**Assignment No 2**

**Cryptography and Network Security Lab (5CS453)**

**Name: Sumit Narake PRN: 2020BTECS00023**

**Class: Final Year - CSE**

**Title:**

**Encryption and Decryption using Transposition Cipher Technique.**

**Aim:**

**To Study and Implement Encryption and Decryption using Rail Fence Transposition Cipher Technique and Columnar Transposition Cipher Technique**

**Theory:**

**1.Rail Fence Transposition Cipher Technique**

🡪

The Rail Fence Transposition Cipher, also known as the Zigzag Cipher, is a simple columnar transposition cipher technique.

It involves arranging the plaintext characters in a zigzag pattern across multiple rows, known as "rails," and then reading them off row by row to create the encrypted message.

While this cipher is easy to understand and implement, it lacks strong security and is mainly used for educational purposes or simple puzzles.

**Encryption:**

* Choose the number of rails (rows) for the zigzag pattern.
* Write the message diagonally across the rails, moving up and down.
* Read the characters row by row to form the encrypted message.

**Decryption:**

* Create the zigzag pattern with the chosen number of rails.
* Leave blank spaces in the pattern for characters to be placed.
* Fill in the blanks with the encrypted characters, row by row.
* Read the characters diagonally to retrieve the original message.

**Advantages**:

* Easy to understand and implement.
* Provides basic encryption and breaks up character repetition.

**Disadvantages:**

* Not secure against modern cryptanalysis.
* Security depends on the number of rails, making it less practical for strong encryption.

**CODE**:

#include <iostream>

#include <string>

using namespace std;

string encryptRailFence(string plaintext, int rails) {

    string ciphertext;

    string railMatrix[rails];

    int row = 0;

    bool directionDown = false;

    for (char c : plaintext) {

        railMatrix[row] += c;

        if (row == 0 || row == rails - 1) {

            directionDown = !directionDown;

        }

        if (directionDown) {

            row++;

        } else {

            row--;

        }

    }

    for (int i = 0; i < rails; i++) {

        ciphertext += railMatrix[i];

    }

    return ciphertext;

}

string decryptRailFence(string ciphertext, int rails) {

    string plaintext;

    string railMatrix[rails];

    int row = 0;

    bool directionDown = false;

    for (int i = 0; i < ciphertext.length(); i++) {

        railMatrix[row] += 'X'; // Fill the railMatrix with placeholders

        if (row == 0 || row == rails - 1) {

            directionDown = !directionDown;

        }

        if (directionDown) {

            row++;

        } else {

            row--;

        }

    }

    int index = 0;

    for (int i = 0; i < rails; i++) {

        for (int j = 0; j < railMatrix[i].length(); j++) {

            railMatrix[i][j] = ciphertext[index++];

        }

    }

    row = 0;

    directionDown = false;

    for (int i = 0; i < ciphertext.length(); i++) {

        plaintext += railMatrix[row][0];

        railMatrix[row].erase(0, 1);

        if (row == 0 || row == rails - 1) {

            directionDown = !directionDown;

        }

        if (directionDown) {

            row++;

        } else {

            row--;

        }

    }

    return plaintext;

}

int main() {

    string plaintext, ciphertext, decryptedText;

    int rails;

    cout << "Enter the plaintext: ";

    getline(cin, plaintext);

    cout << "Enter the number of rails: ";

    cin >> rails;

    ciphertext = encryptRailFence(plaintext, rails);

    cout << "Encrypted ciphertext: " << ciphertext << endl;

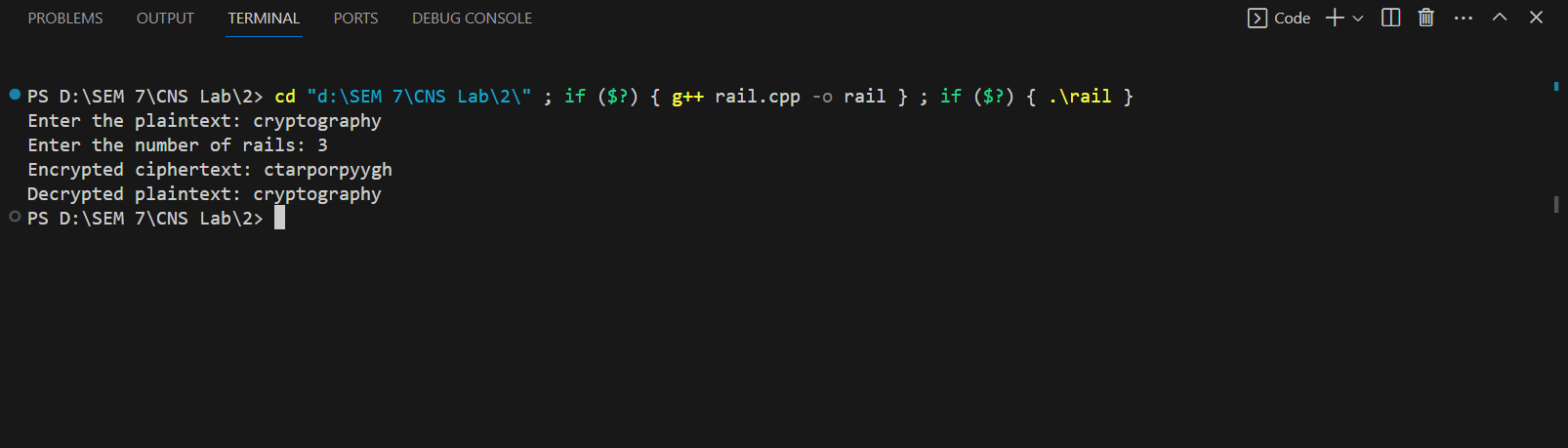
    decryptedText = decryptRailFence(ciphertext, rails);

    cout << "Decrypted plaintext: " << decryptedText << endl;

    return 0;

}

**OUTPUT:**



**2.Columnar Transposition Cipher Technique**

🡪

The Columnar Transposition Cipher is a more advanced transposition cipher technique that involves reordering the characters of a message based on a chosen keyword or key phrase.

It provides a higher level of security compared to simpler ciphers like the Rail Fence Cipher. Here's how the Columnar Transposition Cipher works:

**Encryption:**

* Choose a keyword or key phrase. The unique characters of the keyword determine the order of columns in the transposition grid.
* Write the message row by row into a grid, using the keyword to determine the order of columns.
* Read the characters column by column to obtain the encrypted message.

**Decryption:**

* Use the keyword to determine the order of columns in the transposition grid.
* Write the encrypted message into the grid column by column.
* Read the characters row by row to retrieve the original plaintext.

**Advantages:**

* Offers stronger security compared to simpler ciphers.
* Security depends on the length and uniqueness of the keyword.

**Disadvantages**:

* Can be vulnerable to attacks if the keyword is short or easily guessed.
* May require additional padding characters for messages that don't fit evenly into the grid.

**Code:**

#include <bits/stdc++.h>

using namespace std;

#define ll long long

int main()

{

    string plainText, key;

    cout << "\n Choose an option:\n";

    cout << "  1. Encryption\n";

    cout << "  2. Decryption\n";

    int choice;

    cin >> choice;

    cin.ignore();

    if (choice == 1)

    {

        // Encryption

        cout << "\n Enter plain text : ";

        getline(cin, plainText);

        cout << "\n Enter key : ";

        getline(cin, key);

        // Removing spaces and converting to lowercase from plaintext

        string temp = "";

        for (int i = 0; i < plainText.size(); i++)

        {

            if (plainText[i] != ' ')

                temp += tolower(plainText[i]);

        }

        plainText = temp;

        // Removing spaces and converting to lowercase from key

        string temp2 = "";

        for (int i = 0; i < key.size(); i++)

        {

            if (key[i] != ' ')

                temp2 += tolower(key[i]);

        }

        key = temp2;

        // Encryption

        map<char, vector<char>> mp;

        int keyCounter = 0;

        for (int i = 0; i < plainText.size(); i++)

        {

            mp[key[keyCounter++]].push\_back(plainText[i]);

            if (keyCounter == key.size())

                keyCounter = 0;

        }

        string cipherText;

        for (auto it : mp)

        {

            for (int i = 0; i < it.second.size(); i++)

            {

                cipherText += it.second[i];

            }

        }

        cout << "\n Cipher text is : " << cipherText << endl;

    }

    else if (choice == 2)

    {

        // Decryption

        cout << "\n Enter cipher text : ";

        getline(cin, plainText);

        cout << "\n Enter key : ";

        getline(cin, key);

        // Removing spaces and converting to lowercase from key

        string temp2 = "";

        for (int i = 0; i < key.size(); i++)

        {

            if (key[i] != ' ')

                temp2 += tolower(key[i]);

        }

        key = temp2;

        // Decryption

        map<int, int> dmp;

        int common = plainText.size() / key.size();

        int extra = plainText.size() % key.size();

        for (int i = 0; i < key.size(); i++)

        {

            if (i < extra)

                dmp[i] = common + 1;

            else

                dmp[i] = common;

        }

        map<int, vector<char>> dmp2;

        int start = 0;

        string sortedKey = key;

        sort(sortedKey.begin(), sortedKey.end());

        for (int i = 0; i < sortedKey.size(); i++)

        {

            for (int j = 0; j < key.size(); j++)

            {

                if (sortedKey[i] == key[j])

                {

                    for (int k = 0; k < dmp[j]; k++)

                    {

                        dmp2[key[j]].push\_back(plainText[start++]);

                    }

                }

            }

        }

        string afterDecryption;

        vector<int> counters(key.size(), 0);

        int i = 0;

        while (afterDecryption.size() < plainText.size())

        {

            for (int i = 0; i < key.size(); i++)

            {

                if (counters[i] < dmp[i])

                    afterDecryption += dmp2[key[i]][counters[i]++];

            }

        }

        cout << "\n\n Text after decryption is : " << afterDecryption << endl;

    }

    else

    {

        cout << "\n Invalid choice" << endl;

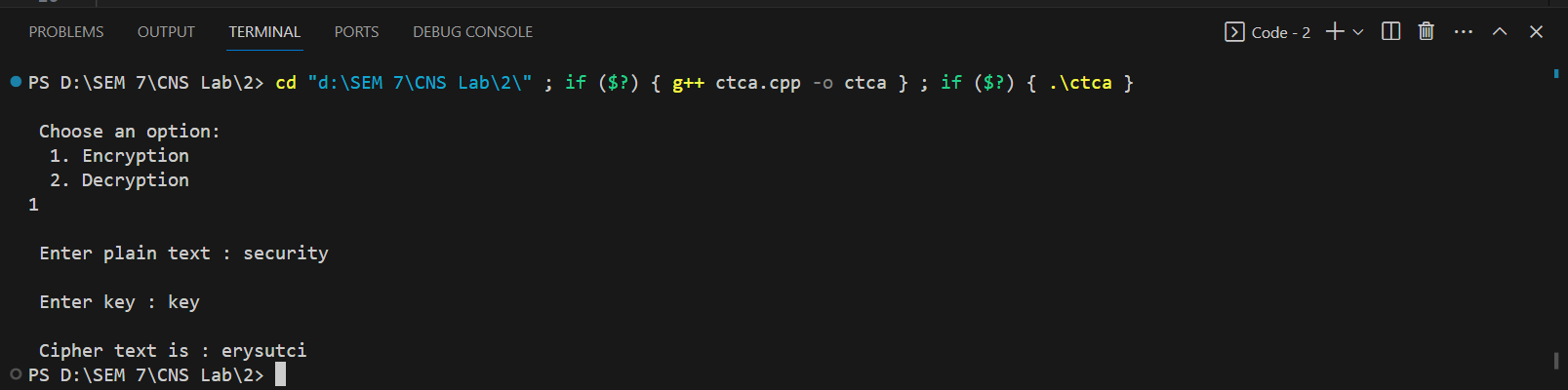
    }

    return 0;

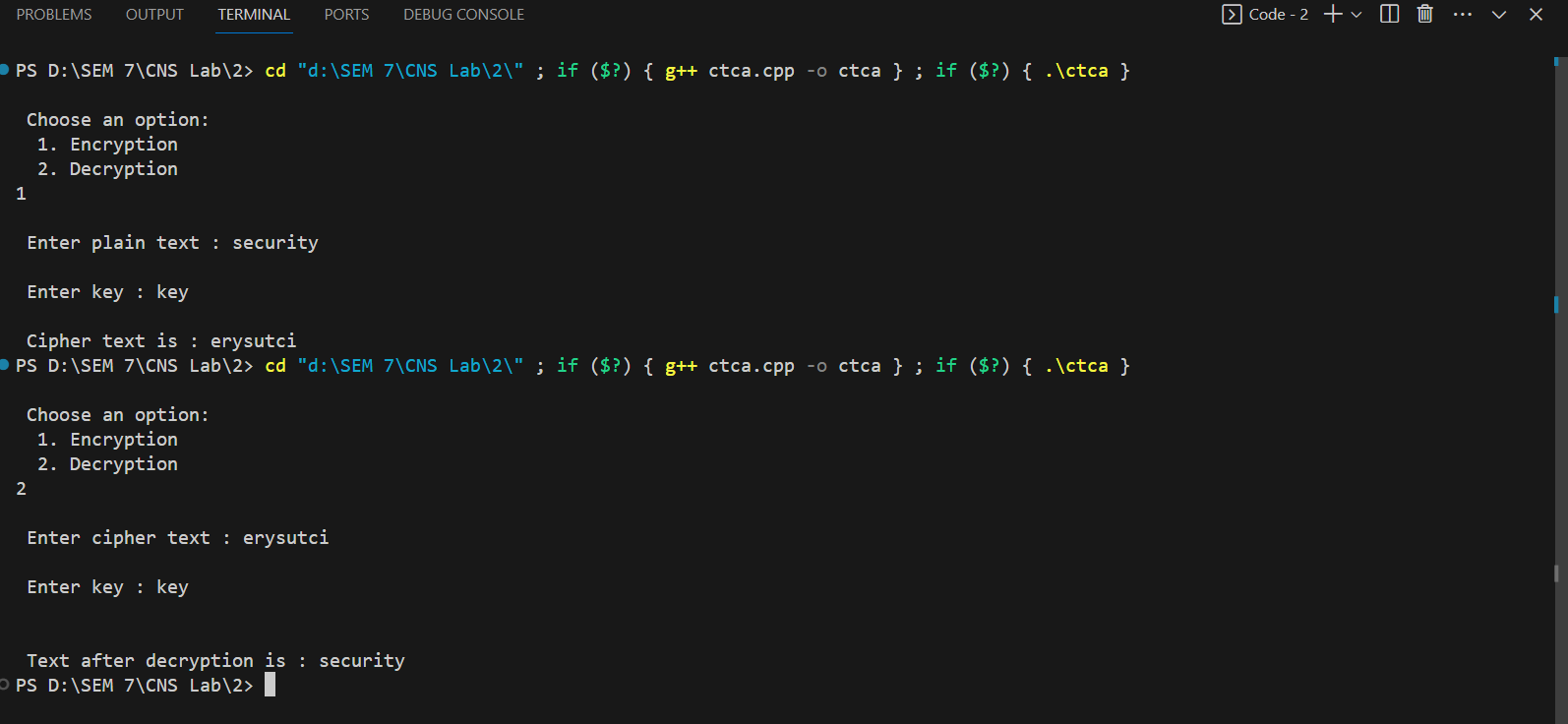
}

**Output:**

**Encryption:**

****

**Decryption:**

****