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.

Python Implementation:

```
def rail_fence_encrypt(text, key):
    rail = [['\n' for i in range(len(text))] for j in range(key)]
    dir_down = False
    row, col = 0, 0

    for char in text:
        if row == 0 or row == key - 1:
            dir_down = not dir_down
        rail[row][col] = char
        col += 1
        row += 1 if dir_down else -1

    result = []
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return "".join(result)

def rail_fence_decrypt(cipher, key):
    rail = [['\n' for i in range(len(cipher))] for j in range(key)]
    dir_down = None
    row, col = 0, 0

    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
        rail[row][col] = '*'
        col += 1
        row += 1 if dir_down else -1

    index = 0
    for i in range(key):
        for j in range(len(cipher)):
            if rail[i][j] == '*' and index < len(cipher):
                rail[i][j] = cipher[index]
                index += 1

    result = []
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
```

```
        if rail[row][col] != '*':
            result.append(rail[row][col])
            col += 1

        row += 1 if dir_down else -1

    return "".join(result)

# Example:
plaintext = "HELLOWORLD"
key = 3

ciphertext = rail_fence_encrypt(plaintext, key)
print("Encrypted:", ciphertext)

decrypted = rail_fence_decrypt(ciphertext, key)
print("Decrypted:", decrypted)
```

Java Implementation

```
public class Rail Fence Cipher {

    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; // direction: 1 = down, -1 = up
        int 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) {
        boolean[] marked = new boolean[cipher.length()];
```

```
    int row = 0, dir = 1;

    // mark positions that would be filled
    for (int i = 0; i < cipher.length(); i++) {
        marked[i] = (row == 0 || row == key - 1) || (dir == 1 || dir == -1);
        row += dir;
        if (row == key - 1) dir = -1;
        else if (row == 0) dir = 1;
    }

    char[] result = new char[cipher.length()];
    int index = 0;

    // place characters into rails
    int[] railLengths = new int[key];
    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];
    row = 0;
    dir = 1;

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

    return new String(result);
}

public static void main(String[] args) {
    String plaintext = "WEAREDISCOVEREDFLEEATONCE";
    int key = 3;

    String encrypted = encrypt(plaintext, key);
```

```
        System.out.println("Encrypted (Rail Fence): " + encrypted);

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

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**

Python Implementation:

```
def create_order(key):
    order = sorted(list(key))
    return [order.index(k) + 1 for k in key]

def row_column_encrypt(plaintext, key):
    key_len = len(key)
```

```
    order = create_order(key)

    # Remove spaces and pad plaintext to fill matrix
    plaintext = plaintext.replace(" ", "")
    rows = len(plaintext) // key_len + (len(plaintext) % key_len != 0)
    matrix = [['X'] * key_len for _ in range(rows)]  # Padding with X

    # Fill matrix row-wise
    index = 0
    for r in range(rows):
        for c in range(key_len):
            if index < len(plaintext):
                matrix[r][c] = plaintext[index]
                index += 1

    # Read columns in order of key
    ciphertext = ""
    for num in range(1, key_len + 1):
        col = order.index(num)
        for r in range(rows):
            ciphertext += matrix[r][col]

    return ciphertext

def row_column_decrypt(ciphertext, key):
    key_len = len(key)
    order = create_order(key)

    rows = len(ciphertext) // key_len
    matrix = [[''] * key_len for _ in range(rows)]

    # Fill matrix column-wise by order
    index = 0
    for num in range(1, key_len + 1):
        col = order.index(num)
        for r in range(rows):
            matrix[r][col] = ciphertext[index]
            index += 1

    # Read matrix row-wise
    plaintext = ""
    for r in range(rows):
        for c in range(key_len):
            plaintext += matrix[r][c]

    return plaintext.rstrip('X')

# Example:
plaintext = "WEAREDISCOVEREDFLEEATONCE"
key = "ZEBRA"
```

```
ciphertext = row_column_encrypt(plaintext, key)
print ("Encrypted:", ciphertext)

decrypted = row_column_decrypt (ciphertext, key)
print("Decrypted:", decrypted)
```

Java Implementation

```
import java.util.Arrays;

public class Row Column Transposition {

    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 the matrix row-wise
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (index < text.length()) {
                    matrix[r][c] = text.charAt(index++);
                } else {
                    matrix[r][c] = 'X'; // Padding character
                }
            }
        }

        StringBuilder result = new StringBuilder();

        // Read columns in the order of key
        for (int k = 0; k < cols; k++) {
            int col = key[k] - 1; // Assuming key is 1-based indexing
            for (int r = 0; r < rows; r++) {
                result.append(matrix[r][col]);
            }
        }
        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;
```

```
        // Fill columns based on key order
        for (int k = 0; k < cols; k++) {
            int col = key[k] - 1;
            for (int r = 0; r < rows; r++) {
                if (index < cipher.length()) {
                    matrix[r][col] = cipher.charAt(index++);
                } else {
                    matrix[r][col] = 'X';
                }
            }
        }

        StringBuilder result = new StringBuilder();

        // Read row-wise to get original text
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                result.append(matrix[r][c]);
            }
        }

        return result.toString();
    }

    public static void main(String[] args) {
        String plaintext = "WEAREDISCOVEREDFLEEATONCE";
        int[] key = {3, 1, 4, 2, 5}; // Column permutation key

        String encrypted = encrypt(plaintext, key);
        System.out.println("Encrypted (Row-Column): " + encrypted);

        String decrypted = decrypt(encrypted, key);
        System.out.println("Decrypted (Row-Column): " + decrypted);
    }
}
```

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.