Assignment No. 02

Encryption and Decryption Using Transposition Ciphers

Objective:

- To understand and implement encryption and decryption using the Rail Fence cipher.
- To understand and implement encryption and decryption using the Row and Column Transposition cipher.

A: Rail Fence Cipher

Theory:

The Rail Fence cipher is a form of transposition cipher that writes the plaintext in a zigzag pattern across multiple "rails" and then reads off each row to create the ciphertext.

Example with 3 rails:

Plaintext: **HELLO WORLD**

Write in rails:

H L O L
E L W R D
L O _

Read row-wise: **HLOLELWRDLO**

Steps:

Encryption:

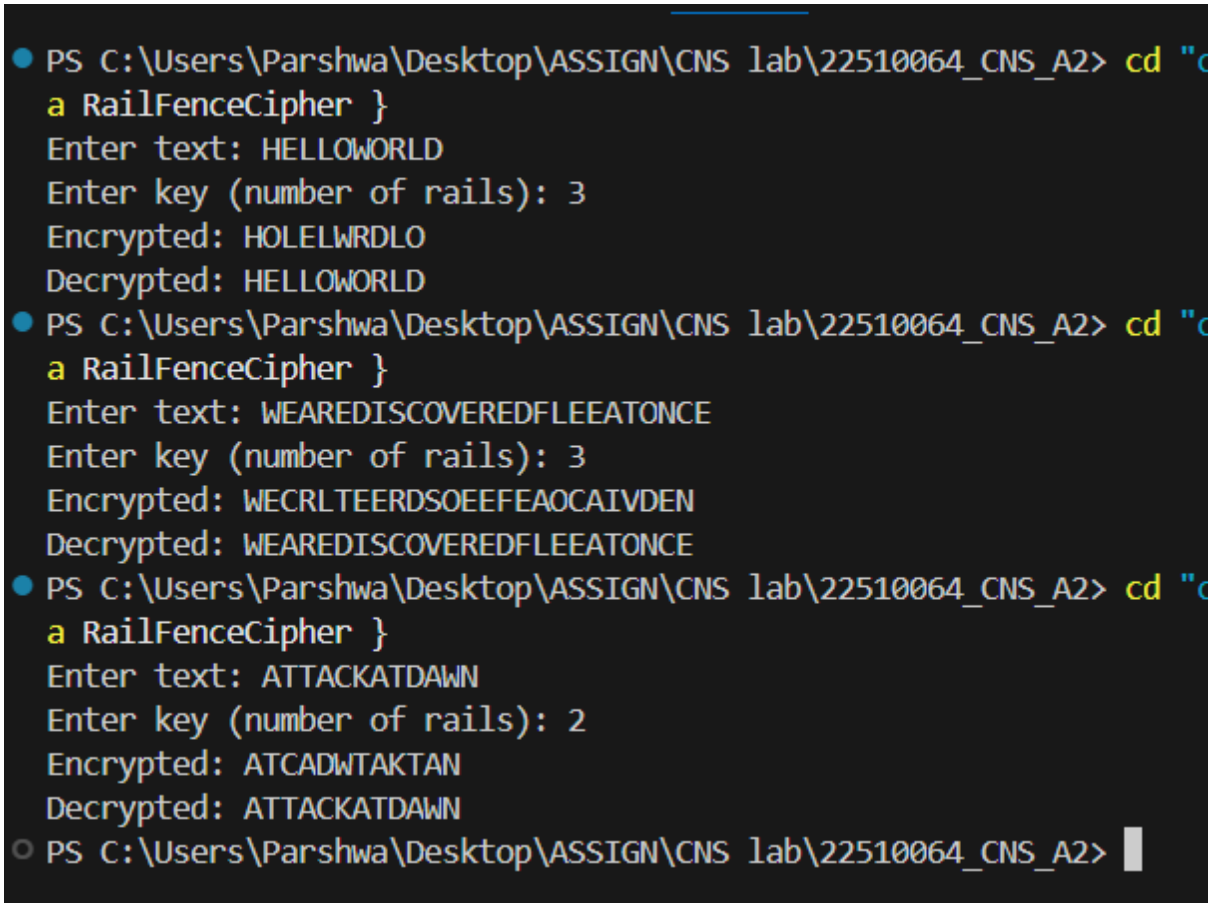
1. Choose the number of rails (key).
2. Write the plaintext in a zigzag pattern on rails.
3. Read the character's row-wise to get ciphertext.

Decryption:

1. Write the ciphertext row-wise in rails.
2. Reconstruct the zigzag pattern to retrieve original plaintext.

```java
import java.util.Scanner;

public class RailFenceCipher {

    public static String encrypt(String text, int key) {
        StringBuilder[] rails = new StringBuilder[key];
        for (int i = 0; i < key; i++) rails[i] = new StringBuilder();

            int dir = 1, row = 0;
            for (char c : text.toCharArray()) {
                rails[row].append(c);
                row += dir;
                if (row == key - 1) dir = -1;
                else if (row == 0) dir = 1;
            }
            StringBuilder result = new StringBuilder();
            for (StringBuilder rail : rails) result.append(rail);
            return result.toString();
        }

        public static String decrypt(String cipher, int key) {
            int[] railLengths = new int[key];
            int row = 0, dir = 1;

            for (int i = 0; i < cipher.length(); i++) {
                railLengths[row]++;
                row += dir;
                if (row == key - 1) dir = -1;
                else if (row == 0) dir = 1;
            }

            String[] rails = new String[key];
            int start = 0;
            for (int i = 0; i < key; i++) {
                rails[i] = cipher.substring(start, start + railLengths[i]);
                start += railLengths[i];
            }

```

```
            int[] railIndices = new int[key];
            StringBuilder result = new StringBuilder();
            row = 0; dir = 1;
            for (int i = 0; i < cipher.length(); i++) {
                result.append(rails[row].charAt(railIndices[row]++));
                row += dir;
                if (row == key - 1) dir = -1;
                else if (row == 0) dir = 1;
            }
            return result.toString();
        }

        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);

            System.out.print("Enter text: ");
            String text = sc.nextLine().replaceAll("\\s+", "").toUpperCase();

            System.out.print("Enter key (number of rails): ");
            int key = sc.nextInt();

            String encrypted = encrypt(text, key);
            System.out.println("Encrypted: " + encrypted);

            String decrypted = decrypt(encrypted, key);
            System.out.println("Decrypted: " + decrypted);

            sc.close();
        }
    }

```



Observation:

1-The zigzag pattern distributes characters across 3 rails, and reading row-by-row shuffles them. Decryption perfectly restores original text.

2-The Rail Fence works well with longer text; the distribution is more uniform across rails. No padding is required since length fits naturally.

3-With 2 rails, the pattern is simpler (alternating characters). Useful for faster manual encryption but easier to break.

General Analysis for Rail Fence:

Works by rearranging order of characters without changing them.

Security depends on the number of rails (small keys are easier to crack).

Easy to implement and decrypt when key is known.

Not suitable for strong security in modern use.

B: Row and Column Transposition Cipher

Theory:

The Row and Column transposition cipher arranges the plaintext into a matrix and then permutes the columns based on a key to get ciphertext.

Steps:

Encryption:

1. Write the plaintext in rows of a matrix (number of columns depends on key length).
2. Rearrange columns according to the alphabetical order of the key.
3. Read the matrix column-wise to get ciphertext.

Decryption:

1. Write ciphertext column-wise based on the key order.
2. Rearrange columns back to the original key order.
3. Read rows to get plaintext.

Example:

- Key: **ZEBRA** (Assign numerical order based on alphabetical: A=1, B=2, E=3, R=4, Z=5)
- Plaintext: **WE ARE DISCOVERED FLEE AT ONCE**

Conclusion:

- Rail Fence cipher uses zigzag pattern for transposition.
- Row and Column cipher rearranges characters in a matrix based on a key.
- Both ciphers provide a basic introduction to transposition techniques.

```java
import java.util.*;

public class RowColumnTransposition {

    // Generate column read order based on the key sequence
    public static int[] getOrder(int[] key) {
        int[] order = new int[key.length];
        Integer[] idx = new Integer[key.length];
        for (int i = 0; i < key.length; i++) idx[i] = i;

        Arrays.sort(idx, Comparator.comparingInt(i -> key[i]));

        for (int i = 0; i < key.length; i++) {
            order[i] = idx[i];
        }
        return order;
    }

    public static String encrypt(String text, int[] key) {
        int cols = key.length;
        int rows = (int) Math.ceil((double) text.length() / cols);
        char[][] matrix = new char[rows][cols];
        int index = 0;

        // Fill row-wise
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                matrix[r][c] = (index < text.length()) ?
text.charAt(index++) : 'X';
            }
        }

        StringBuilder result = new StringBuilder();
        int[] order = getOrder(key);

        // Read columns in sorted key order
        for (int colIndex : order) {
            for (int r = 0; r < rows; r++) {
                result.append(matrix[r][colIndex]);
            }
        }
        return result.toString();
```

```
    }

    public static String decrypt(String cipher, int[] key) {
        int cols = key.length;
        int rows = (int) Math.ceil((double) cipher.length() / cols);
        char[][] matrix = new char[rows][cols];
        int index = 0;

        int[] order = getOrder(key);

        // Fill columns in sorted key order
        for (int colIndex : order) {
            for (int r = 0; r < rows; r++) {
                matrix[r][colIndex] = cipher.charAt(index++);
            }
        }

        StringBuilder result = new StringBuilder();
        // Read row-wise
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                result.append(matrix[r][c]);
            }
        }
        return result.toString().replaceAll("X+$", "");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter text: ");
        String text = sc.nextLine().replaceAll("\\s+",
"").toUpperCase();

        System.out.print("Enter key length: ");
        int keyLength = sc.nextInt();

        int[] key = new int[keyLength];
        System.out.println("Enter key sequence (1-based column
order): ");
        for (int i = 0; i < keyLength; i++) {
            key[i] = sc.nextInt();
```

```
        }

        String encrypted = encrypt(text, key);
        System.out.println("Encrypted: " + encrypted);

        String decrypted = decrypt(encrypted, key);
        System.out.println("Decrypted: " + decrypted);

        sc.close();
    }
}
```

```
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A2> cd "
) { java RowColumnTransposition }
Enter text: WEAREDISCOVEREDFLEEATONCE
Enter key length: 5
Enter key sequence (1-based column order):
3 1 4 2 5
Encrypted: EIELORCEECWDVFTASRENEODAE
Decrypted: WEAREDISCOVEREDFLEEATONCE
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A2> cd "
) { java RowColumnTransposition }
Enter text: HELLOWORLD
Enter key length: 3
Enter key sequence (1-based column order):
2 1 3
Encrypted: EORXHLODLWLX
Decrypted: HELLOWORLD
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A2> cd "
) { java RowColumnTransposition }
Enter text: DEFENDTHEEASTWALLOFTHECASTLE
Enter key length: 4
Enter key sequence (1-based column order):
4 3 1 2
Encrypted: FTAAFCLEHSLTAEEDEWOETDNETLHS
Decrypted: DEFENDTHEEASTWALLOFTHECASTLE
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A2>
```

1-Observation: Padding 'X' is added at the end to fill the last row. The key sequence changes column order, producing a more scrambled ciphertext.

2-Padding 'X' again ensures the final matrix is complete. Smaller keys lead to simpler permutations.

3-Column order drastically changes placement of letters, producing a ciphertext that is harder to guess without knowing the exact key order.

General Analysis for Row & Column:

Stronger than Rail Fence for the same text length because of key-based permutation.

Padding may be required to complete the matrix.

More resistant to simple frequency analysis but still vulnerable to known-plaintext attacks.