Practical-5

Q1. Implement RailFence Cipher and Improvised RailFence Cipher

A1. **Rail Fence Cipher:**

The Rail Fence Cipher is a type of transposition cipher where the characters of the plaintext are arranged in a zigzag pattern across multiple "rails" or rows. The message is then read off row by row to create the ciphertext.

Steps in the Rail Fence Cipher:

1. **Matrix Construction:**
    - A matrix is created with a number of rows equal to the key (the number of rails) and columns equal to the length of the plaintext.
    - The plaintext is placed in the matrix in a zigzag pattern. The direction changes when the top or bottom of the matrix is reached.
2. **Encryption:**
    - Once the matrix is filled, the characters are read off row by row to generate the ciphertext.
3. **Decryption:**
    - The decryption process involves reversing the encryption steps.
    - The matrix is reconstructed by filling the zigzag pattern with placeholders and then replacing them with the characters from the ciphertext. The plaintext is then read in the zigzag order to retrieve the original message.

Code Explanation for Rail Fence Cipher:

- **Initialization:**
    - The class RailFence initializes with a key, which determines the number of rows (rails).
- **Matrix Creation (getRailFence method):**
    - The matrix is filled with the plaintext in a zigzag pattern using the key.
    - The dir_down boolean variable controls the direction of filling, switching direction at the top and bottom rails.
- **Encryption (encrypt method):**
    - The matrix is read row by row to create the ciphertext.
- **Decryption (decrypt method):**
    - The ciphertext is placed back into the matrix, filling the zigzag pattern.

- The plaintext is then reconstructed by reading the matrix in the zigzag order.

Improved Rail Fence Cipher:

The Improved Rail Fence Cipher is an extension of the basic Rail Fence Cipher with added complexity to increase security. In this version, the zigzag pattern is more elaborate, and the columns are shuffled based on a random seed generated from the key.

Steps in the Improved Rail Fence Cipher:

1. **Matrix Construction with Phases:**
   - The plaintext is placed in a matrix with the number of rows equal to the length of the plaintext and columns equal to the key.
   - The filling pattern is more complex, involving multiple phases that change the direction and column arrangement.
2. **Column Shuffling:**
   - After filling the matrix, the columns are shuffled randomly based on a seed derived from the key.
   - This step introduces an additional layer of complexity, making the ciphertext more secure.
3. **Encryption:**
   - The ciphertext is generated by reading the matrix column by column after the shuffle.
4. **Decryption:**
   - During decryption, the matrix is filled again in the shuffled column order.
   - The plaintext is reconstructed by reading the matrix in the order of the filling phases.

Code Explanation for Improved Rail Fence Cipher:

- **Initialization:**
  - The ImprovedRailFence class initializes with a key similar to the basic version.
- **Matrix Creation (getRailFence method):**
  - The matrix is filled with the plaintext in a more complex zigzag pattern involving different phases (phase_value controls the phase).
- **Column Shuffling:**
  - After constructing the matrix, columns are shuffled using Python's random.shuffle() with a seed set by the key.
- **Encryption (encrypt method):**

- The matrix is read in the shuffled column order to generate the ciphertext.

- **Decryption (decrypt method):**
  - The matrix is filled with the shuffled columns, and the plaintext is read by reversing the filling phases.

Conclusion:

The basic Rail Fence Cipher is a straightforward transposition cipher that rearranges the characters in a zigzag pattern, while the Improved Rail Fence Cipher adds complexity through multiple phases of filling and column shuffling, making it more secure. The improved version is harder to crack because of the randomization and the more intricate filling pattern.

Code:

```
import random

def printMat(mat):
    # Print the matrix row by row
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            if(mat[i][j] == "\t"):
                print("*\t", end="")
            else:
                print(mat[i][j], end="")
        print()

class RailFence:
    def __init__(self, key):
        self.key = key  # Initialize the key

    def setKey(self, key):
        self.key = key  # Set a new key

    def getRailFence(self, plaintext, key):
        # Create a matrix with 'key' rows and length of plaintext columns
        railFenceMat = [["\t" for i in range(len(plaintext))] for j in range(key)]

        dir_down = False  # Direction control
        row, col = 0, 0

        # Fill the matrix with characters in a zigzag pattern
        for i in range(len(plaintext)):
            if row == 0 or row == key - 1:
                dir_down = not dir_down  # Change direction at the top or bottom

            railFenceMat[row][col] = plaintext[i] + "\t"
            col += 1
```

```
            if dir_down:
                row += 1
            else:
                row -= 1

        return railFenceMat

    def encrypt(self, plaintext):
        # Get the Rail Fence matrix for the given plaintext
        railFenceMat = self.getRailFence(plaintext, self.key)

        ciphertext = ""

        # Read the matrix row by row to get the encrypted text
        for i in range(self.key):
            for j in range(len(plaintext)):
                if railFenceMat[i][j] != "\t":
                    ciphertext += railFenceMat[i][j][0]

        return ciphertext

    def decrypt(self, ciphertext):
        # Get the Rail Fence matrix for the given ciphertext
        railfenceMat = self.getRailFence(ciphertext, self.key)

        plaintext = ""

        dir_down = False
        row = 0
        col = 0
        lookahead = 0

        # Place the characters from the ciphertext back into the matrix
        for i in range(self.key):
            for j in range(len(ciphertext)):
                if railfenceMat[i][j] != "\t":
                    railfenceMat[i][j] = ciphertext[lookahead] + "\t"
                    lookahead += 1

        dir_down = False
        row = 0
        col = 0

        # Read the matrix in a zigzag pattern to decrypt the text
        for j in range(len(ciphertext)):
            if row == 0 or row == self.key - 1:
                dir_down = not dir_down

            if railfenceMat[row][col] != "\t":
                plaintext += railfenceMat[row][col][0]
```

```
                col += 1

            if dir_down:
                row += 1
            else:
                row -= 1

        return plaintext

class ImprovedRailFence:
    def __init__(self, key):
        self.key = key  # Initialize the key

    def setKey(self, key):
        self.key = key

    def getRailFence(self, plaintext, key):
        # Create a matrix with 'key' rows and length of plaintext columns
        railFenceMat = [["\t" for i in range(key)] for j in range(len(plaintext))]

        phase_value = 0
        end = False
        # Two Left To Right, then right to left


        row, col = 0, 0
        i = 0

        while not end:

            if phase_value == 0: # Left to Right, Top to Bottom
                for _ in range(self.key):
                    railFenceMat[row][col] = plaintext[i] + "\t"
                    i += 1
                    if i == len(plaintext): # If all characters are placed
                        end = True
                        break
                    col = (col + 1) % self.key # Change the column
                    row += 1
                phase_value = (phase_value + 1) % self.key # Change the phase value

            elif phase_value == 1: # Left to Right, Bottom to Top
                for _ in range(self.key):
                    railFenceMat[row][col] = plaintext[i] + "\t" # Place the
character in the matrix
                    i += 1
                    if i == len(plaintext):
                        end = True
                        break
                    col = (col + 1) % self.key # Change the column
                    row += 1
```

```
                phase_value = (phase_value + 1) % self.key # Change the phase value

            elif phase_value == 2: # Right to Left, Bottom to Top
                for _ in range(self.key):
                    col = self.key - 1
                    railFenceMat[row][col] = plaintext[i] + "\t"
                    i += 1
                    if i == len(plaintext):
                        end = True
                        break
                    col = (col - 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

            elif phase_value == 3: # Right to Left, Top to Bottom
                for _ in range(self.key):
                    col = self.key - 1
                    railFenceMat[row][col] = plaintext[i] + "\t"
                    i += 1
                    if i == len(plaintext):
                        end = True
                        break
                    col = (col - 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

        return railFenceMat

    def encrypt(self, plaintext):

        railFenceMat = self.getRailFence(plaintext, self.key) # Get the Rail Fence
matrix

        random.seed(self.key) # Seed the random number generator
        column_order = list(range(self.key)) # Create a list of columns
        random.shuffle(column_order) # Shuffle the columns

        cipher = ""

        for col in column_order: # Read the matrix column by column
            for row in range(len(plaintext)): # Read the matrix row by row
                if railFenceMat[row][col] != "\t": # If the character is not a
placeholder
                    cipher += railFenceMat[row][col][0] # Append the character to
the ciphertext

        return cipher

    def decrypt(self, ciphertext):

        temp = "*" * len(ciphertext) # Create a string of '*' characters
```

```
        railFenceMat = self.getRailFence(temp, self.key) # Get the Rail Fence
matrix

        random.seed(self.key) # Seed the random number generator
        column_order = list(range(self.key)) # Create a list of columns
        random.shuffle(column_order)     # Shuffle the columns

        plaintext = "" # Initialize the plaintext
        lookahead = 0 # Initialize the lookahead

        for col in column_order: # Read the matrix column by column
            for row in range(len(ciphertext)): # Read the matrix row by row
                if railFenceMat[row][col] == "*\t": # If the character is a
placeholder
                    railFenceMat[row][col] = ciphertext[lookahead] + "\t" # Replace
the placeholder with the character
                    lookahead += 1

        phase_value = 0 # Initialize the phase value
        end = False # Initialize the end flag
        row, col = 0, 0 # Initialize the row and column

        i = 0 # Initialize the index

        while not end:

            if phase_value == 0: # Left to Right, Top to Bottom
                for _ in range(self.key):
                    plaintext += railFenceMat[row][col][0] # Append the character
to the plaintext
                    i += 1
                    if i == len(ciphertext):
                        end = True
                        break
                    col = (col + 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

            elif phase_value == 1: # Left to Right, Bottom to Top
                for _ in range(self.key):
                    plaintext += railFenceMat[row][col][0] # Append the character
to the plaintext
                    i += 1
                    if i == len(ciphertext):
                        end = True
                        break
                    col = (col + 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

            elif phase_value == 2: # Right to Left, Bottom to Top
```

```
                for _ in range(self.key):
                    col = self.key - 1
                    plaintext += railFenceMat[row][col][0] # Append the character
to the plaintext
                    i += 1
                    if i == len(ciphertext):
                        end = True
                        break
                    col = (col - 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

            elif phase_value == 3: # Right to Left, Top to Bottom
                for _ in range(self.key):
                    col = self.key - 1
                    plaintext += railFenceMat[row][col][0] # Append the character
to the plaintext
                    i += 1
                    if i == len(ciphertext):
                        end = True
                        break
                    col = (col - 1) % self.key
                    row += 1
                phase_value = (phase_value + 1) % self.key

        return plaintext



# Example usage
rf = RailFence(3)

# Print the Rail Fence matrix for the given plaintext
print()
printMat(rf.getRailFence("HELLO", 3))

# Encrypt the plaintext
enc = rf.encrypt("HELLO")
print("\nEncrypted using Rail Fence: ", enc, end="\n")  # Output the encrypted text

# Decrypt the ciphertext
dec = rf.decrypt(enc)
print("\nDecrypted using Rail Fence: ", dec, end="\n")  # Output the decrypted text

imp = ImprovedRailFence(3) # Initialize the Improved Rail Fence object

print()
printMat(imp.getRailFence("HELLO", 3)) # Print the Rail Fence matrix for the given
plaintext
```

```
enc = imp.encrypt("HELLO") # Encrypt the plaintext
print("\nEncrypted using Improved Rail Fence: ", enc, end="\n") # Output the
encrypted text

dec = imp.decrypt(enc) # Decrypt the ciphertext
print("\nDecrypted using Improved Rail Fence: ", dec, end="\n") # Output the
decrypted text
```

Output:

```
H       *       *       *       O
*       E       *       L       *
*       *       L       *       *

Encrypted using Rail Fence:  HOELL

Decrypted using Rail Fence:  HELLO

H       *       *
*       E       *
*       *       L
L       *       *
*       O       *

Encrypted using Improved Rail Fence:  EOLHL

Decrypted using Improved Rail Fence:  HELLO
```