**Experiment-4**

**Aim:** Study and Implement a program for Rail Fence Transposition Cipher to encrypt and decrypt the message.

**Introduction:**

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

**Encryption:**

In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.

When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus, the alphabets of the message are written in a zig-zag manner.

After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

**Decryption:**

As we have seen in encryption, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

Hence, rail matrix can be constructed accordingly. Once we have got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).

Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

**Program (Source Code):**

//Note: in railfence cipher, spaces ARE considered, so do not truncate them.

#include <bits/stdc++.h>

using namespace std;

//logic: make a matrix of rail \* length(plainText) dimensions, and traverse the string and fill the matrix

string encrypt(string plainText, int rail){

    string encryptedText;

    int len = plainText.length();

    char matrix[rail][len];

    //filling the matrix with '-'

    for (int i=0;i<rail;i++){

        for (int j=0;j<len;j++){

            matrix[i][j] = '-';

        }

    }

    int flag = 1;   //flag = 1 means increment the value of  rows, flag = 0 means decrement the value of rows. [Initially set to 1]

    int row = 0;

    for (int i=0;i<len;i++){    // traversing the plainText. [value of i will be the column number]

        //setting the flag (to choose whether to increment or decrement)

        if (row == 0){

            flag = 1;

        }

        else if (row == (rail-1)){

            flag = 0;

        }

        else{

            //do nothing

        }

        //push the value

        matrix[row][i] = plainText[i];

        //choosing the next row value in the matrix [update value of row]

        if (flag == 1){

            row++;

        }

        else{

            row--;

        }

    }

    // //printing the matrix

    // cout<<"\nMatrix:\n";

    // for (int i=0;i<rail;i++){

    //     for (int j=0;j<len;j++){

    //         cout<<matrix[i][j]<<" ";

    //     }

    //     cout<<"\n";

    // }

    //making the encrypted text

    for (int i=0;i<rail;i++){

        for (int j=0;j<len;j++){

            if (matrix[i][j] != '-'){

                encryptedText += matrix[i][j];

            }

        }

    }

    return encryptedText;

}

//logic: make the matrix and fill the places (where the encrypted text characters will be placed) with \* and then traverse the matrix again and fill it with ciphertext wherever there are \*

string decrypt(string encryptedText, int rail){

    string decryptedText;

    int len = encryptedText.length();

    char matrix[rail][len];

    //filling the matrix with '-'

    for (int i=0;i<rail;i++){

        for (int j=0;j<len;j++){

            matrix[i][j] = '-';

        }

    }

    //filling the places in the matrix with '\*' wherever the encrypted text characters need to be put (same way as done in encryption)

    int flag = 1;

    int row = 0;

    for (int i=0;i<len;i++){

        if (row == 0){

            flag = 1;

        }

        else if (row == rail-1){

            flag = 0;

        }

        else{

            //do nothing

        }

        matrix[row][i] = '\*';

        if (flag == 1){

            row++;

        }

        else{

            row--;

        }

    }

    // //printing the matrix

    // cout<<"\nMatrix:\n";

    // for (int i=0;i<rail;i++){

    //     for (int j=0;j<len;j++){

    //         cout<<matrix[i][j]<<" ";

    //     }

    //     cout<<"\n";

    // }

    //traversing through the matrix and filling it with the encrypted text wherever there is a '\*'

    int stringPointer = 0;

    for (int i=0;i<rail;i++){

        for (int j=0;j<len;j++){

            if (matrix[i][j] == '\*'){

                matrix[i][j] = encryptedText[stringPointer++];

            }

        }

    }

    //forming the decrypted text the same way we put the '\*' (traversing method is same) [this is the same way encryption was done, but now we're reading it instead of writing it]

    flag = 1;

    row = 0;

    for (int i=0;i<len;i++){

        if (row == 0){

            flag = 1;

        }

        else if (row == rail-1){

            flag = 0;

        }

        else{

            //do nothing

        }

        decryptedText += matrix[row][i];

        if (flag == 1){

            row++;

        }

        else{

            row--;

        }

    }

    return decryptedText;

}

int main(){

    string plainText = "My name is Ravi and I study in PDEU";

    int rail = 4;

    cout<<"Plain Text: "<<plainText;

    cout<<"\nRail: "<<rail;

    string encryptedText = encrypt(plainText, rail);

    cout<<"\nEncrypted Text: "<<encryptedText;

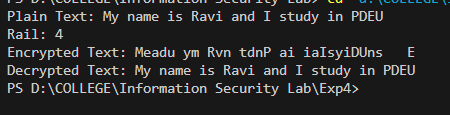
    string decryptedText = decrypt(encryptedText, rail);

    cout<<"\nDecrypted Text: "<<decryptedText;

    return 0;

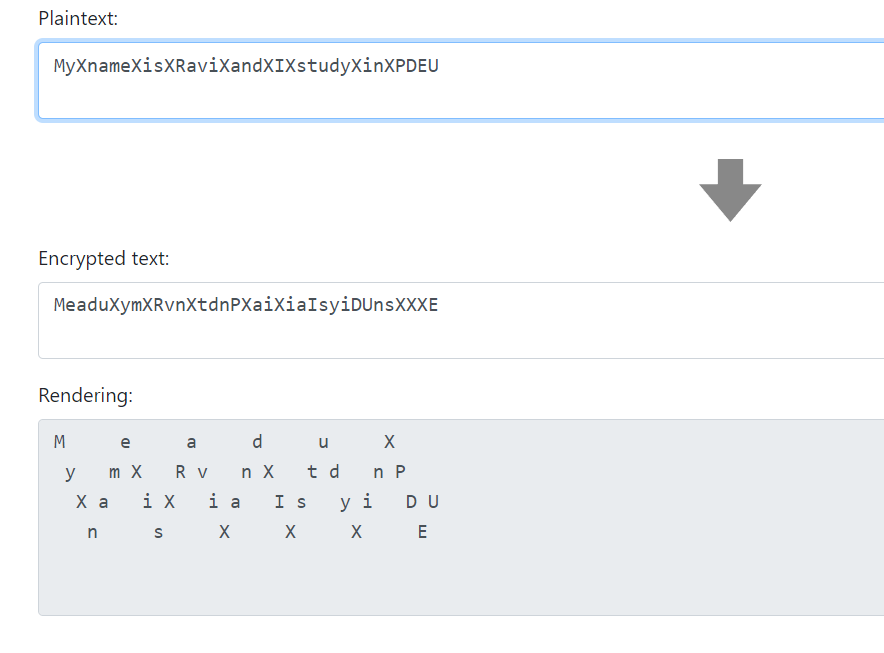
}

**Output (Program):**

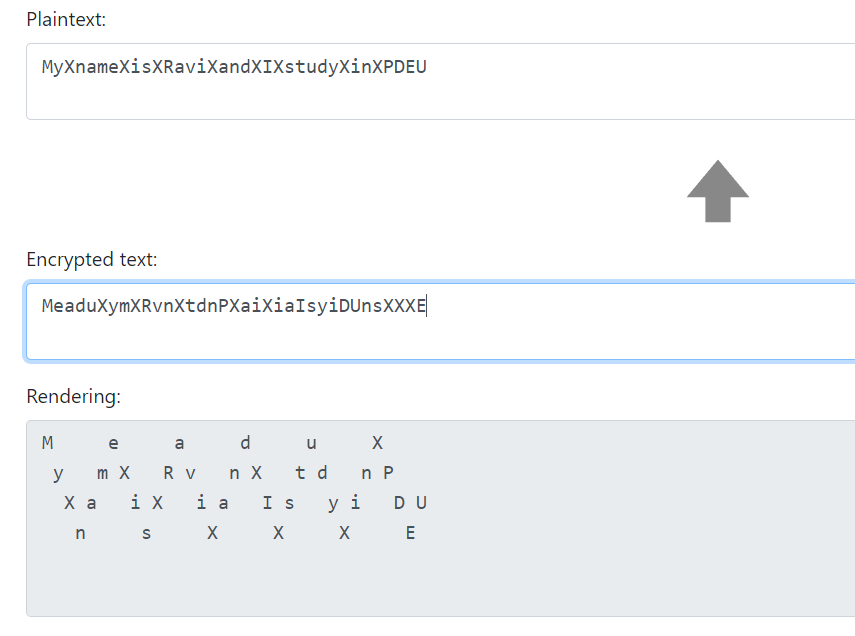
****

**Output (Cryptool):**

**Encryption:**

****

**Decryption:**



**Cryptanalysis:**

1. **Brute Force Attack:** The cipher's key is N, the number of rails. If N is known, the ciphertext can be decrypted by using the above algorithm. Values of N equal to or greater than L, the length of the ciphertext, are not usable, since then the ciphertext is the same as the plaintext. Therefore, the number of usable keys is low, allowing the brute-force attack of trying all possible keys. As a result, the rail-fence cipher is considered weak. We need a way of figuring out which of the keys results in the most English like plaintext after decryption. The key that results in a decryption with the highest likelihood of being English text is most probably the correct key. So, the method used is to take the ciphertext, try decrypting it with each key, then see which decryption looks the best.

**Applications:**

The Rail Fence cipher, despite its simplicity and susceptibility to cryptanalysis, can still find some limited applications where strong security is not a primary concern. Here are a few examples of where the Rail Fence cipher might be used:

1. **Educational Purposes:** The Rail Fence cipher is often used as an introductory example in cryptography courses. It helps students understand basic concepts of encryption, transposition ciphers, and the importance of key management.
2. **Puzzle and Games:** The Rail Fence cipher can be employed as part of puzzles, games, or challenges in recreational settings. It can add an element of fun and intrigue to activities where participants need to decipher messages.
3. **Steganography:** While not secure for cryptographic purposes, the Rail Fence cipher could be used as a form of steganography, where messages are hidden in plain sight within seemingly innocent communication. This could be a casual way of conveying messages without attracting attention.
4. **Historical Interest:** The Rail Fence cipher has a historical significance as one of the early encryption techniques. It might be used in reenactments or historical educational settings to demonstrate how encryption was practiced in the past.
5. **Artistic Expression:** Some artists and writers might use the Rail Fence cipher to encode messages within their work, adding an extra layer of meaning or playfulness to their creations.
6. **Basic Communication:** In scenarios where very, basic security is required (e.g., leaving a note for a family member or friend), the Rail Fence cipher could be used to prevent casual readers from understanding the message immediately.
7. **Ciphers for Children:** The Rail Fence cipher's simple structure makes it suitable for introducing young children to the concept of encryption in a playful manner.

**References:**

1. GeeksforGeeks
2. www.practicalcryptography.com
3. crypto.interactive-maths.com
4. en.wikipedia.org