INTRODUCTION

It is a web application using Flask, a popular web framework in Python, to perform encryption and decryption using Rail Fence and Caesar Cipher techniques.This project is designed to showcase the combination of two encryption methods to enhance the security of plaintext messages. Rail Fence Cipher is a transposition cipher that rearranges the characters of the message in a zigzag pattern, and Caesar Cipher is a substitution cipher that shifts the characters by a fixed number of positions.

ABSTRACT

In this project, we aim to design and implement a secure communication system using two classical encryption techniques: Rail Fence and Caesar Cipher. These techniques have been widely used historically and are still relevant due to their simplicity and effectiveness. The project will involve developing a web application that allows users to encrypt and decrypt messages using these ciphers. It will also include a detailed explanation of the algorithms used, their strengths, and weaknesses, as well as the implementation details of the web application. The project will demonstrate how these ciphers can be combined to enhance the security of communication in a digital environment.

PURPOSE

The purpose of this project is to demonstrate the implementation and integration of Rail Fence Cipher and Caesar Cipher for secure message transmission. By combining these two encryption techniques, the project aims to provide an extra layer of security to plaintext messages, making it more challenging for unauthorized individuals to decipher the original content. The web application allows users to input a plaintext message, set encryption parameters such as Rail Fence key and Caesar shift, and then view the combined ciphertext and the decrypted result.

FEATURES

Rail Fence Cipher:

- **Zigzag Pattern:** The Rail Fence Cipher rearranges characters in a zigzag pattern, making it visually complex for anyone attempting to decrypt the message without the proper key.
- **Configurable Key:** Users can set the Rail Fence key, controlling the number of rails in the zigzag pattern, allowing for customization of the encryption process.

Caesar Cipher:

- **Substitution Encryption:** Caesar Cipher substitutes each character in the message by shifting it a fixed number of positions in the alphabet.
- **User-defined Shift:** Users can specify the Caesar shift, determining the amount by which characters are shifted during encryption.

Web Application:

- **Flask Framework:** The project utilizes the Flask web framework, providing a straightforward and efficient way to build web applications in Python.
- **User Interface:** The web application offers a user-friendly interface where users can input their plaintext and encryption parameters, and view the encrypted result along with the decryption outcome.

Combined Encryption:

- **Integration of Ciphers:** The project showcases the combination of Rail Fence Cipher and Caesar Cipher, enhancing the overall security of the encrypted message.
- **Encryption and Decryption Flow:** Users can observe the complete encryption and decryption flow, from Rail Fence encryption to Caesar encryption, and then reverse the process during decryption.

Educational Purpose:

- **Learning Encryption Techniques:** The project serves an educational purpose, helping users understand the concepts of Rail Fence and Caesar Cipher encryption and their integration.
- **Practical Implementation:** Users can see a practical implementation of cryptographic techniques, reinforcing their understanding of how these methods work in real-world scenarios.

IMPLEMENTATION

The web page is designed to be intuitive and accessible, accommodating users with varying levels of cryptography knowledge. Through a straightforward interface, users can input their plaintext, select the encryption or decryption mode, and specify the numeric key for the Caesar cipher along with the Rail fence for the category shift. The underlying algorithm processes the input, producing the corresponding ciphertext or decrypted text.

We use the visual studio code for create webpage

CODE:

app.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

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

    for i in range(len(text)):
        if (row == 0) or (row == key - 1):
            dir_down = not dir_down
        rail[row][col] = text[i]
        col += 1
        if dir_down:
            row += 1
        else:
            row -= 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 decryptRailFence(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
        if dir_down:
            row += 1
        else:
            row -= 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
        if dir_down:
            row += 1
```

```
        else:
            row -= 1
    return "".join(result)


def encrypt(text, s):
    result = ""

    # traverse text
    for i in range(len(text)):
        char = text[i]

        # Encrypt uppercase characters
        if char.isupper():
            result += chr((ord(char) + s - 65) % 26 + 65)

        # Encrypt lowercase characters
        else:
            result += chr((ord(char) + s - 97) % 26 + 97)

    return result
    # Your encryption code here

def decrypt(text, s):
    result = ""

    # traverse text
    for i in range(len(text)):
        char = text[i]

        # Decrypt uppercase characters
        if char.isupper():
            result += chr((ord(char) - s - 65) % 26 + 65)

        # Decrypt lowercase characters
        else:
            result += chr((ord(char) - s - 97) % 26 + 97)

    return result
```

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/result', methods=['POST'])
def result():
    plaintext = request.form['plaintext']
    rail_fence_key = int(request.form['rail_fence_key'])
    caesar_shift = int(request.form['caesar_shift'])

    # Encrypt using Rail Fence
    rail_fence_cipher = encryptRailFence(plaintext, rail_fence_key)

    # Encrypt using Caesar Cipher
    combined_cipher = encrypt(rail_fence_cipher, caesar_shift)

    # Decrypt using Caesar Cipher
    decrypted_caesar = decrypt(combined_cipher, caesar_shift)

    # Decrypt using Rail Fence
    decrypted_text = decryptRailFence(decrypted_caesar, rail_fence_key)

    return render_template('result.html', encrypted_text=combined_cipher,
decrypted_text=decrypted_text)

if __name__ == '__main__':
    app.run(debug=True)
```

In the Templates folder:

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Encryption and Decryption</title>
    <style>
        body {
            background-color: #1a1a1a;
            color: #00ff00;
            font-family: 'Courier New', monospace;
            text-align: center;
            padding: 20px;
        }

        h1 {
            color: #00ff00;
        }

        form {
            display: inline-block;
            text-align: left;
        }

        label {
            color: #00ff00;
            font-size: 18px;
        }

        input {
            padding: 8px;
            margin: 5px;
            font-size: 16px;
            border: 1px solid #00ff00;
            background-color: #1a1a1a;
```

```
            color: #00ff00;
        }

        input[type="submit"] {
            background-color: #00ff00;
            color: #1a1a1a;
            cursor: pointer;
        }

        input[type="submit"]:hover {
            background-color: #008000;
        }
    </style>
</head>
<body>
    <h1>Encryption and Decryption</h1>
    <form action="/result" method="POST">
        <label for="plaintext">Enter the plaintext:</label>
        <input type="text" id="plaintext" name="plaintext"><br><br>

        <label for="rail_fence_key">Enter the Rail Fence key:</label>
        <input type="number" id="rail_fence_key"
name="rail_fence_key"><br><br>

        <label for="caesar_shift">Enter the Caesar shift:</label>
        <input type="number" id="caesar_shift"
name="caesar_shift"><br><br>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

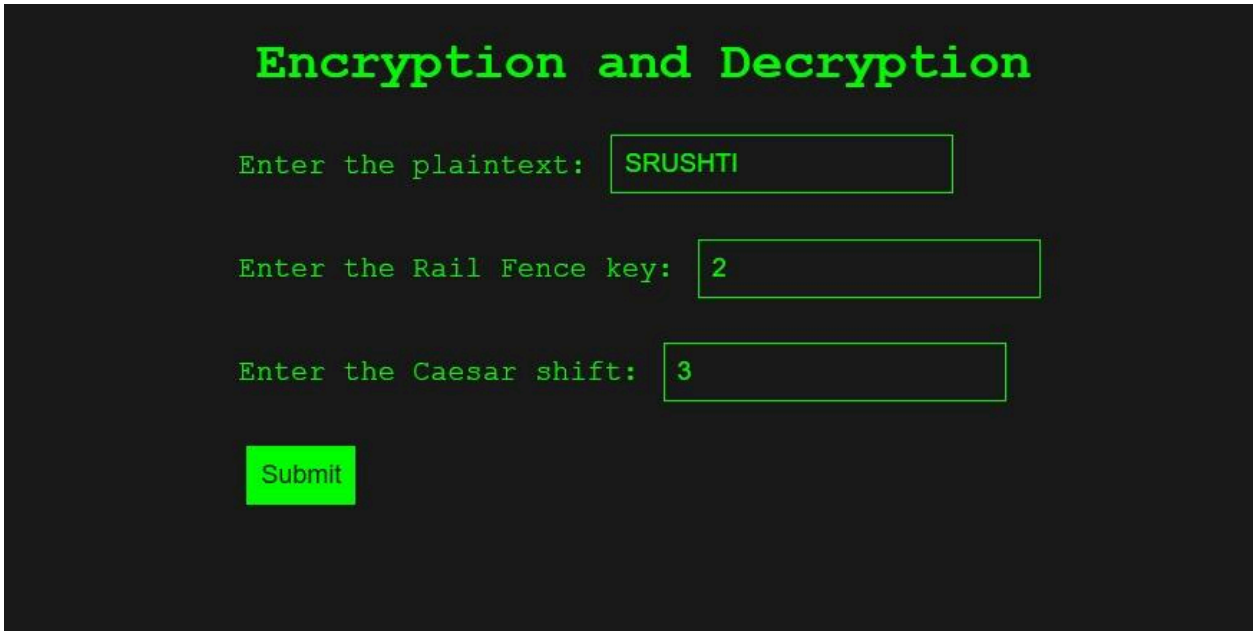
Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Result</title>
    <style>
        body {
            background-color: #1a1a1a;
            color: #00ff00;
            font-family: 'Courier New', monospace;
            text-align: center;
            padding: 20px;
        }

        h1 {
            color: #00ff00;
        }

        p {
            font-size: 18px;
        }
    </style>
</head>
<body>
    <h1>Result</h1>
    <p>Encrypted Text: {{ encrypted_text }}</p>
    <p>Decrypted Text: {{ decrypted_text }}</p>
</body>
</html>
```

Output:


Encryption and Decryption
Enter the plaintext:
SRUSHTI
Enter the Rail Fence key:
2
Enter the Caesar shift:
3
Submit


Result
Encrypted Text: VXKLUVW
Decrypted Text: SRUSHTI

CONCLUSION

This web application provides a basic example of how encryption can be implemented using Python and Flask. It showcases two classic ciphers, the Rail Fence Cipher and the Caesar Cipher, and demonstrates their use in encrypting and decrypting messages. The application can be extended to include more advanced encryption techniques or additional features, making it a useful starting point for learning about cryptography and web development with Flask.