**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\_ullong() & 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 permutated\_pbox\_right = permutation(new\_res, p\_box, 32); // permute the new right str with pbox

    // cout<<permutated\_pbox\_right<<" "<<permutated\_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, permutated\_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);

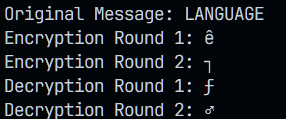
    f4<<"Decryption Round 1: "<<bin\_to\_str(dec1)<<endl;

    f4<<"Decryption Round 2: "<<bin\_to\_str(dec2);

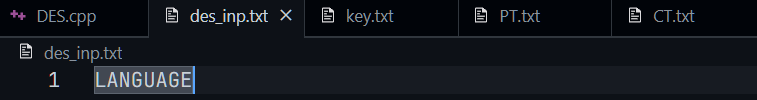
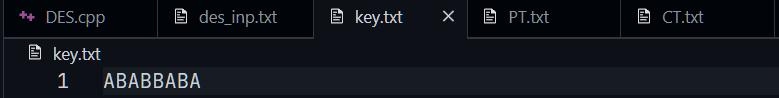
    return 0;

}

OUTPUT:



INPUT:

TEXT FILE OUTPUT: