

Final Year B. Tech., Sem VII 2022-23

Cryptography And Network Security

PRN/ Roll No: 2020BTECS00206

Full Name: SAYALI YOGESH DESAI

Batch: B4

Assignment No. 6

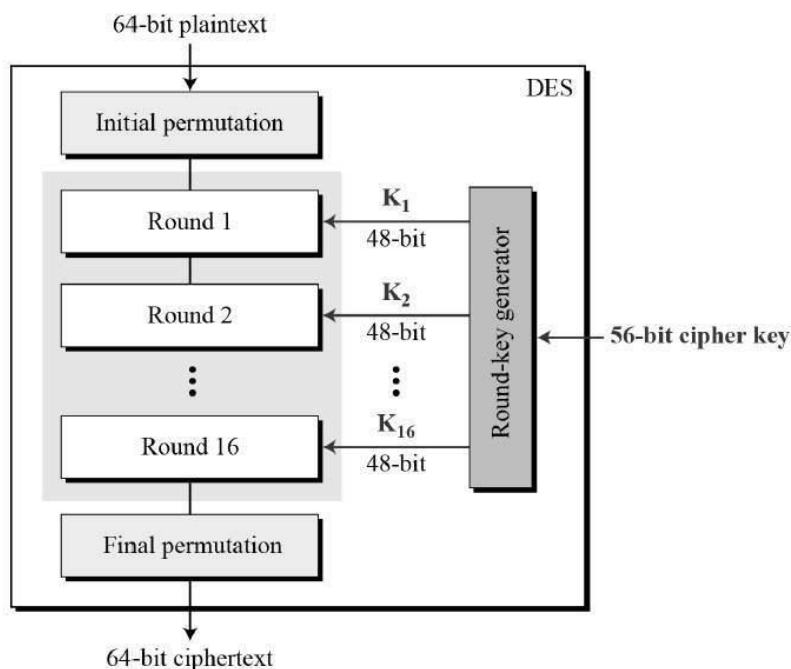
1. Aim:

To encrypt given plain text using the DES algorithm

2. Theory:

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration –

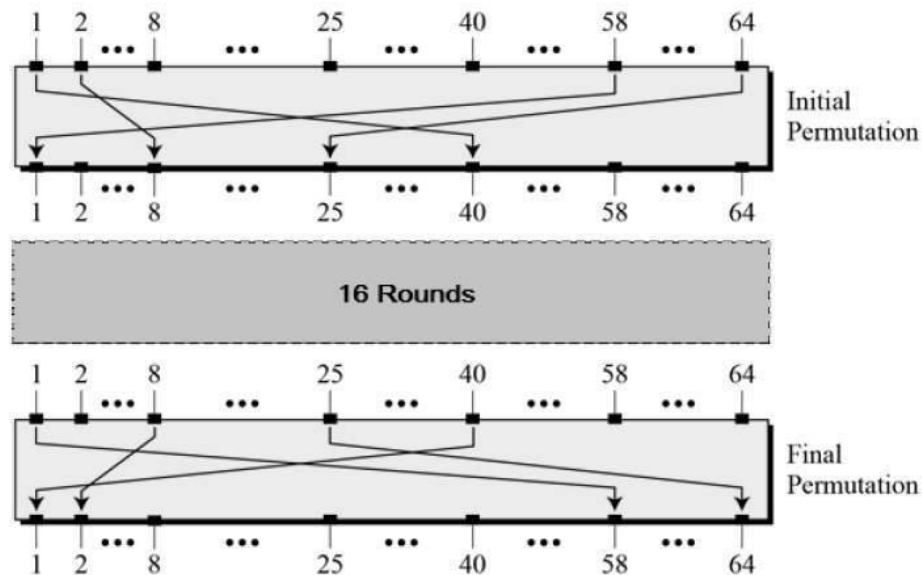


Since DES is based on the Feistel Cipher, all that is required to specify DES is –

- Round function
- Key schedule
- Any additional processing – Initial and final permutation

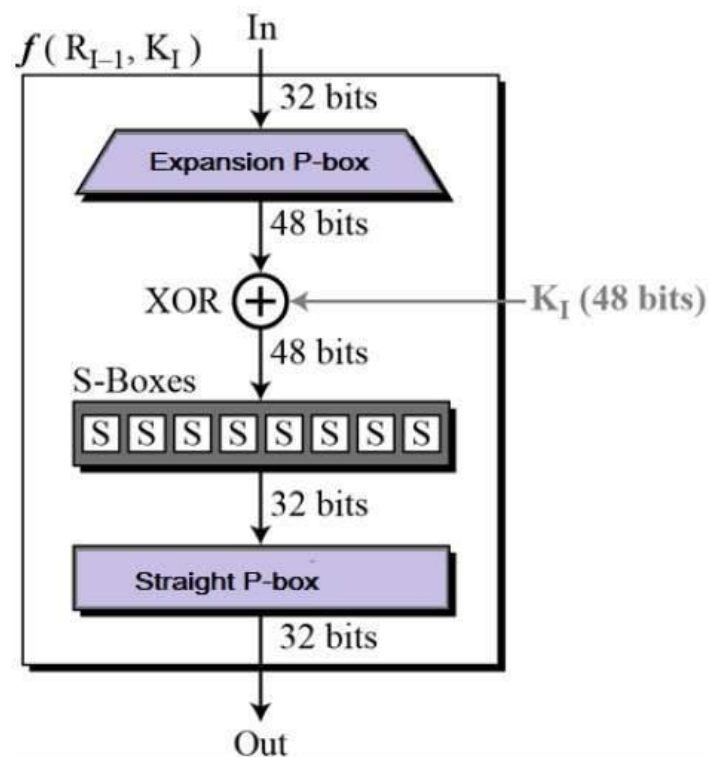
Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows –

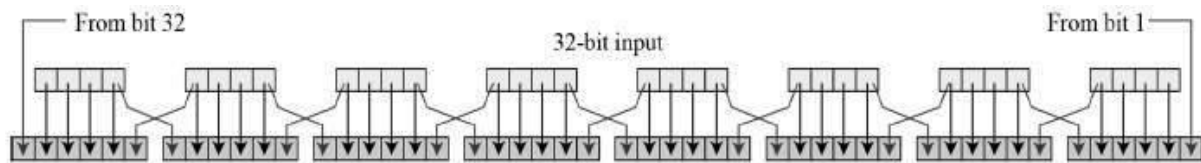


Round Function

The heart of this cipher is the DES function, f . The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



Expansion Permutation Box – Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –

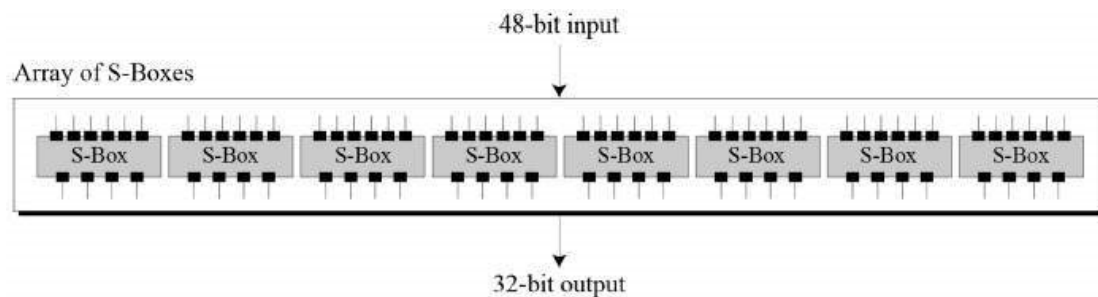


The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

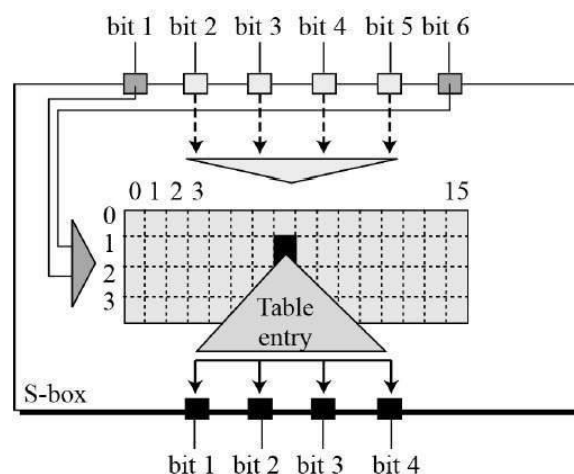
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

XOR (Whitener). – After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.

Substitution Boxes. – The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –



The S-box rule is illustrated below –



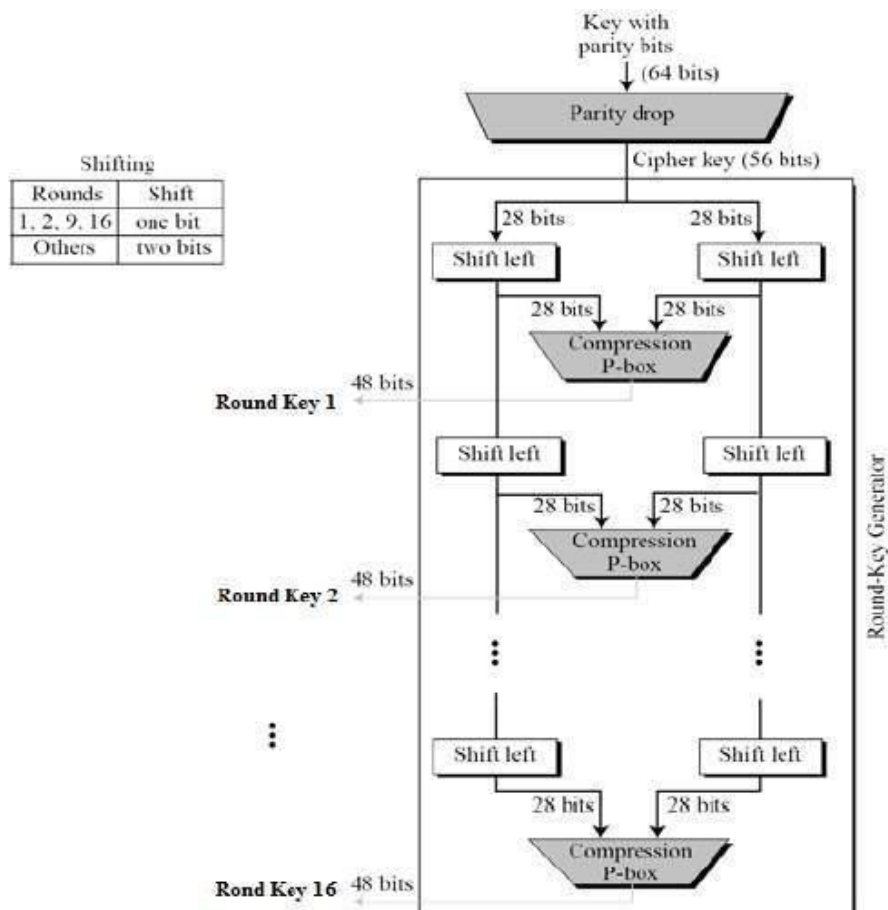
There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.

Straight Permutation – The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

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

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration –



The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

3. Code:

```
#include <bits/stdc++.h>

using namespace std;

string hexToBin(string s) {
    unordered_map<char, string> mp;
    mp['0'] = "0000";
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
    mp['8'] = "1000";
    mp['9'] = "1001";
    mp['A'] = "1010";
    mp['B'] = "1011";
    mp['C'] = "1100";
    mp['D'] = "1101";
    mp['E'] = "1110";
    mp['F'] = "1111";
    stringstream bin;
    for (int i = 0; i < s.size(); i++) {
        bin << mp[s[i]];
    }
    return bin.str();
}
```

```
string binToHex(string s) {  
    unordered_map<string, string> mp;  
    mp["0000"] = "0";  
    mp["0001"] = "1";  
    mp["0010"] = "2";  
    mp["0011"] = "3";  
    mp["0100"] = "4";  
    mp["0101"] = "5";  
    mp["0110"] = "6";  
    mp["0111"] = "7";  
    mp["1000"] = "8";  
    mp["1001"] = "9";  
    mp["1010"] = "A";  
    mp["1011"] = "B";  
    mp["1100"] = "C";  
    mp["1101"] = "D";  
    mp["1110"] = "E";  
    mp["1111"] = "F";  
    stringstream hex;  
    for (int i = 0; i < s.length(); i += 4) {  
        string ch = s.substr(i, 4);  
        hex << mp[ch];  
    }  
    return hex.str();  
}  
  
string permute(string k, int *arr, int n) {  
    stringstream per;
```

```
    for (int i = 0; i < n; i++) {  
        per << k[arr[i] - 1];  
    }  
    return per.str();  
}  
  
string shiftLeft(string k, int shifts) {  
    string s = "";  
    for (int i = 0; i < shifts; i++) {  
        for (int j = 1; j < 28; j++) {  
            s += k[j];  
        }  
        s += k[0];  
        k = s;  
        s = "";  
    }  
    return k;  
}  
  
string XOR(string a, string b) {  
    stringstream ans;  
    for (int i = 0; i < a.size(); i++) {  
        if (a[i] == b[i]) {  
            ans << "0";  
        } else {  
            ans << "1";  
        }  
    }  
    return ans.str();  
}
```

```

}

string encrypt(string plain, vector<string> rkb, vector<string> rk) {
    // Hexadecimal to binary
    plain = hexToBin(plain);

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

    // Initial Permutation
    plain = permute(plain, initial_perm, 64);

    cout << "After initial permutation: " << binToHex(plain) << endl;

    // Splitting
    string left = plain.substr(0, 32);
    string right = plain.substr(32, 32);

    cout << "After splitting: L0=" << binToHex(left)
         << " R0=" << binToHex(right) << endl;

    // Expansion D-box Table
    int exp_d[48] = {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
};

// S-box Table
int s[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
}
};

```

// Straight Permutation Table

```

int per[32] = { 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
    };

    cout << endl;

    for (int i = 0; i < 16; i++) {
        // Expansion D-box

        string right_expanded = permute(right, exp_d, 48);

        // XOR RoundKey[i] and right_expanded

        string x = XOR(rkb[i], right_expanded);

        // S-boxes

        string op = "";

        for (int i = 0; i < 8; i++) {

            int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');

            int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] - '0') + 2 * int(x[i * 6 + 3] - '0')
+ int(x[i * 6 + 4] - '0');

            int val = s[i][row][col];

            op += char(val / 8 + '0');

            val = val % 8;

            op += char(val / 4 + '0');

            val = val % 4;

            op += char(val / 2 + '0');

            val = val % 2;

            op += char(val + '0');

        }

        // Straight D-box

        op = permute(op, per, 32);

        // XOR left and op

        x = XOR(op, left);

```

```

    left = x;

    // Swapper
    if (i != 15) {
        swap(left, right);
    }

    cout << "Round " << i + 1 << " " << binToHex(left) << " "
        << binToHex(right) << " " << rk[i] << endl;
}

// Combination
string combine = left + right;

// Final Permutation Table
int final_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
};

// Final Permutation
string cipher = binToHex(permute(combine, final_perm, 64));

return cipher;
}

int main() {

```

```
string plain, key;

// plain = "This is a test text";

// key = "this is a test";

// Key Generation

cout << "Enter the plain text: ";

getline(cin, plain);

cout << "Enter the key: ";

getline(cin, key);

// Hex to binary

key = hexToBin(key);

// Parity bit drop table

int keyp[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
               };

// getting 56 bit key from 64 bit using the parity bits

key = permute(key, keyp, 56); // key without parity
```

```
// Number of bit shifts
```

```
int shift_table[16] = { 1, 1, 2, 2,
                        2, 2, 2, 2,
                        1, 2, 2, 2,
                        2, 2, 2, 1
                        };
```

```
// Key- Compression Table
```

```
int key_comp[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
                     };
```

```
// Splitting
```

```
string left = key.substr(0, 28);
```

```
string right = key.substr(28, 28);
```

```
vector<string> rkb; // rkb for RoundKeys in binary
```

```
vector<string> rk; // rk for RoundKeys in hexadecimal
```

```
for (int i = 0; i < 16; i++) {
```

```
    // Shifting
```

```
left = shiftLeft(left, shift_table[i]);
right = shiftLeft(right, shift_table[i]);

// Combining
string combine = left + right;

// Key Compression
string RoundKey = permute(combine, key_comp, 48);

rkb.push_back(RoundKey);
rk.push_back(binToHex(RoundKey));
}
cout << "\nEncryption:\n\n";
string cipher = encrypt(plain, rkb, rk);
cout << "\nCipher Text: " << cipher << endl;
cout << "\nDecryption\n\n";
reverse(rkb.begin(), rkb.end());
reverse(rk.begin(), rk.end());
string text = encrypt(cipher, rkb, rk);
cout << "\nPlain Text: " << text << endl;
}
```

4. Output:

```

PS D:\Walchand\7 Semester\Crypto\Assignment 6> g++ .\des.cpp
PS D:\Walchand\7 Semester\Crypto\Assignment 6> ./a.exe
Enter the plain text: cryptography lab
Enter the key: cns

Encryption:

After initial permutation:
After splitting: L0= R0=

Round 1  FFFFFFFF
Round 2  FFFFFFFF FFFFFFFF
Round 3  FFFFFFFF C7248636
Round 4  C7248636 C7240634
Round 5  C7240634 FFFF7FFD
Round 6  FFFF7FFD FBFFFEFB
Round 7  FBFFFEFB C72C8626
Round 8  C72C8626 C7240630
Round 9  C7240630 FFF77FED
Round 10 FFF77FED FF7FFDF9
Round 11 FF7FFDF9 472C9626
Round 12 472C9626 C7A40432
Round 13 C7A40432 7BF66EED
Round 14 7BF66EED BF7FBDE9
Round 15 BF7FBDE9 432D9726
Round 16 87A44422 432D9726

Cipher Text: E8CB7E2008338458

```

```

Decryption

After initial permutation: 87A44422432D9726
After splitting: L0=87A44422 R0=432D9726

Round 1  432D9726 BF7FBDE9
Round 2  BF7FBDE9 7BF66EED
Round 3  7BF66EED C7A40432
Round 4  C7A40432 472C9626
Round 5  472C9626 FF7FFDF9
Round 6  FF7FFDF9 FFF77FED
Round 7  FFF77FED C7240630
Round 8  C7240630 C72C8626
Round 9  C72C8626 FBFFFEFB
Round 10 FBFFFEFB FFFF7FFD
Round 11 FFFF7FFD C7240634
Round 12 C7240634 C7248636
Round 13 C7248636 FFFFFFFF
Round 14 FFFFFFFF FFFFFFFF
Round 15 FFFFFFFF 47641634
Round 16 C7240634 47641634

Plain Text: C0CCFF00B33E040
PS D:\Walchand\7 Semester\Crypto\Assignment 6> 

```

5. Conclusion:

Successfully encrypted the given plain text using DES Algorithm.