Cryptography, Network and Security

Assignment 2

2. Perform encryption and decryption using following transposition techniques

a. Rail fence

b. row and Column Transformation

Code:

a.

```cpp
#include <iostream>
#include <string>
using namespace std;

// Function to encrypt using Rail Fence Cipher
string railFenceEncrypt(string plaintext, int rails)
{
    string matrix[rails];

    int row = 0;
    bool down = true;

    for (int i = 0; i < plaintext.length(); i++)
    {
        matrix[row].push_back(plaintext[i]);

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }
```

```
    string ciphertext = "";
    for (int i = 0; i < rails; i++)
    {
        ciphertext += matrix[i];
    }
    return ciphertext;
}

// Function to decrypt using Rail Fence Cipher
string railFenceDecrypt(string ciphertext, int rails)
{
    string matrix[rails];

    int length = ciphertext.length();
    int row = 0, index = 0;
    bool down = true;

    bool mark[length][rails];
    for (int i = 0; i < rails; i++)
        for (int j = 0; j < length; j++)
            mark[j][i] = false;

    for (int i = 0; i < length; i++)
    {
        mark[i][row] = true;

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }

    for (int i = 0; i < rails; i++)
    {
```

```
        for (int j = 0; j < length; j++)
        {
            if (mark[j][i])
            {
                matrix[i].push_back(ciphertext[index++]);
            }
        }
    }

    row = 0;
    down = true;
    string plaintext = "";
    for (int i = 0; i < length; i++)
    {
        plaintext += matrix[row][0];
        matrix[row].erase(matrix[row].begin());

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }

    return plaintext;
}

int main()
{
    cout<<"Enter the text to be encrypted: ";
    string plaintext;
    getline(cin, plaintext);

    int rails = 3;
```

```
    string encrypted = railFenceEncrypt(plaintext, rails);
    cout << "Encrypted Text (Rail Fence): " << encrypted << endl;

    string decrypted = railFenceDecrypt(encrypted, rails);
    cout << "Decrypted Text (Rail Fence): " << decrypted << endl;

    return 0;
}
```

b.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Function to generate the column order based on the key
vector<int> getColumnOrder(string key)
{
    vector<pair<char, int>> keyWithIndices;
    for (int i = 0; i < key.size(); i++)
    {
        keyWithIndices.push_back({key[i], i});
    }

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

    vector<int> order;
    for (auto &p : keyWithIndices)
    {
        order.push_back(p.second);
    }

    return order;
}

// Function to encrypt using Row and Column Transformation Cipher
string rowColumnEncrypt(string plaintext, string key)
{
    int columns = key.size();
    int rows = (plaintext.size() + columns - 1) / columns;

    while (plaintext.size() < rows * columns)
    {
        plaintext += 'X';
    }
```

```
    vector<vector<char>> matrix(rows, vector<char>(columns));
    int index = 0;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = plaintext[index++];
        }
    }

    vector<int> columnOrder = getColumnOrder(key);

    string ciphertext = "";
    for (int col : columnOrder)
    {
        for (int i = 0; i < rows; i++)
        {
            ciphertext += matrix[i][col];
        }
    }
    return ciphertext;
}

// Function to decrypt using Row and Column Transformation Cipher
string rowColumnDecrypt(string ciphertext, string key)
{
    int columns = key.size();
    int rows = (ciphertext.size() + columns - 1) / columns;

    vector<vector<char>> matrix(rows, vector<char>(columns));

    vector<int> columnOrder = getColumnOrder(key);

    int index = 0;
    for (int col : columnOrder)
    {
        for (int i = 0; i < rows; i++)
        {
            matrix[i][col] = ciphertext[index++];
        }
    }

    string plaintext = "";
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            plaintext += matrix[i][j];
```

```
        }
    }
    while (plaintext.back() == 'X')
    {
        plaintext.pop_back();
    }

    return plaintext;
}

int main()
{
    string plaintext;
    cout << "Enter the text to be encrypted: ";
    getline(cin>>ws, plaintext);
    string key;
    cout << "Enter the key: ";
    cin >> key;

    string encrypted = rowColumnEncrypt(plaintext, key);
    cout << "Encrypted Text (Row-Column): " << encrypted << endl;

    string decrypted = rowColumnDecrypt(encrypted, key);
    cout << "Decrypted Text (Row-Column): " << decrypted << endl;

    return 0;
}
```