CRYPTOGRAPHY AND NETWORK SECURITY
Lab 2 - DES

23BCE1033
Priya Baskar

Aim:

To design and implement the Data Encryption Standard (DES) algorithm for secure data communication between a sender and a receiver using symmetric key cryptography, employing a 64-bit key size and 64-bit block size, and to verify correct encryption and decryption of plaintext.

Algorithm:

Step 1: Input Acquisition

1. Accept an 8-character plaintext from the sender.
2. Accept an 8-character secret key shared between sender and receiver.

Step 2: ASCII to Binary Conversion

1. Convert each character of plaintext into its 8-bit binary equivalent, forming a 64-bit plaintext block.
2. Convert the key similarly into a 64-bit binary key.

Step 3: Key Generation

1. Apply Permuted Choice-1 (PC-1) to reduce the 64-bit key to 56 bits.
2. Split the key into two 28-bit halves.
3. Perform left circular shifts as defined in the DES shift table.
4. Apply Permuted Choice-2 (PC-2) to generate 16 round keys, each of 48 bits.

Step 4: Initial Permutation

1. Apply the Initial Permutation (IP) to the 64-bit plaintext block.
2. Divide the permuted block into Left (L0) and Right (R0) halves of 32 bits each.

Step 5: 16 Feistel Rounds

For each round (1 to 16):

1. Expand the 32-bit right half to 48 bits using the Expansion (E) table.
2. XOR the expanded right half with the corresponding round key.
3. Apply S-Box substitution to compress the data to 32 bits.
4. Apply the Permutation (P) function.
5. XOR the result with the left half.
6. Swap the left and right halves.

Step 6: Final Permutation

1. After the 16th round, swap the halves.
2. Apply the Final Permutation (FP) to obtain the 64-bit ciphertext.

Step 7: Decryption

1. Repeat the same process using the round keys in reverse order.
2. Convert the final binary output back to ASCII to retrieve the original plaintext.

Code:

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

// -------------------- DES TABLES --------------------

int IP[] = {
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
};

int FP[] = {
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
};

int E[] = {
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
};

int P[] = {
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 PC1[] = {
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 PC2[] = {
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
};

int SHIFTS[] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

// -------------------- S-BOXES --------------------

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},
```

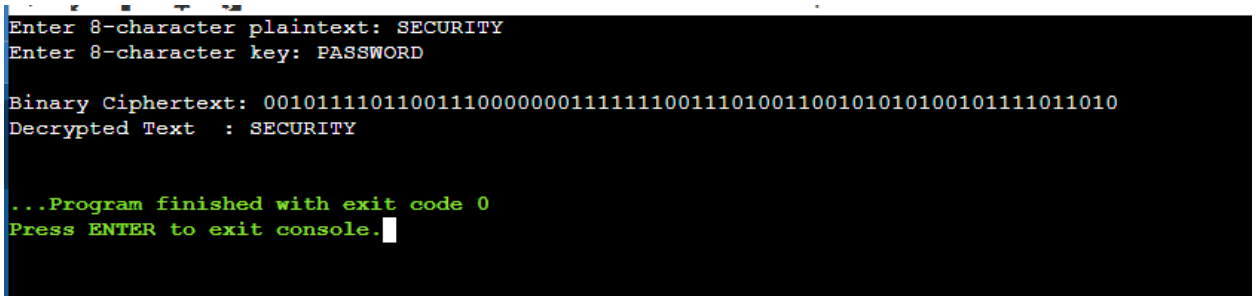
```
{10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4},
{3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14}
},
{
{2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9},
{14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6},
{4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14},
{11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3}
},
{
{12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11},
{10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8},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

/* ------------------ ASCII ↔ BINARY ------------------ */

string ascii_to_binary(string text) {
    string binary = "";
    for (char c : text) {
        binary += bitset<8>(c).to_string();
    }
    return binary;
}

string binary_to_ascii(string binary) {
    string text = "";
    for (int i = 0; i < binary.size(); i += 8) {
        bitset<8> bits(binary.substr(i, 8));
        text += char(bits.to_ulong());
```

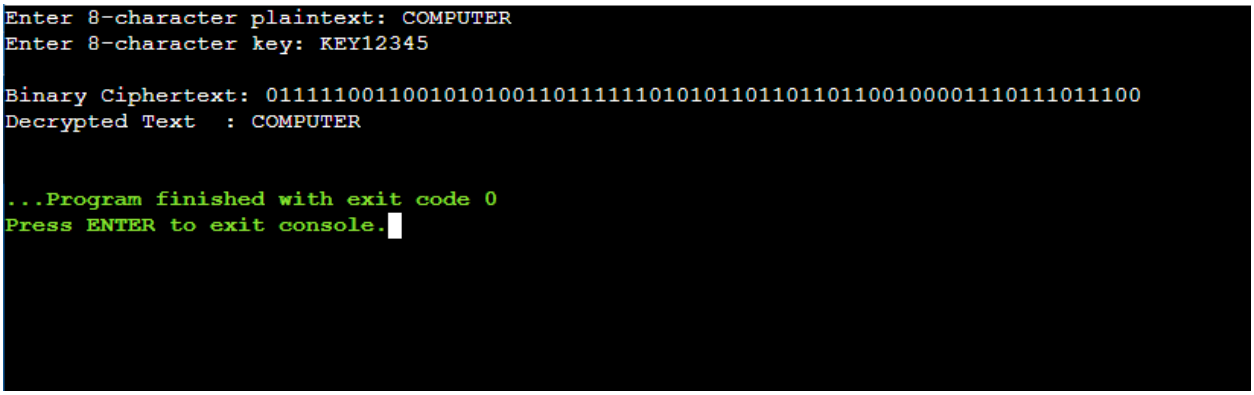
```
    }
    return text;
}

/* ------------------ HELPER FUNCTIONS ------------------ */

string permute(string in, int *table, int n) {
    string out = "";
    for (int i = 0; i < n; i++)
        out += in[table[i] - 1];
    return out;
}

string shift_left(string s, int n) {
    return s.substr(n) + s.substr(0, n);
}

string xor_(string a, string b) {
    string r = "";
    for (int i = 0; i < a.size(); i++)
        r += (a[i] == b[i]) ? '0' : '1';
    return r;
}

/* ------------------ S-BOX FUNCTION ------------------ */

string sbox_substitution(string in) {
    string out = "";
    for (int i = 0; i < 8; i++) {
        string block = in.substr(i * 6, 6);
        int row = (block[0] - '0') * 2 + (block[5] - '0');
        int col = stoi(block.substr(1, 4), nullptr, 2);
        out += bitset<4>(S[i][row][col]).to_string();
    }
    return out;
}

/* ------------------ KEY GENERATION ------------------ */

vector<string> generate_keys(string key) {
    vector<string> keys;
    key = permute(key, PC1, 56);
    string left = key.substr(0, 28);
    string right = key.substr(28, 28);
```

```
    for (int i = 0; i < 16; i++) {
        left = shift_left(left, SHIFTS[i]);
        right = shift_left(right, SHIFTS[i]);
        keys.push_back(permute(left + right, PC2, 48));
    }
    return keys;
}

/* ------------------ DES CORE ------------------ */

string DES(string text, vector<string> keys, bool encrypt) {
    text = permute(text, IP, 64);
    string left = text.substr(0, 32);
    string right = text.substr(32, 32);

    for (int i = 0; i < 16; i++) {
        string expanded = permute(right, E, 48);
        string xored = xor_(expanded, encrypt ? keys[i] : keys[15 - i]);
        string sboxed = sbox_substitution(xored);
        string permuted = permute(sboxed, P, 32);

        string new_right = xor_(left, permuted);
        left = right;
        right = new_right;
    }

    return permute(right + left, FP, 64);
}

/* ------------------ MAIN ------------------ */

int main() {
    string plaintext, key;

    cout << "Enter 8-character plaintext: ";
    getline(cin, plaintext);

    cout << "Enter 8-character key: ";
    getline(cin, key);

    if (plaintext.length() != 8 || key.length() != 8) {
        cout << "Plaintext and key must be exactly 8 characters!\n";
        return 0;
```

```
    }

    // ASCII → Binary
    string pt_bin = ascii_to_binary(plaintext);
    string key_bin = ascii_to_binary(key);

    vector<string> round_keys = generate_keys(key_bin);

    // Encryption
    string cipher_bin = DES(pt_bin, round_keys, true);

    // Decryption
    string decrypted_bin = DES(cipher_bin, round_keys, false);
    string decrypted_text = binary_to_ascii(decrypted_bin);

    cout << "\nBinary Ciphertext: " << cipher_bin << endl;
    cout << "Decrypted Text  : " << decrypted_text << endl;

    return 0;
}
```

Output:





Result:

The DES encryption and decryption algorithm was successfully implemented using C++.

- The plaintext was correctly encrypted into a 64-bit ciphertext using a 64-bit symmetric key.

- Decryption using the same key successfully reproduced the original plaintext, confirming the correctness of the algorithm.

Hence, secure confidential data exchange using DES was achieved.