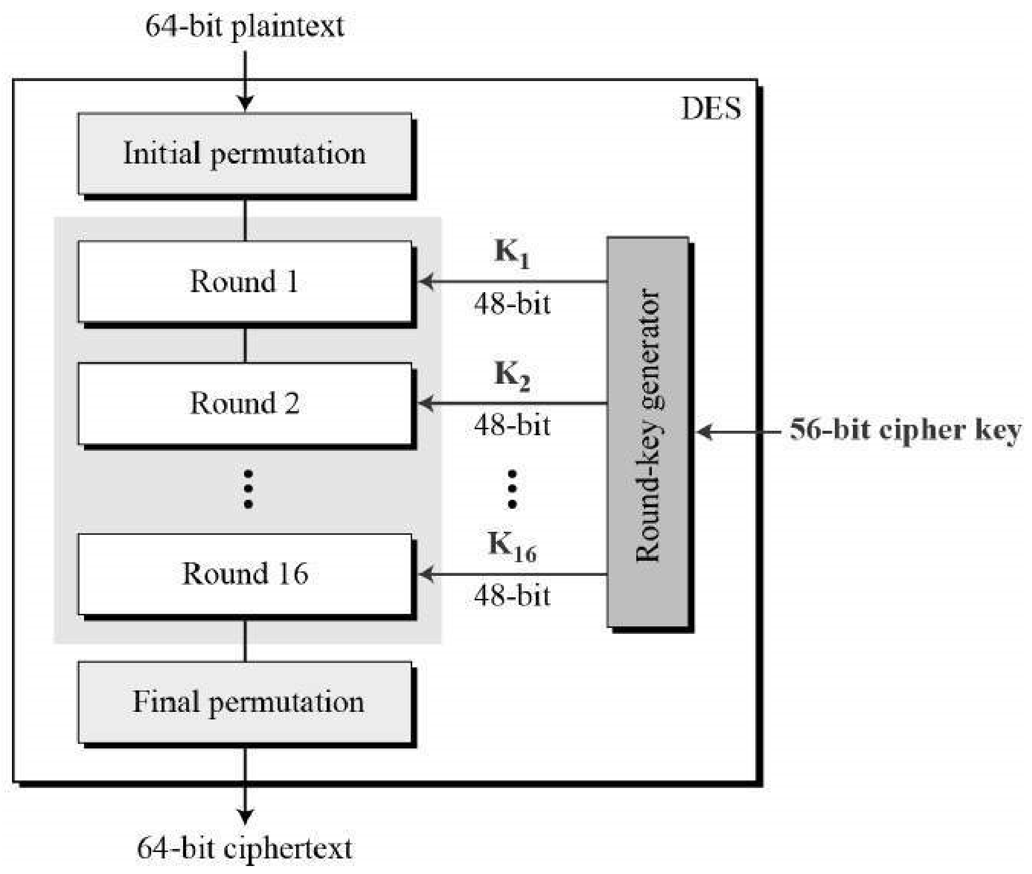
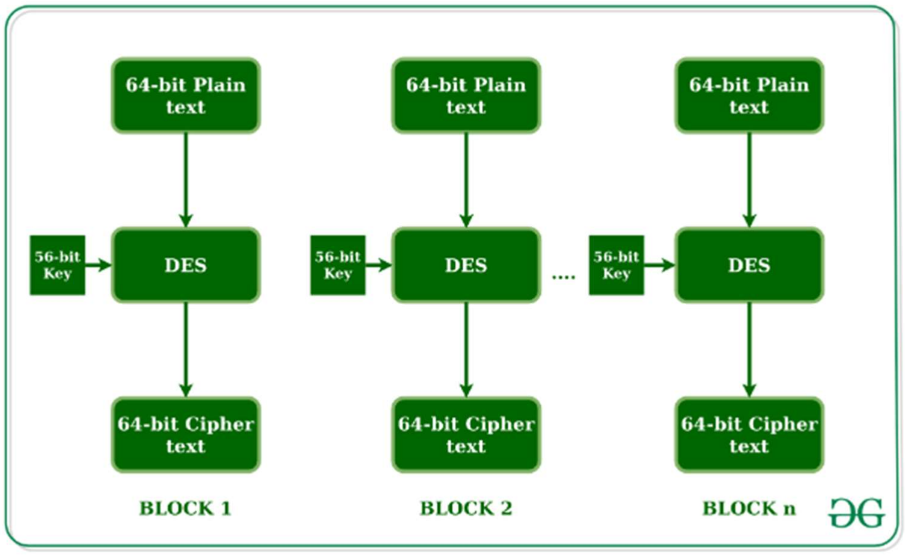
Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 6

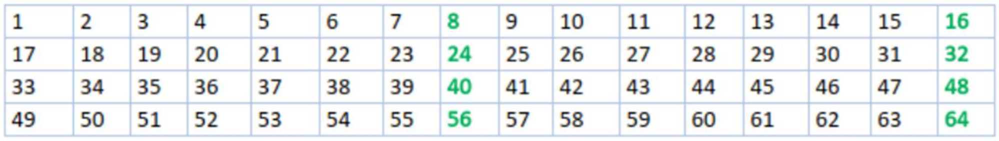
Title: Implementation of DES Algorithm.

Theory:
Data encryption standard (DES) has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in the figure:





We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56-bit key. That is bit positions 8, 16, 24, 32, 40, 48, 56, and 64 are discarded.

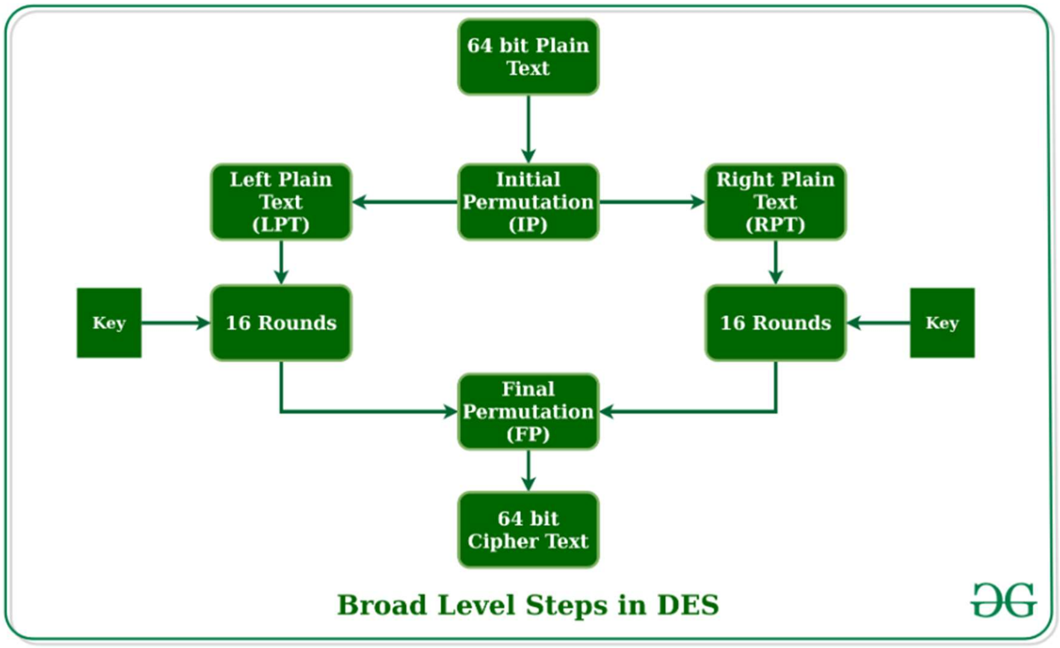


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Figure - discording of every 8th bit of original key

Thus, the discarding of every 8th bit of the key produces a **56-bit key** from the original **64-bit key**. DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

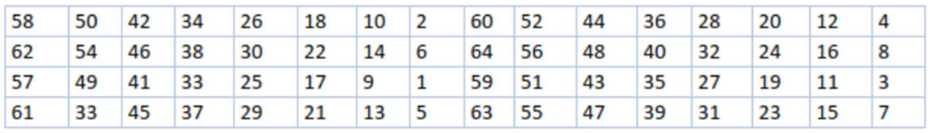
- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
- The result of this process produces 64-bit ciphertext.



Initial Permutation (IP):

As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

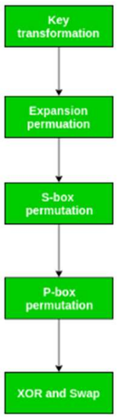
This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.



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	33	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Figure - Initial permutation table

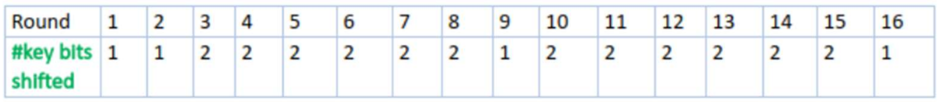
As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad-level steps outlined in the figure.



Step-1: Key transformation:

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

For example: if the round numbers 1, 2, 9, or 16 the shift is done by only one position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in the figure.



Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

After an appropriate shift, 48 of the 56 bits are selected. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves to the first position, bit number 17 moves to the second position, and so on. If we observe the table carefully, we will realize that it contains only 48-bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.

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

Figure - compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

Step-2: Expansion Permutation:

Recall that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.

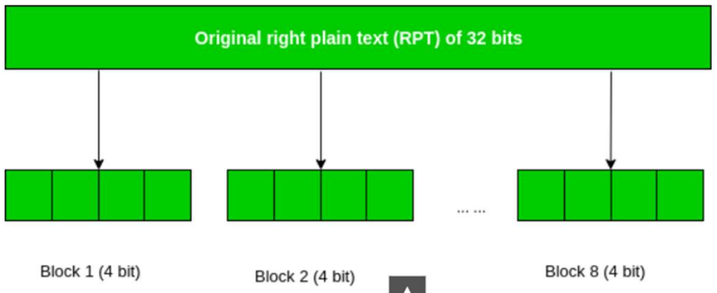
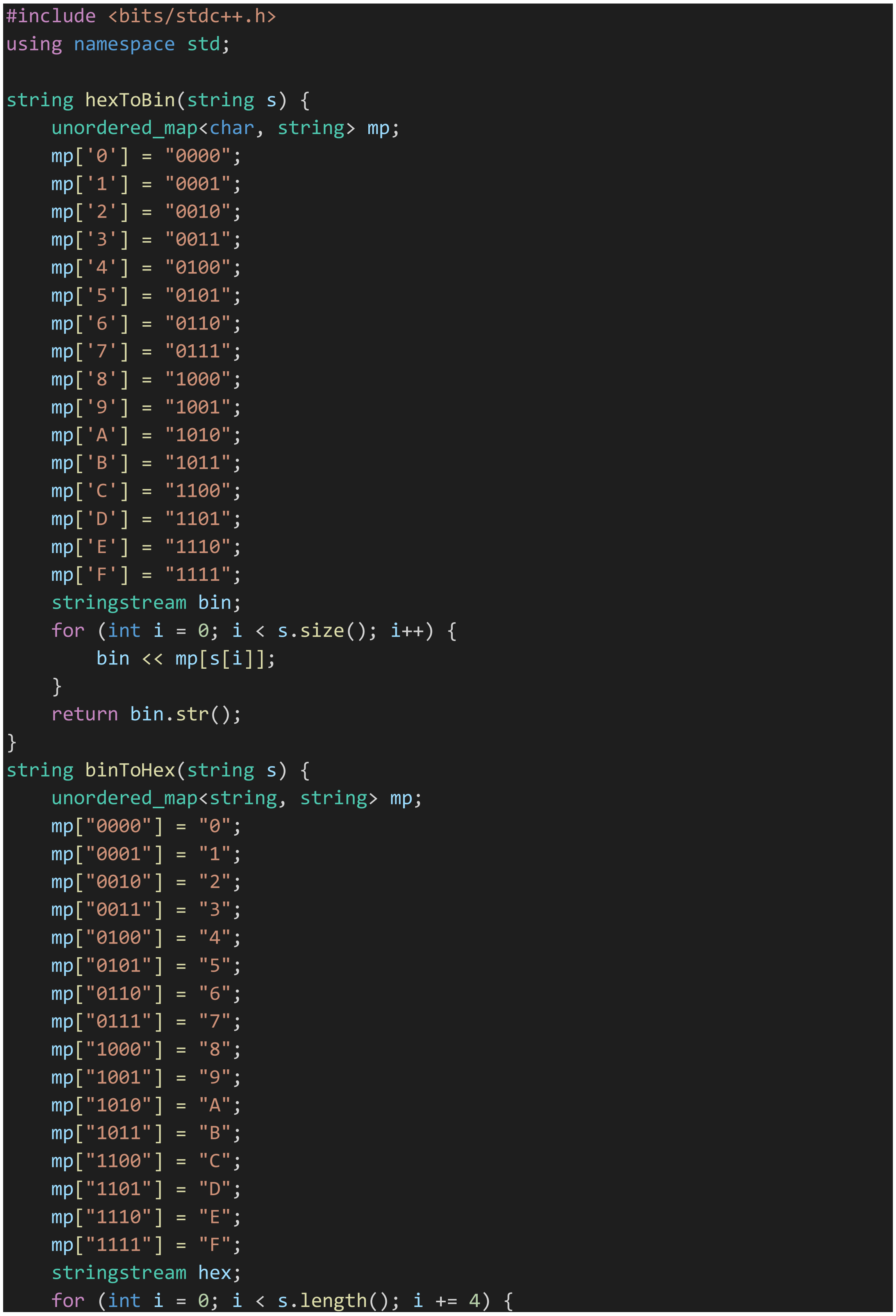


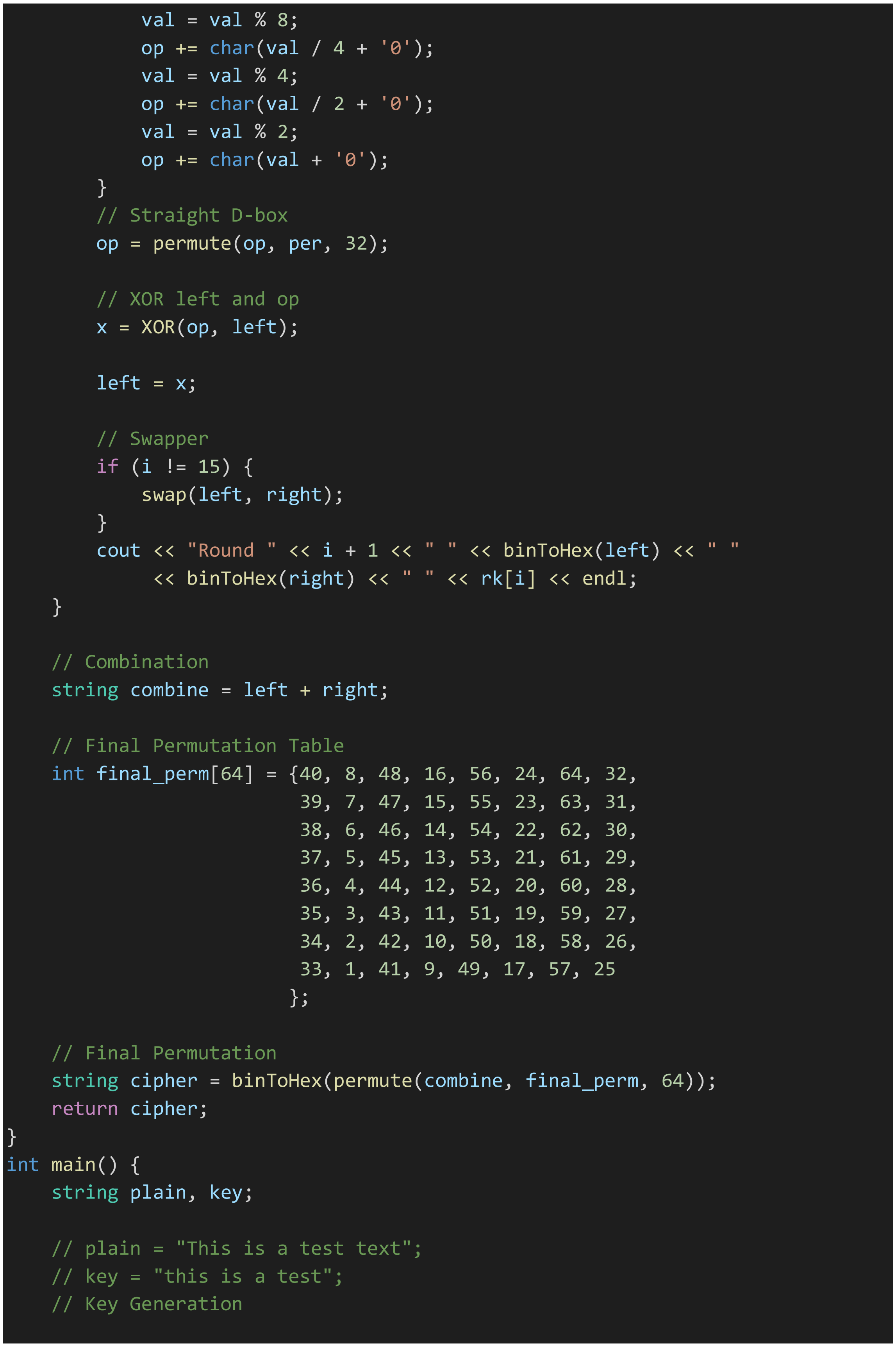
Figure - division of 32 bit RPT into 8 bit blocks

This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the **32-bit RPT** to **48-bits**. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the **S-Box substitution**.

Code Snapshots:



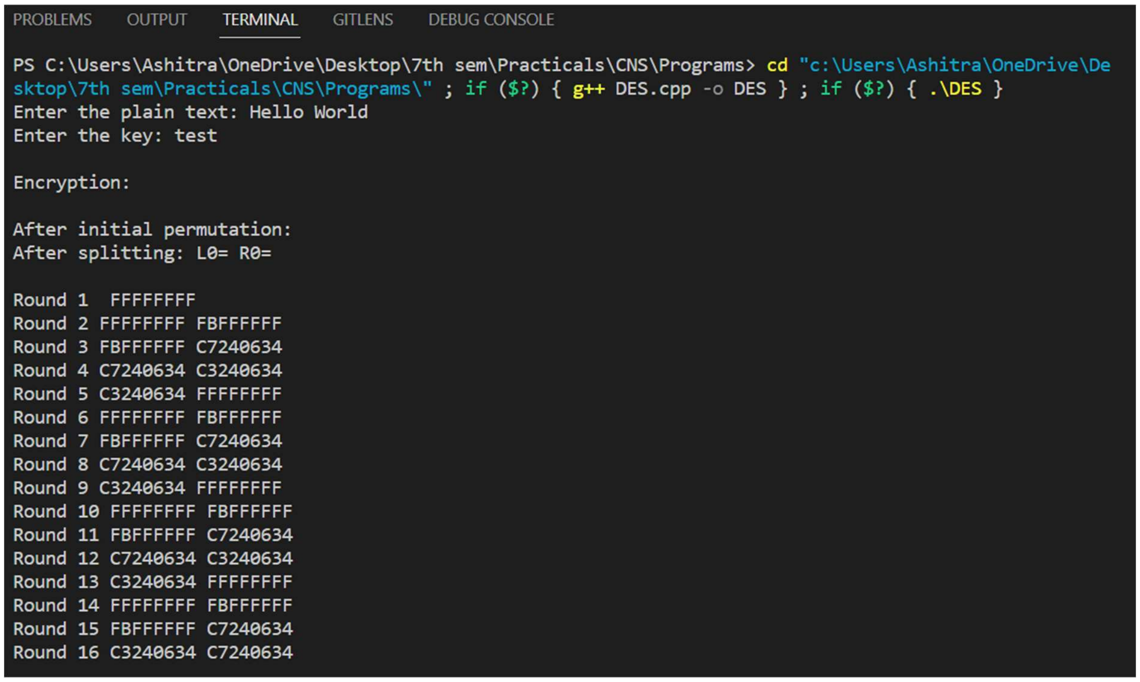
```cpp
#include <bits/stdc++.h>
using namespace std;

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');
```



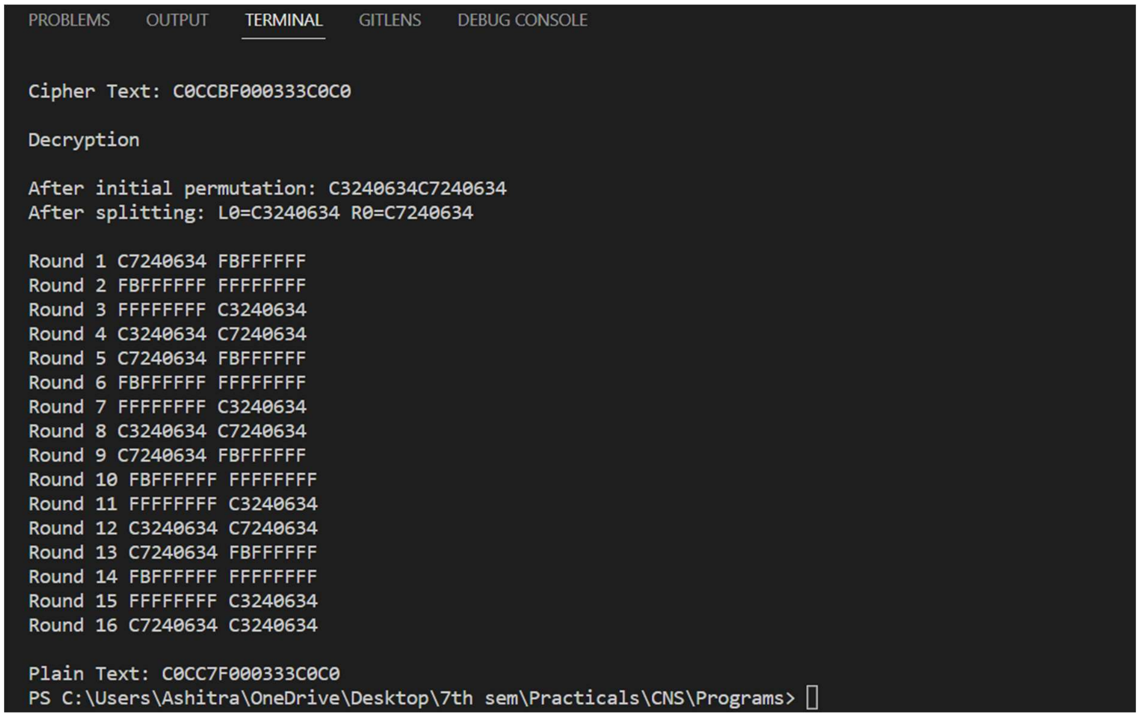
```
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Straight D-box
        op = permute(op, per, 32);

        // 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
```

```cpp
    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;
}
```

Output Snapshots:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\De
sktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ DES.cpp -o DES } ; if ($?) { .\DES }
Enter the plain text: Hello World
Enter the key: test

Encryption:

After initial permutation:
After splitting: L0= R0=

Round 1  FFFFFFFF
Round 2 FFFFFFFF FBFFFFFF
Round 3 FBFFFFFF C7240634
Round 4 C7240634 C3240634
Round 5 C3240634 FFFFFFFF
Round 6 FFFFFFFF FBFFFFFF
Round 7 FBFFFFFF C7240634
Round 8 C7240634 C3240634
Round 9 C3240634 FFFFFFFF
Round 10 FFFFFFFF FBFFFFFF
Round 11 FBFFFFFF C7240634
Round 12 C7240634 C3240634
Round 13 C3240634 FFFFFFFF
Round 14 FFFFFFFF FBFFFFFF
Round 15 FBFFFFFF C7240634
Round 16 C3240634 C7240634
```



```
Cipher Text: C0CCBF000333C0C0

Decryption

After initial permutation: C3240634C7240634
After splitting: L0=C3240634 R0=C7240634

Round 1 C7240634 FBFFFFFF
Round 2 FBFFFFFF FFFFFFFF
Round 3 FFFFFFFF C3240634
Round 4 C3240634 C7240634
Round 5 C7240634 FBFFFFFF
Round 6 FBFFFFFF FFFFFFFF
Round 7 FFFFFFFF C3240634
Round 8 C3240634 C7240634
Round 9 C7240634 FBFFFFFF
Round 10 FBFFFFFF FFFFFFFF
Round 11 FFFFFFFF C3240634
Round 12 C3240634 C7240634
Round 13 C7240634 FBFFFFFF
Round 14 FBFFFFFF FFFFFFFF
Round 15 FFFFFFFF C3240634
Round 16 C7240634 C3240634

Plain Text: C0CC7F000333C0C0
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> 
```

Conclusion:

1. DES is a symmetric block cipher that can be used to encrypt 64-bits of plaintext into 64-bits of ciphertext.
2. The algorithm is the same for the process of encryption as well as decryption. The only difference is that the decryption procedure is the opposite of the encryption procedure.