**Cryptography and Network Security**

**Lab**

**Assignment No. 2**

**Name : Sourabh Shankar Patil**

**PRN : 21510045**

**Batch : B2**

**1. Rail Fence Cipher**

The **Rail Fence Cipher** is a form of transposition cipher that rearranges the letters of the plaintext to create ciphertext. The method involves writing the plaintext in a zigzag pattern across multiple "rails" (lines) and then reading the letters off row by row.

**How It Works:**

1. **Determine the Number of Rails**: Decide how many rails (lines) you want to use.
2. **Write the Plaintext**: Write the plaintext diagonally down and up across the rails. When you reach the bottom rail, go back up to the top rail, continuing this zigzag pattern.
3. **Read Off the Ciphertext**: After filling the rails, read the letters from each rail sequentially to form the ciphertext.

**Example:**

* **Plaintext**: "HELLO WORLD"
* **Number of Rails**: 3
* **Ciphertext**: "HOR ELWLD"

**Weakness**: The Rail Fence Cipher is relatively easy to break with frequency analysis, especially if the number of rails is known.

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Function to encrypt using the Rail Fence Cipher

string railFenceEncrypt(string text, int key)

{

    if (key == 1)

        return text; // Special case where key is 1 (no encryption)

    vector<string> rail(key);

    int direction = -1;

    int row = 0;

    for (char c : text)

    {

        rail[row] += c;

        // Change direction when you reach the top or bottom rail

        if (row == 0 || row == key - 1)

        {

            direction \*= -1;

        }

        row += direction;

    }

    // Combine all the rails to form the ciphertext

    string cipherText;

    for (const string &line : rail)

    {

        cipherText += line;

    }

    return cipherText;

}

// Function to decrypt using the Rail Fence Cipher

string railFenceDecrypt(string cipherText, int key)

{

    if (key == 1)

        return cipherText; // Special case where key is 1 (no decryption)

    vector<int> railLength(key, 0);

    int direction = -1;

    int row = 0;

    // First determine the length of each rail

    for (char c : cipherText)

    {

        railLength[row]++;

        if (row == 0 || row == key - 1)

        {

            direction \*= -1;

        }

        row += direction;

    }

    // Now, populate the rails with the cipher text

    vector<string> rail(key);

    int index = 0;

    for (int i = 0; i < key; i++)

    {

        rail[i] = cipherText.substr(index, railLength[i]);

        index += railLength[i];

    }

    // Reconstruct the original text by reading in zigzag order

    string plainText;

    row = 0;

    direction = -1;

    int railPos[key] = {0};

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

    {

        plainText += rail[row][railPos[row]++];

        if (row == 0 || row == key - 1)

        {

            direction \*= -1;

        }

        row += direction;

    }

    return plainText;

}

int main()

{

    string text;

    int key;

    cout << "Enter the plaintext: ";

    getline(cin, text);

    cout << "Enter the key (number of rails): ";

    cin >> key;

    string cipherText = railFenceEncrypt(text, key);

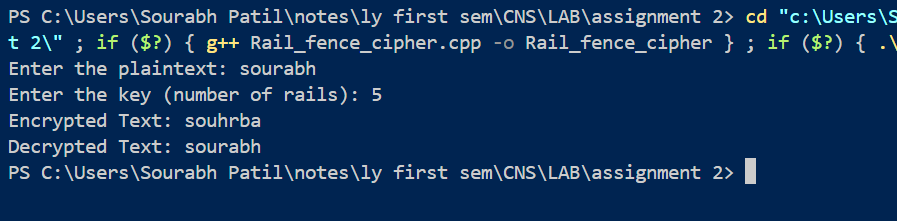
    cout << "Encrypted Text: " << cipherText << endl;

    string decryptedText = railFenceDecrypt(cipherText, key);

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

    return 0;

}

****

**2. Row Column Transformation Cipher**

The **Row Column Transformation Cipher** is a transposition cipher that encrypts the plaintext by rearranging the characters based on a specified row and column structure. This method typically involves creating a grid or matrix to organize the plaintext letters and then applying a specific transformation to generate the ciphertext.

**How It Works:**

1. **Choose a Key**: Select a key that will dictate the arrangement of the rows and columns.
2. **Create a Matrix**: Write the plaintext into a matrix (grid) based on the number of rows and columns defined by the key. If the plaintext doesn’t fill the matrix completely, you may need to pad it with a filler character (like 'X').
3. **Rearrange Based on the Key**: The key determines the order in which the columns (or rows) will be read to form the ciphertext. For example, if the key is a number representing the order of columns, you would read the columns in that order.

**Example:**

* **Plaintext**: "HELLO WORLD"
* **Key**: 3 (indicating a 3-column matrix)
* **Reading Order**: Based on the key, the columns are read in a specific order (e.g., 2nd column, 1st column, 3rd column).
* **Ciphertext**: "EOLHROLD"

**Weakness**: Like other transposition ciphers, the Row Column Transformation Cipher can be vulnerable to frequency analysis and can be broken if the arrangement pattern is known.

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// Row and Column Transformation Encryption

string rowColumnEncrypt(string text, vector<int> key)

{

    int n = key.size();

    int paddedLen = text.length() + (n - text.length() % n) % n;

    text.append(paddedLen - text.length(), 'X');

    vector<string> grid;

    for (int i = 0; i < text.length(); i += n)

    {

        grid.push\_back(text.substr(i, n));

    }

    string encryptedText;

    for (int i : key)

    {

        for (const string &row : grid)

        {

            encryptedText += row[i - 1];

        }

    }

    return encryptedText;

}

// Row and Column Transformation Decryption

string rowColumnDecrypt(string cipher, vector<int> key)

{

    int n = key.size();

    int numRows = cipher.length() / n;

    vector<string> grid(numRows, string(n, ' '));

    int index = 0;

    for (int i : key)

    {

        for (int j = 0; j < numRows; ++j)

        {

            grid[j][i - 1] = cipher[index++];

        }

    }

    string decryptedText;

    for (const string &row : grid)

    {

        decryptedText += row;

    }

    while (decryptedText.back() == 'X')

    {

        decryptedText.pop\_back();

    }

    return decryptedText;

}

int main()

{

    string plaintext = "Sourabh Patil";

    vector<int> key = {3, 1, 4, 2};

    string encryptedText = rowColumnEncrypt(plaintext, key);

    string decryptedText = rowColumnDecrypt(encryptedText, key);

    cout << "\nRow and Column Transformation:\nEncrypted: " << encryptedText << "\nDecrypted: " << decryptedText << endl;

    return 0;

}

