k = skey1 + skey2;
                string st = "";
                for(int j = 0; j < 48; ++j)
           {
              st += k[perm48[j] - 1];
           }
                subkeys.push_back(st);
                st = "";
        }
        string st = "";

        for(int i = 0; i < 16; ++i)
        {
                cout<<i+1<<" "<<subkeys[i]<<endl;
        }

}

string encrypt(string pt_){

        string pt = HexBin(pt_);
    int initial_perm[] = {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};

    // Expansion D-box Table
    int exp_d[] = {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};

   // Straight Permutation Table
   int per[] = {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};
   int inverse_permutation[64]= {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};

   // S-box Table
   int substition_boxes[8][4][16] = {
      {
         {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
         {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
         {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
         {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
      },
      {
         {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
         {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
         {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
         {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
      },
      {
         {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}
      },
      {
         {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}
      },
      {
         {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}
    },
    {
        {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}
    },
    {
        {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}
    },
    {
        {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}
    }
};

//Final Permutation Table
int final_perm[] = {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};
string perm = "";
for(int i = 0; i < 64; i++){
    perm += pt[initial_perm[i]-1];
}
string left = perm.substr(0, 32);
string right = perm.substr(32, 32);
for(int i=0; i<16; i++) {
    string right_expanded = "";
    for(int i = 0; i < 48; i++) {
        right_expanded += right[exp_d[i]-1];
    }
    string xored = Xor(subkeys[i], right_expanded);
```

```
        string res = "";
        for(int i=0;i<8; i++){
            string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
            int row = convertBinaryToDecimal(row1);
            string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 + 2,1) +
xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);
            int col = convertBinaryToDecimal(col1);
            int val = substition_boxes[i][row][col];
            res += convertDecimalToBinary(val);
        }
        string perm2 ="";
        for(int i = 0; i < 32; i++){
            perm2 += res[per[i]-1];
        }
        xored = Xor(perm2, left);
        left = xored;
        if(i < 15){
            string temp = right;
            right = xored;
            left = temp;
        }
    }
    string combined_text = left + right;
    string ciphertext ="";
    for(int i = 0; i < 64; i++){
        ciphertext+= combined_text[inverse_permutation[i]-1];
    }
    return binaryToHex(ciphertext);
}


int main() {
    string key = "0f1571c947d9e859";
    cout<<"Key is: "<<key<<endl;
    // Create socket
    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1) {
        std::cerr << "Error creating socket" << std::endl;
        return -1;
    }

    // Connect to the server
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
```

```
    serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1"); // Server IP address
    serverAddress.sin_port = htons(8080);

    if (connect(clientSocket, reinterpret_cast<struct sockaddr*>(&serverAddress),
sizeof(serverAddress)) == -1) {
        std::cerr << "Error connecting to the receiver" << std::endl;
        close(clientSocket);
        return -1;
    }

    std::cout << "Connected to the receiver." << std::endl;

    // Send data to the server
    //char* message = "1010010110";
    string plaintext = "02468aceeca86420";
    cout<<"Plaintext is: "<<plaintext<<endl;
    printSubKeys(key);
    string ct = encrypt(plaintext);
    cout<<"Cipher text is: "<<ct<<endl;
    const char * message = ct.c_str();
    ssize_t bytesSent = send(clientSocket, message, strlen(message), 0);

    if (bytesSent == -1) {
        std::cerr << "Error sending cipher text" << std::endl;
    } else {
        std::cout << "Sent cipher text to the receiver.\n" << std::endl;
    }

    // Close the socket
    close(clientSocket);

    return 0;
}
```

Receiver Side:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#include<cmath>
#include<vector>
#include <bitset>
#include <sstream>

using namespace std;

#define BUF_SIZE 2048
#define PORT 8080

vector<string> subkeys;

string binaryToHex(const string& binaryString) {
    bitset<128> binaryValue(binaryString); // Assuming a maximum length of 128 bits

    // Convert binary to hexadecimal
    stringstream hexStream;
    hexStream << hex << binaryValue.to_ullong();

    return hexStream.str();
}
string HexBin(string hexdec)
{
     //Skips "0x" if present at beggining of Hex string
    size_t i = (hexdec[1] == 'x' || hexdec[1] == 'X')? 2 : 0;
    string hextobin = "";

    while (hexdec[i]) {
        switch (hexdec[i]) {
        case '0':
            hextobin += "0000";
            break;
        case '1':
            hextobin += "0001";
            break;
        case '2':
            hextobin += "0010";
            break;
        case '3':
            hextobin += "0011";
            break;
        case '4':
            hextobin += "0100";
            break;
```

```
case '5':
   hextobin += "0101";
   break;
case '6':
   hextobin += "0110";
   break;
case '7':
   hextobin += "0111";
   break;
case '8':
   hextobin += "1000";
   break;
case '9':
   hextobin += "1001";
   break;
case 'A':
case 'a':
   hextobin += "1010";
   break;
case 'B':
case 'b':
   hextobin += "1011";
   break;
case 'C':
case 'c':
   hextobin += "1100";
   break;
case 'D':
case 'd':
   hextobin += "1101";
   break;
case 'E':
case 'e':
   hextobin += "1110";
   break;
case 'F':
case 'f':
   hextobin += "1111";
   break;
case '.':
   hextobin += ".";
   break;
default:
   cout << "\nInvalid hexadecimal digit "<< hexdec[i];
```

```
        }
        i++;
    }
    return hextobin;
}


string convertDecimalToBinary(int decimal)
{
        string binary;
    while(decimal != 0) {
                binary = (decimal % 2 == 0 ? "0" : "1") + binary;
                decimal = decimal/2;
        }
        while(binary.length() < 4){
                binary = "0" + binary;
        }
    return binary;
}
// Function to convert a number in binary to decimal
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
        int counter = 0;
        int size = binary.length();
        for(int i = size-1; i >= 0; i--)
        {
        if(binary[i] == '1'){
        decimal += pow(2, counter);
        }
    counter++;
        }
        return decimal;
}

string shift(string half, int index)
{
        string str = half.substr(index, 28-index) + half.substr(0, index);
        return str;
}

string Xor(string a, string b){
        string result = "";
        int size = b.size();
```

```
        for(int i = 0; i < size; i++){
                if(a[i] != b[i]){
                        result += "1";
                }
                else{
                        result += "0";
                }
        }
        return result;
}

void printSubKeys(string key1){
   string key = HexBin(key1);
        int perm56[56] = {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};
        int perm48[48] =
{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};
        string perm_key = "";
        string subkey1_left, subkey1_right, subkey2_left, subkey2_right, left_half,
right_half, sk1, sk2;

        for(int i = 0; i < 56; ++i)
        {
                perm_key += key[perm56[i] - 1];
                if(i < 28)
                {
                        left_half += perm_key[i];
                }
                else
                {
                        right_half += perm_key[i];
                }
        }
        //vector<string> subkeys;
        int lcs[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
        for(int i = 0; i < 16; ++i)
        {
                string skey1 = shift(left_half, lcs[i]);
                string skey2 = shift(right_half, lcs[i]);
                left_half = skey1;
                right_half = skey2;
                string k = skey1 + skey2;
```

```
                string st = "";
                for(int j = 0; j < 48; ++j)
            {
               st += k[perm48[j] - 1];
            }
                subkeys.push_back(st);
                st = "";
        }
}

string decrypt(string pt_){

        string pt = HexBin(pt_);
    int inverse_permutation[] = {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, //initial_perm
              59, 51, 43, 35, 27, 19, 11, 3,
              61, 53, 45, 37, 29, 21, 13, 5,
              63, 55, 47, 39, 31, 23, 15, 7};

    // Expansion D-box Table
    int exp_d[] = {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};

    // Straight Permutation Table
    int per[] = {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};
    int initial_perm[64]= {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};

    // S-box Table
    int substition_boxes[8][4][16] = {
```

```
{
   {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
   {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
   {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
   {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
},
{
   {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
   {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
   {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
   {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
},
{
   {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}
},
{
   {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}
},
{
   {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}
},
{
   {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}
},
{
   {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}
},
{
   {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}
    }
};

//Final Permutation Table
int final_perm[] = {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};
string perm = "";
for(int i = 0; i < 64; i++){
    perm += pt[initial_perm[i]-1];
}
string left = perm.substr(0, 32);
string right = perm.substr(32, 32);
for(int i=0; i<16; i++) {
    string right_expanded = "";
    for(int i = 0; i < 48; i++) {
        right_expanded += right[exp_d[i]-1];
    }
    string xored = Xor(subkeys[i], right_expanded);
    string res = "";
    for(int i=0;i<8; i++){
        string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
        int row = convertBinaryToDecimal(row1);
        string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 + 2,1) +
xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);
        int col = convertBinaryToDecimal(col1);
        int val = substition_boxes[i][row][col];
        res += convertDecimalToBinary(val);
    }
    string perm2 ="";
    for(int i = 0; i < 32; i++){
        perm2 += res[per[i]-1];
    }
    xored = Xor(perm2, left);
    left = xored;
    if(i < 15){
```

```
            string temp = right;
            right = xored;
            left = temp;
        }
    }
    string combined_text = left + right;
    string ciphertext ="";
    for(int i = 0; i < 64; i++){
        ciphertext+= combined_text[inverse_permutation[i]-1];
    }
    return binaryToHex(ciphertext);
}

int main() {


    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        std::cerr << "Error creating socket" << std::endl;
        return -1;
    }

    // Bind the socket to an address and port
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(8080);

    if (bind(serverSocket, reinterpret_cast<struct sockaddr*>(&serverAddress),
sizeof(serverAddress)) == -1) {
        std::cerr << "Error binding socket" << std::endl;
        close(serverSocket);
        return -1;
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        std::cerr << "Error listening for connections" << std::endl;
        close(serverSocket);
        return -1;
    }

    std::cout << "listening on port 8080..." << std::endl;
```

```
    // Accept a connection
    sockaddr_in clientAddress;
    socklen_t clientAddressLength = sizeof(clientAddress);
    int clientSocket = accept(serverSocket, reinterpret_cast<struct
sockaddr*>(&clientAddress), &clientAddressLength);

    if (clientSocket == -1) {
        std::cerr << "Error accepting connection" << std::endl;
        close(serverSocket);
        return -1;
    }

    //std::cout << "Connection accepted from " << inet_ntoa(clientAddress.sin_addr)
<< ":" << ntohs(clientAddress.sin_port) << std::endl;

    // Receive data from the client
    char buffer[1024];
    string ptext = "02468aceeca86420";
    ssize_t bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

    if (bytesRead <= 0) {
        std::cerr << "Error receiving cipher text" << std::endl;
    } else {
        buffer[bytesRead] = '\0';
        std::cout << "Received cipher text from sender: " << buffer << std::endl;
    }
    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    string ct = buffer;
    string key = "0f1571c947d9e859";
    printSubKeys(key);

    int i = 15;
          int j = 0;
          while(i > j)
          {
                    string temp = subkeys[i];
                    subkeys[i] = subkeys[j];
                    subkeys[j] = temp;
                    i--;
                    j++;
          }
```

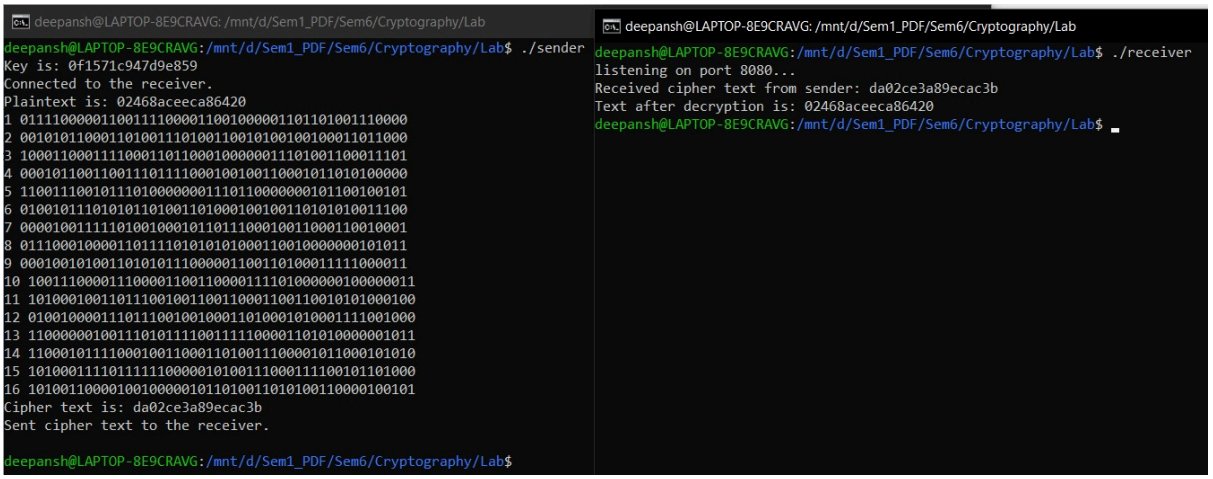
```
    string pt = decrypt(ct);

    cout<<"Text after decryption is: "<<ptext<<endl;



    return 0;
}
```

Output:



Result:

DES was successfully implemented using client server programming in C++.