IS LAB_3

~ Prof. Dr. Aashka Raval

✅ Title 3: Implementation of Rail Fence Cipher (Standard)

🎯 Objective:

• To implement the classical **Rail Fence Cipher** encryption and decryption technique.
• To understand transposition-based symmetric encryption and how it affects the structure of plaintext without altering characters.

📘 Introduction:

The **Rail Fence Cipher** is a classical transposition cipher where the characters of the plaintext are written in a zigzag pattern across multiple "rails" (rows), and then read row by row to form the ciphertext. It is a basic example of transposition encryption, offering a clear understanding of how permutation enhances cryptographic strength without changing characters.

📚 Concepts Used:

• Symmetric Key Cryptography
• Transposition Cipher
• Zigzag Pattern Writing
• Rail Matrix Construction
• Reading by Rail for Encryption
• Zigzag Reconstruction for Decryption

🧠 Logic:

1. Encryption:

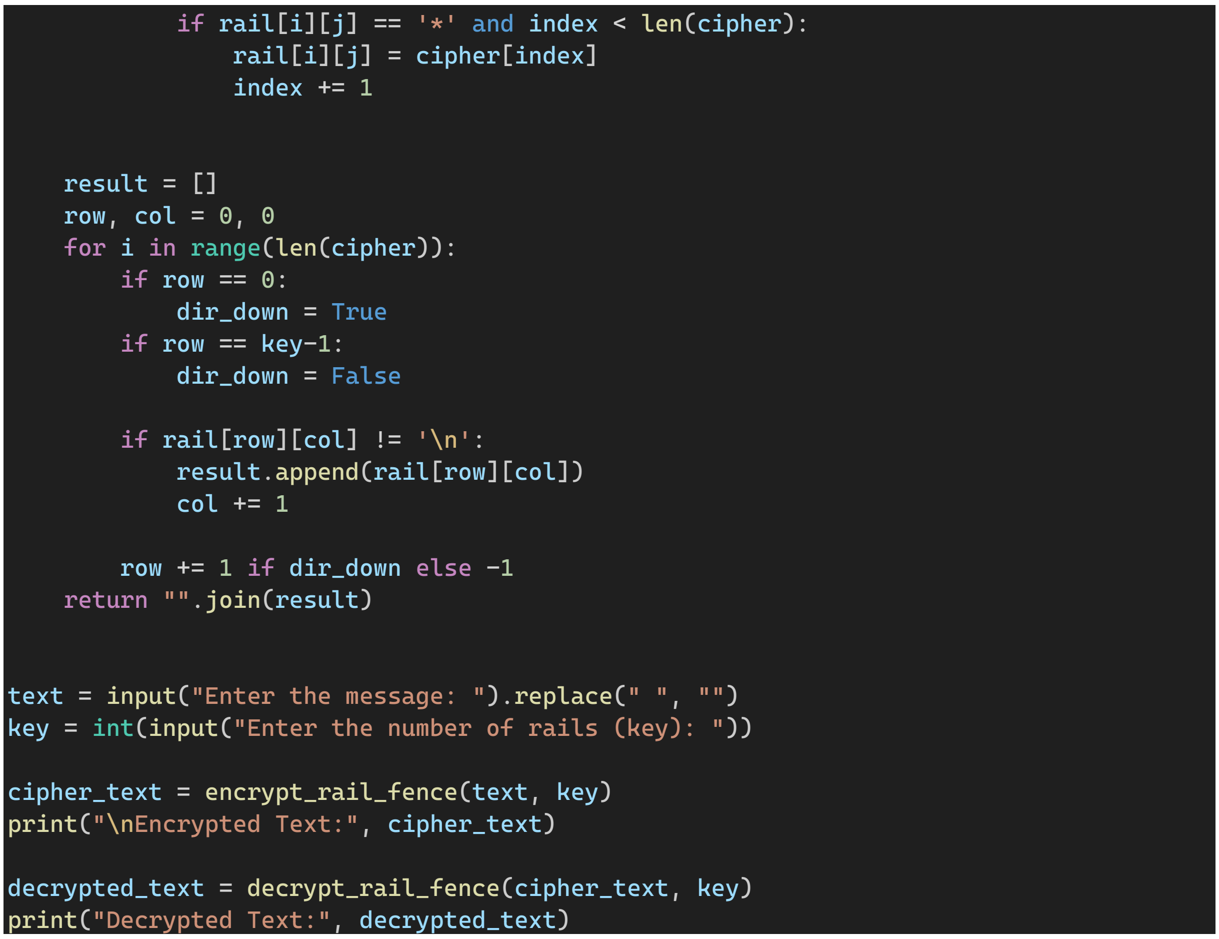
• Write the plaintext in a zigzag form across `n` rails.
• Start at the top rail, move downward rail by rail until the bottom is reached, then reverse and go upward.
• Fill characters in this pattern.
• Read the matrix row by row to get the ciphertext.

2. Decryption:

• Reconstruct the zigzag path used during encryption.
• Mark the positions where characters would go.
• Fill the matrix row by row with ciphertext.
• Traverse the matrix in zigzag again to extract the plaintext.

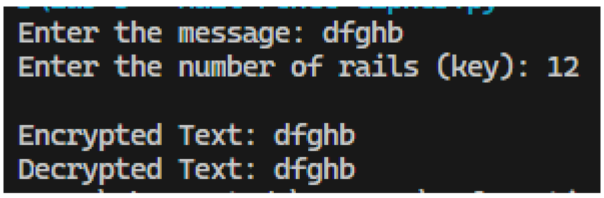
💻 Python Code:

```python
def encrypt_rail_fence(text, key):
    rail = [['\n' for i in range(len(text))]
              for j in range(key)]

    dir_down = False
    row, col = 0, 0

    for ch in text:
        if row == 0 or row == key - 1:
            dir_down = not dir_down

        rail[row][col] = ch
        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 decrypt_rail_fence(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] != '\n':
            result.append(rail[row][col])
            col += 1

        row += 1 if dir_down else -1
    return "".join(result)


text = input("Enter the message: ").replace(" ", "")
key = int(input("Enter the number of rails (key): "))

cipher_text = encrypt_rail_fence(text, key)
print("\nEncrypted Text:", cipher_text)

decrypted_text = decrypt_rail_fence(cipher_text, key)
print("Decrypted Text:", decrypted_text)
```

🧪 Sample Output:



```
Enter the message: dfghb
Enter the number of rails (key): 12

Encrypted Text: dfghb
Decrypted Text: dfghb
```

✅ Conclusion:

The **Rail Fence Cipher** demonstrates how plaintext structure can be altered through simple transposition without modifying the characters themselves. This experiment introduces the concept of confusion and diffusion in cryptography and highlights how reordering alone can secure data to a basic extent. It sets a strong foundation for understanding more advanced transposition-based encryption techniques.

✅ Title 2 (Modified): Implementation of 5x5 Playfair Cipher with Unicode/Multilingual Support

---

🎯 Objective:

• To enhance the classical Playfair Cipher to handle **Unicode characters and multilingual input**.
• To support encryption and decryption of texts from **languages beyond English**, such as Hindi, Gujarati, Japanese, etc.
• To explore how legacy ciphers can be adapted for modern communication scenarios.

📘 Introduction:

The traditional Playfair Cipher was designed for the 26-letter English alphabet and does not natively support Unicode. This modification modernizes the cipher to support any script or language supported by Unicode, making it relevant for multilingual use cases.
By removing the strict 5x5 constraint and using **dynamic grid sizing**, we can handle **custom character sets** based on user input.

📚 Concepts Used:

• Dynamic Matrix Construction
• Unicode Character Handling
• Digraph Substitution Logic
• Language-Agnostic Encryption/Decryption
• Python String and Set Manipulations

🧠 Logic:

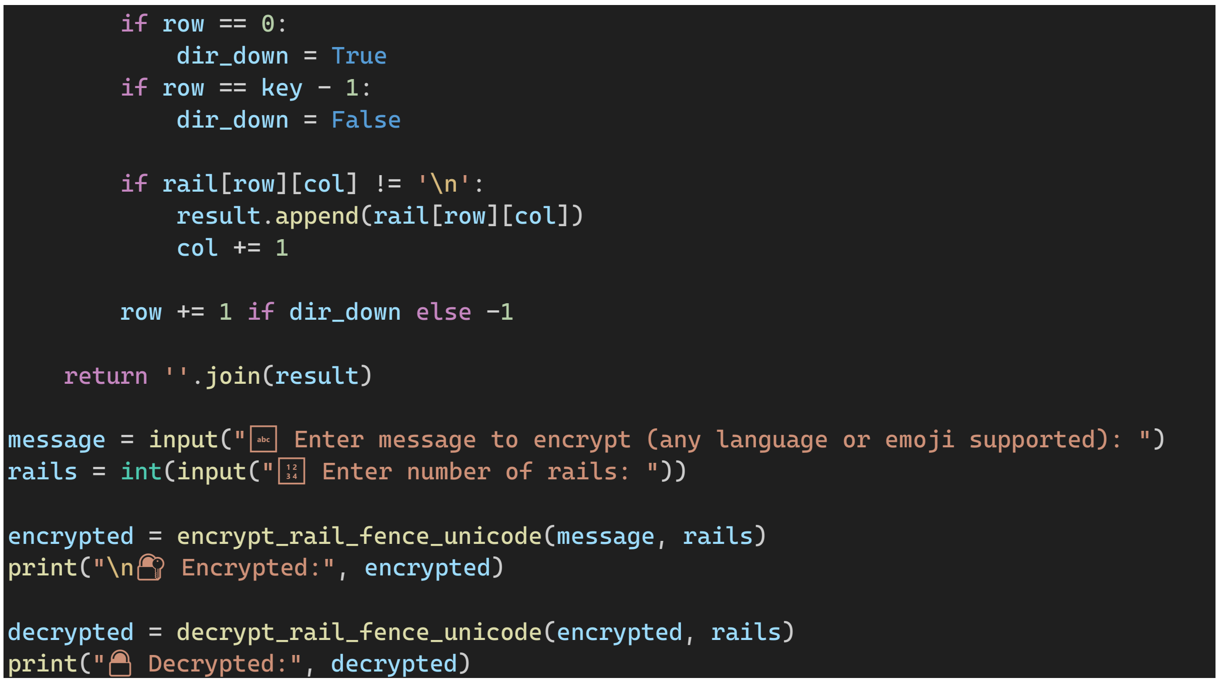
1. **Matrix Generation**
   • Remove duplicate characters from the key.
   • Build a list of unique characters used in both key and message.
   • Form an `N x N` matrix dynamically based on total unique characters (square root rounded up).
2. **Text Preprocessing**
   • Preserve Unicode characters (no filtering of English-only letters).
   • Break input into digraphs (pairs), inserting a filler (e.g., '⬛') when needed.
3. **Encryption/Decryption Rules (Same as classic Playfair)**
   • **Same Row**: Replace each character with the one to its immediate right.
   • **Same Column**: Replace each character with the one directly below.

• **Rectangle Rule**: Replace each with the character in the same row but column of the other.

4. **Dynamic Support**
   • Works with emojis, symbols, non-Latin languages, etc.
   • Matrix size adapts to characters in key and message.

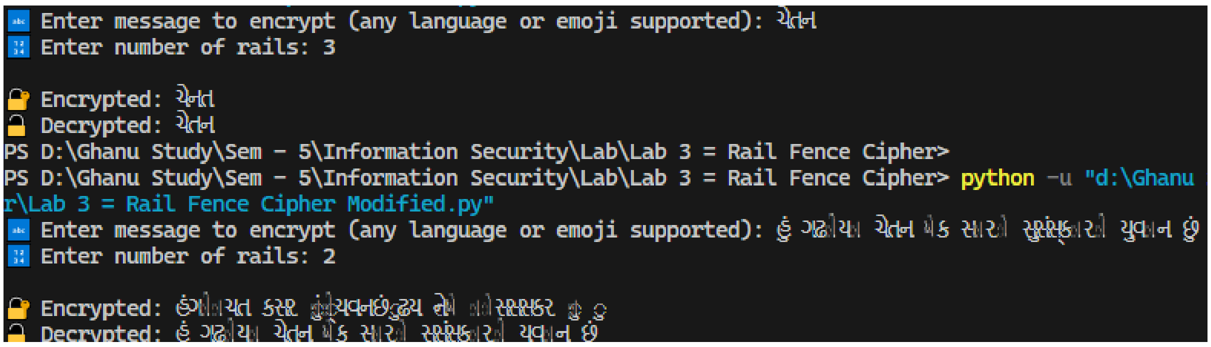
💻 Python Code:

```python
def encrypt_rail_fence_unicode(text, key):
    rail = [['\n' for _ in range(len(text))] for _ 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 decrypt_rail_fence_unicode(cipher, key):
    # Create the empty rail matrix
    rail = [['\n' for _ in range(len(cipher))] for _ in range(key)]

    # Mark the places with '*'
    dir_down = None
    row, col = 0, 0
    for _ 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 _ in range(len(cipher)):
```



```python
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False

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

        row += 1 if dir_down else -1

    return ''.join(result)

message = input("🔤 Enter message to encrypt (any language or emoji supported): ")
rails = int(input("🔢 Enter number of rails: "))

encrypted = encrypt_rail_fence_unicode(message, rails)
print("\n🔐 Encrypted:", encrypted)

decrypted = decrypt_rail_fence_unicode(encrypted, rails)
print("🔓 Decrypted:", decrypted)
```

🧪 Sample Output:



```
🔤 Enter message to encrypt (any language or emoji supported): ચેતન
🔢 Enter number of rails: 3

🔐 Encrypted: ચેનત
🔓 Decrypted: ચેતન
PS D:\Ghanu Study\Sem - 5\Information Security\Lab\Lab 3 = Rail Fence Cipher>
PS D:\Ghanu Study\Sem - 5\Information Security\Lab\Lab 3 = Rail Fence Cipher> python -u "d:\Ghanu
r\Lab 3 = Rail Fence Cipher Modified.py"
🔤 Enter message to encrypt (any language or emoji supported): હું ગઢોયા ચેતન એક સારો સુસંસ્કારી યુવાન છું
🔢 Enter number of rails: 2

🔐 Encrypted: [illegible]
🔓 Decrypted: હું ગઢોયા ચેતન એક સારો સુસંસ્કારી યુવાન છું
```

✅ Conclusion:

This modified Playfair Cipher demonstrates how classical encryption techniques can be enhanced for modern communication systems.
By supporting **Unicode and multilingual text**, it makes the cipher more versatile and inclusive, while preserving the logic of **pair-based symmetric encryption**.
