

Information and Network Security

2CSDE54

Practical 5

21BCE020

Implementation of DES Algorithm

CODE:

```
#include <iostream>
#include <algorithm>
#include <bitset>
#include <cstring>
#include <string>
#include <fstream>
#include <sstream>
#include <utility>
#include <functional>
#include <vector>
#include <cstdint>
using namespace std;

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};

int e_table[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, 28,
                   24, 25, 26, 27, 28, 29,
                   28, 29, 30, 31, 32, 1};

int pc2[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
};

int reverse_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};

int s_box[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}};

int p_box[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};

string ascii_to_bin(string &ip)
{
    string binstr;
    for (char c : ip)
    {
        binstr += bitset<8>(c).to_string();
    }
    return binstr;
}

string bin_to_str(string &resultant_bin) {

```

```

    string str = "";
    stringstream sstream(resultant_bin);
    while (sstream.good()) {
        bitset<64> bits;
        sstream >> bits;

        // Check for replacement character (ASCII 26) and
space (ASCII 32)
        if (bits.to_ullong() == 26) {
            str += "[REPLACEMENT]";
        } else if (bits.to_ullong() == 32) {
            str += "[SPACE]";
        } else {
            // Convert to char if not a special case
            str += static_cast<char>(bits.to_ullong());
        }
    }
    return str;
}

pair<string, string> splitting(string &binstr) {
    string l = "";
    string r = "";
    pair<string, string> p;
    int len = binstr.length();
    l = binstr.substr(0, len / 2);
    r = binstr.substr(len / 2);
    p.first = l;
    p.second = r;
    return p;
}

string XOR(string& str1, string& str2) {
    string result;
    for (size_t i = 0; i < str1.length(); ++i) { //size_t to
avoid negative size
        if (str1[i] == str2[i]) {
            result += '0';
        } else {
            result += '1';
        }
    }
}

```

```

    }
    return result;
}

void leftshift(string &str, int shift_count) {
    rotate(str.begin(), str.begin() + shift_count, str.end());
}

string block(string &binstr)
{
    // string binstr = ascii_to_bin(ip);
    char bit_block[binstr.size() + 1];
    copy(binstr.begin(), binstr.end(), bit_block);
    bit_block[binstr.size()] = '\0';
    return bit_block;
}

string permutation(string &ip, int permutation_block[], int
block_size)
{
    string bit_block = block(ip);
    bitset<64> new_bit_block(bit_block);
    bitset<64> resultant_bits;
    for (int i = 0; i < block_size; ++i) {
        resultant_bits[i] = new_bit_block[permutation_block[i]
- 1];
    }
    string result = resultant_bits.to_string().substr(64 -
block_size, block_size); // Corrected substring
    return result;
}

string key_gen(string &key, int round){
    string binkey = ascii_to_bin(key);
    string reduced_key;
    for (int i = 0; i < binkey.size(); ++i) {
        if ((i + 1) % 8 != 0) {
            reduced_key += binkey[i];
        }
    }
    pair<string, string> key_pair = splitting(reduced_key);
}

```

```

    string left_k = key_pair.first;
    string right_k = key_pair.second;
    int shift_count = (round == 1) ? 1 : 2;
    leftshift(left_k, shift_count);
    leftshift(right_k, shift_count);
    string resultant_str = left_k + right_k;
    string final_key = permutation(resultant_str, pc2, 48);
    return final_key;
}

void substring_div(bitset<48> input, bitset<6> substrings[8])
{
    for (int i = 0; i < 8; ++i) {
        substrings[i] = (input >> (i * 6)).to_ulong() &
0b111111;
    }
}

bitset<4> substitution(bitset<6> input, int s_box[4][16]) {
    int row = static_cast<int>((input[0] << 1) + input[5]);
    int col = static_cast<int>((input[1] << 3) + (input[2] <<
2) + (input[3] << 1) + input[4]);
    int sBoxValue = s_box[row][col];
    bitset<4> result(sBoxValue);
    return result;
}

string DES(string &perm_inp, string &k){
    pair<string, string> p = splitting(perm_inp); // split
input str
    string left_ip = p.first;
    string right_ip = p.second;
    string expanded_right = permutation(right_ip, e_table,
48); // pass R through expansion table
    // cout<<expanded_right<<"
"<<expanded_right.length()<<endl;
    string f1 = XOR(expanded_right, k); // XOR expanded key
with generated key
    bitset<48> r1(f1); // store XOR op
    bitset<6> substrings[8]; // divide str
    substring_div(r1, substrings);
}

```

```

        bitset<4> result[8];
        for (int i = 0; i < 8; ++i) {
            result[i] = substitution(substrings[i], s_box); //
pass the substr into substitution table
        }
        bitset<32> intermediate_result;
        for (int i = 0; i < 8; ++i) {
            intermediate_result <= 4;
            intermediate_result |=
static_cast<uint32_t>(result[i].to_ulong()); // combine new
substr
        }
        string new_res = intermediate_result.to_string();
        // cout<<new_res<<" "<<new_res.length()<<endl;
        string permuted_pbox_right = permutation(new_res, p_box,
32); // permute the new right str with pbox
        // cout<<permuted_pbox_right<<"
"<<permuted_pbox_right.length()<<endl;
        // string left_str = ascii_to_bin(left_ip);
        // string right_str = ascii_to_bin(right_ip);
        string r2 = XOR(left_ip, permuted_pbox_right); //
generate new right str r2
        // cout<<r2<<" "<<r2.length()<<endl;
        string l2 = right_ip; // pass the initial right str to new
left str l2
        // cout<<l2<<" "<<l2.length()<<endl;
        string concatenated_str = l2 + r2;
        // cout<<concatenated_str<<"
"<<concatenated_str.length()<<endl;
        string final_bit_res = permutation(concatenated_str,
reverse_permutation, 64); // pass the final str into reverse
inverse permutation table
        // cout<<final_bit_res<<" "<<final_bit_res.length()<<endl;
        // string final_str = bin_to_str(final_bit_res);
        return final_bit_res;
    }

int main() {
    ifstream f1("des_inp.txt");
    ifstream f2("key.txt");
    ofstream f3("CT.txt");

```

```

ofstream f4("PT.txt");
string ip;
string key;
f1>>ip;
f2>>key;
string binstr = ascii_to_bin(ip);
string key1 = key_gen(key, 1);
string key2 = key_gen(key, 2);
string enc1 = DES(binstr, key1);
string enc2 = DES(enc1, key2);
string dec1 = DES(enc2, key2);
string dec2 = DES(dec1, key1);
cout<<"Original Message: "<<ip<<endl;
cout<<"Encryption Round 1: "<<bin_to_str(enc1)<<endl;
cout<<"Encryption Round 2: "<<bin_to_str(enc2)<<endl;
cout<<"Decryption Round 1: "<<bin_to_str(dec1)<<endl;
cout<<"Decryption Round 2: "<<bin_to_str(dec2)<<endl;

f3<<"Encryption Round 1: "<<bin_to_str(enc1)<<endl;
f3<<"Encryption Round 2: "<<bin_to_str(enc2);
f4<<"Decryption Round 1: "<<bin_to_str(dec1)<<endl;
f4<<"Decryption Round 2: "<<bin_to_str(dec2);
return 0;
}

```

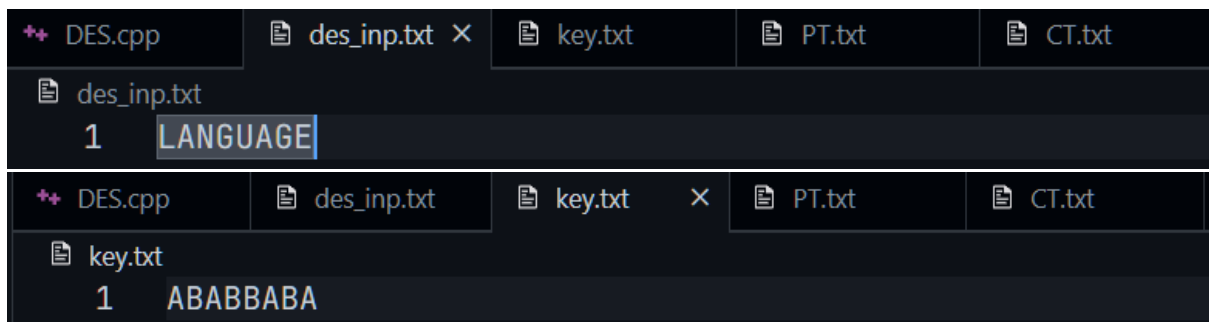
OUTPUT:

```

Original Message: LANGUAGE
Encryption Round 1: ê
Encryption Round 2: 7
Decryption Round 1: f
Decryption Round 2: 8

```

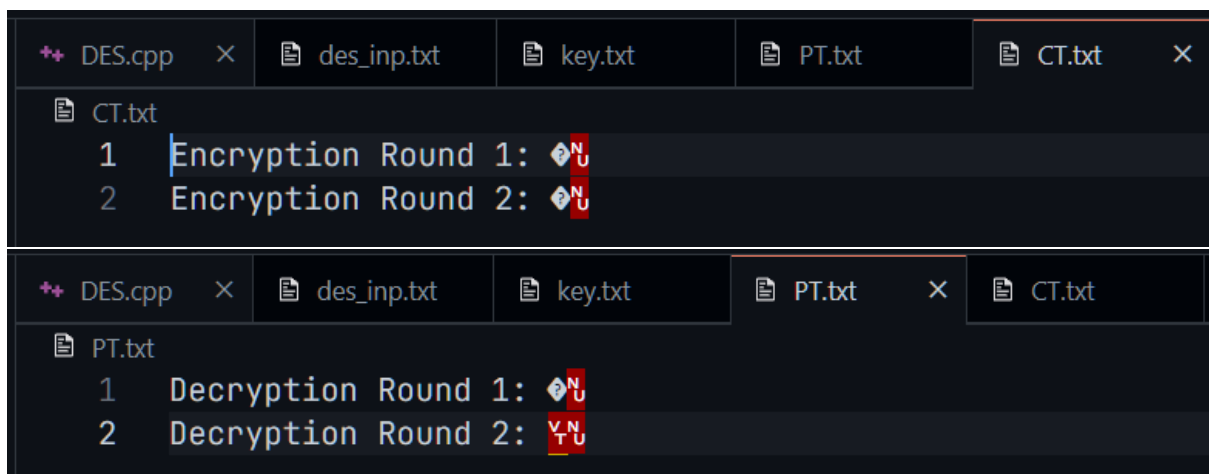
INPUT:



```
DES.cpp  des_inp.txt x key.txt PT.txt CT.txt
des_inp.txt
1 LANGUAGE

DES.cpp  des_inp.txt key.txt x PT.txt CT.txt
key.txt
1 ABABBABA
```

TEXT FILE OUTPUT:



```
DES.cpp x des_inp.txt key.txt PT.txt CT.txt x
CT.txt
1 Encryption Round 1: 00000000
2 Encryption Round 2: 00000000

DES.cpp x des_inp.txt key.txt PT.txt x CT.txt
PT.txt
1 Decryption Round 1: 00000000
2 Decryption Round 2: 00000000
```

.....