NIS LAB-11

-Vidit Shah

SEM-VI CE003

AIM: Write a program to implement DES Cipher.

- Encryption
- Decryption
- Key Generation (optional)

- Source code:

```
#include <bits/stdc++.h>
using namespace std;
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    for (int i = binary.length() - 1, j = 0; i >= 0; i--, j++)
    {
        decimal += (binary[i] - '0') * pow(double(2), double(j));
    }
    return decimal;
}
string convertDecimalToBinary(int decimal)
{
    string binary;
    while (decimal != 0)
    {
        binary += to_string(decimal % 2);
        decimal /= 2;
    }
    reverse(binary.begin(), binary.end());
    binary.insert(0, 4 - binary.length(), '0');
    return binary;
}
string convertHexToBinary(string hex)
{
    string binary;
    for (int i = 0; i < hex.size(); i++)
    {
        if (hex[i] >= '0' && hex[i] <= '9')
        {
            binary += convertDecimalToBinary(hex[i] - '0');
        }
```

```
        else if (hex[i] >= 'A' && hex[i] <= 'F')
        {
            binary += convertDecimalToBinary(hex[i] - 'A' + 10);
        }
        else if (hex[i] >= 'a' && hex[i] <= 'f')
        {
            binary += convertDecimalToBinary(hex[i] - 'a' + 10);
        }
        else
        {
            cout << "Please enter the valid hexadecimal value.";
            exit(0);
        }
    }
    return binary;
}
string convertBinaryToHex(string binary)
{
    string hex;
    for (int i = 0; i < binary.size(); i += 4)
    {
        int temp = convertBinaryToDecimal(binary.substr(i, 4));
        if (temp >= 10 && temp <= 15)
        {
            hex += 'A' + temp - 10;
        }
        else
        {
            hex += to_string(temp);
        }
    }
    return hex;
}
string findXor(string str1, string str2)
{
    string xor_result;
    for (int i = 0; i < str1.length(); i++)
    {
        xor_result += to_string(str1[i] ^ str2[i]);
    }
    return xor_result;
}
string expansionPBox(string r)
{
    string expanded_r;
    expanded_r += r[0];
    for (int i = 1; i < r.length(); i++)
    {
```

```
        if (i % 4 == 0)
        {
            expanded_r += r[i];
            expanded_r += r[i - 1];
        }
        expanded_r += r[i];
    }
    expanded_r.insert(0, 1, r[r.length() - 1]);
    expanded_r += r[0];
    return expanded_r;
}
string roundFunction(string r, string k)
{
    r = expansionPBox(r);
    string xor_num = findXor(r, k);
    int s_box[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}};
```

```
    string reduced_str;
    for (int i = 0, j = 0; i < xor_num.length(); i += 6, j++)
    {
        string temp1;
        temp1 += xor_num[i];
        temp1 += xor_num[i + 5];
        string temp2 = xor_num.substr(i + 1, 4);
        reduced_str +=
convertDecimalToBinary(s_box[j][convertBinaryToDecimal(temp1)][co
nvertBinaryToDecimal(temp2)]);
    }
    vector<int> straight_p_table = {
        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};
    string result(32, '0');
    for (int i = 0; i < straight_p_table.size(); i++)
    {
        result[i] = reduced_str[straight_p_table[i] - 1];
    }
    return result;
}
string encryption(string plain_text, vector<string> round_key)
{
    string encrypted_text;
    string l = convertHexToBinary(plain_text.substr(0,
plain_text.size() / 2)), r =
convertHexToBinary(plain_text.substr(plain_text.size() / 2,
plain_text.size() / 2));
    plain_text = convertHexToBinary(plain_text);
    string prev_l;
    for (int i = 0; i < 16; i++)
    {
        prev_l = l;
        l = r;
        r = findXor(prev_l, roundFunction(r,
convertHexToBinary(round_key[i])));
    }
    encrypted_text += r + l;
    return convertBinaryToHex(encrypted_text);
}
string decryption(string encrypted_text, vector<string>
round_key)
{
    string decrypted_text;
    string l = convertHexToBinary(encrypted_text.substr(0,
encrypted_text.size() / 2)), r =
```

```
convertHexToBinary(encrypted_text.substr(encrypted_text.size() /
2, encrypted_text.size() / 2));
    encrypted_text = convertHexToBinary(encrypted_text);
    string prev_l;
    for (int i = 0; i < 16; i++)
    {
        prev_l = l;
        l = r;
        r = findXor(prev_l, roundFunction(r,
convertHexToBinary(round_key[round_key.size() - i - 1])));
    }
    decrypted_text += r + l;
    return convertBinaryToHex(decrypted_text);
}
bool isPlainTextValid(string plain_text)
{
    if (plain_text.size() != 16)
    {
        return false;
    }
    for (int i = 0; i < plain_text.size(); i++)
    {
        if (!((plain_text[i] >= '0' && plain_text[i] <= '9') ||
(plain_text[i] >= 'A' && plain_text[i] <= 'Z') || (plain_text[i]
>= 'a' && plain_text[i] <= 'z')))
        {
            return false;
        }
    }
    return true;
}
int main()
{
    string plain_text;
    cout << "Enter the plain text in hexadecimal: ";
    cin >> plain_text;
    if (!isPlainTextValid(plain_text))
    {
        cout << "Please enter the valid plain text in
hexadecimal.";
        return 0;
    }
    vector<string> round_key{"194CD072DE8C", "4568581ABCCE",
"06EDA4ACF5B5", "DA2D032B6EE3", "69A629FEC913", "C1948E87475E",
"708AD2DDB3C0", "34F822F0C66D",
                             "84BB4473DCCC", "02765708B5BF",
"6D5560AF7CA5", "C2C1E96A4BF3", "99C31397C91F", "251B8BC717D0",
"3330C5D9A36D", "181C5D75C66D"};
```

```
    string encrypted_text = encryption(plain_text, round_key);
    cout << "Encrypted text: " << encrypted_text << endl;
    cout << "Decrypted text: " << decryption(encrypted_text,
round_key);
    return 0;
}
```

- Input/Output:

Test Case1:

```
Enter the plain text in hexadecimal: 123456ABCD132536
Encrypted text: A251DE1A19299E89
Decrypted text: 123456ABCD132536
```

Test Case2:

```
Enter the plain text in hexadecimal: 0123456789ABCDEF
Encrypted text: A3DD68C9C58D7F9D
Decrypted text: 0123456789ABCDEF
```

Test Case3:

```
Enter the plain text in hexadecimal: 1234QWV
Please enter the valid plain text in hexadecimal.
```

xxxENDxxx