## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **SPN and Feistel Cipher**

## Instructor: **Dr. Meenakshi Panda**

## Date: **16th October 2024**

// Q1. Write a program to encrypt and decrypt the message using

// the Substitution and permutation network.

#include <bits/stdc++.h>

using namespace std;

string Encryption(int no\_rows, int len\_key, int len\_msg,

                  string msg, int col\_val[])

{

    int x = 0;

    char enc\_mat[no\_rows + 1][len\_key];

    cout << "Message matrix:\n";

    for (int i = 0; i < no\_rows + 1; i++) {

        for (int j = 0; j < len\_key; j++) {

            if (x >= len\_msg) {

                enc\_mat[i][j] = ' ';

            }

            else {

                enc\_mat[i][j] = msg[x];

            }

            cout << enc\_mat[i][j] << " ";

            x++;

        }

        cout << endl;

    }

    int t = 1;

    string cipher = "";

    cout << "Rearranging columns based on key permutation:\n";

    while (t <= len\_key) {

        for (int i = 0; i < len\_key; i++) {

            int k = col\_val[i];

            if (k == t) {

                cout << "Column " << i << " (key position " << t << "): ";

                for (int j = 0; j < no\_rows + 1; j++) {

                    cipher += enc\_mat[j][i];

                    cout << enc\_mat[j][i];

                }

                cout << endl;

                t++;

            }

        }

    }

    cout << "Cipher text: " << cipher << endl;

    return cipher;

}

string Decryption(int no\_rows, int len\_key, string cipher,

                  int col\_val[])

{

    char dec\_mat[no\_rows + 1][len\_key];

    int x = 0, t = 1;

    while (t <= len\_key) {

        for (int i = 0; i < len\_key; i++) {

            int k = col\_val[i];

            if (k == t) {

                for (int j = 0; j < no\_rows + 1; j++) {

                    dec\_mat[j][i] = cipher[x];

                    x++;

                }

                t++;

            }

        }

    }

    string message = "";

    for (int i = 0; i < no\_rows + 1; i++) {

        for (int j = 0; j < len\_key; j++) {

            if (dec\_mat[i][j] == '\_') {

                dec\_mat[i][j] = ' ';

            }

            message += dec\_mat[i][j];

        }

    }

    return message;

}

int main()

{

    string msg, key;

    cout << "Enter the message: ";

    getline(cin, msg);

    cout << "Enter the key: ";

    cin >> key;

    msg.erase(remove(msg.begin(), msg.end(), ' '), msg.end());

    int len\_key = key.length();

    int len\_msg = msg.length();

    int val = 1, count = 0, ind;

    int col\_val[len\_key];

    memset(col\_val, 0, sizeof(col\_val));

    while (count < len\_key) {

        int min = 999;

        for (int i = 0; i < len\_key; i++) {

            if ((min > int(key[i])) && (col\_val[i] == 0)) {

                min = int(key[i]);

                ind = i;

            }

        }

        col\_val[ind] = val;

        count++;

        val++;

    }

    int no\_rows = len\_msg / len\_key;

    string cipher\_text = " ";

    cipher\_text = Encryption(no\_rows, len\_key, len\_msg, msg,

                             col\_val);

    cout << "Encrypted Message : " << cipher\_text << endl;

    string original\_msg = " ";

    original\_msg = Decryption(no\_rows, len\_key, cipher\_text,

                              col\_val);

    cout << "Decrypted Message : " << original\_msg << endl;

}

## **Output:**

## 

# Q2. Write a program to implement the Feistel Cipher.

import binascii

def rand\_key(p):

    import random

    key1 = ""

    p = int(p)

    for i in range(p):

        temp = random.randint(0,1)

        temp = str(temp)

        key1 = key1 + temp

    return(key1)

def exor(a, b):

    temp = ""

    n = len(a)  # Define n as the length of a

    for i in range(n):

        if (a[i] == b[i]):

            temp += "0"

        else:

            temp += "1"

    return temp

def BinaryToDecimal(binary):

    string = int(binary, 2)

    return string

def get\_user\_input():

    PT = input("Enter the plaintext: ").replace(" ", "")  # Remove spaces

    num\_rounds = int(input("Enter the number of rounds: "))

    return PT, num\_rounds

# Get user input

PT, num\_rounds = get\_user\_input()

print("Plain Text is:", PT)

PT\_Ascii = [ord(x) for x in PT]

PT\_Bin = [format(y,'08b') for y in PT\_Ascii]

PT\_Bin = "".join(PT\_Bin)

n = int(len(PT\_Bin)//2)

L1 = PT\_Bin[0:n]

R1 = PT\_Bin[n::]

m = len(R1)

# Generate keys for each round

keys = [rand\_key(m) for \_ in range(num\_rounds)]

print("Keys:", keys)

# Encryption

L, R = L1, R1

for i in range(num\_rounds):

    f = exor(R, keys[i])

    L, R = R, exor(f, L)

# Cipher text

bin\_data = L + R

str\_data =' '

# Ensure binary data is a multiple of 7 bits for conversion

if len(bin\_data) % 7 != 0:

    bin\_data = bin\_data.zfill(len(bin\_data) + (7 - len(bin\_data) % 7))

for i in range(0, len(bin\_data), 7):

    temp\_data = bin\_data[i:i + 7]

    decimal\_data = BinaryToDecimal(temp\_data)

    str\_data = str\_data + chr(decimal\_data)

print("Cipher Text:", str\_data)

# Decryption

L, R = L, R

for i in range(num\_rounds - 1, -1, -1):

    f = exor(L, keys[i])

    R, L = L, exor(f, R)

PT1 = L + R

PT1 = int(PT1, 2)

try:

    RPT = binascii.unhexlify('%x' % PT1).decode('utf-8')

except (binascii.Error, ValueError) as e:

    print("Error in decryption:", e)

    RPT = None

print("Retrieved Plain Text is: ", RPT)

## **Output:**

## 