Cryptography and Network Security Lab

Assignment 5.1  
Implementation and Understanding of Rail Fence Cipher

2019BTECS00058  
Devang K

Batch: B2

Title: Implementation and Understanding of Rail Fence Cipher

Aim: To Study, Implement and Demonstrate the Rail Fence Cipher Algorithm

Theory:

Rail fence cipher is a type of classical transpositional cipher which is also called zig-zag cipher. It gets name from the way in which it is encoded.

To encrypt using rail fence, we need the plaintext and the ‘number of rails’ i.e., the number of rows. The plaintext is written downwards diagonally on successive ‘rails’ – moving down till the last rail and then climbing up. It follows this until the plaintext is written out. Then, the Cipher-text is generated by traversing the rows.

To decrypt, we generate the table of the given fence size – length of size of encrypted text and height of the number of rails. We replace the position of a character with a special character – say ‘\*’. Then we traverse our encrypted text character-by-character and replace ‘\*’ with the encrypted-text character. Finally, we traverse the shape in the zig-zag order to get the corresponding plaintext.

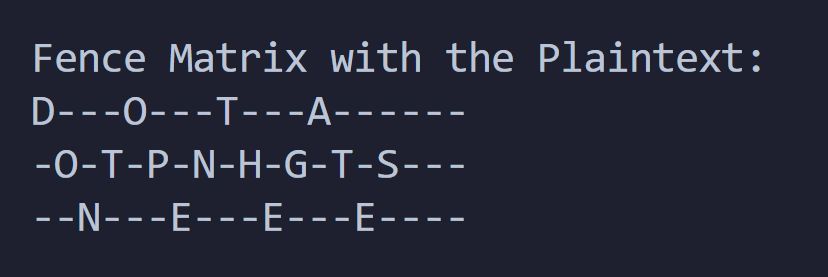
Illustration:

Encryption

Say we wish to encrypt – ‘ATTACKATONCE’ and say with 3 rails. We first make our matrix filled with blanks ‘-’ and then in the zig-zag pattern, plot our plaintext characters one-by-one. Finally, we then take append the characters going row-wise to generate our Cipher text.

Example:

Say, we wish to encrypt ‘DONTOPENTHEGATES’ with rail size 3. We first make the structure, and then plot the text in zig-zag pattern.

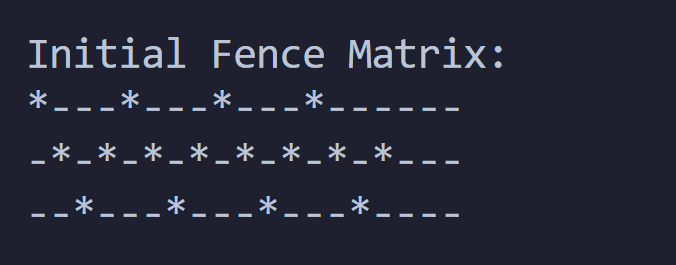


We then take the characters row-wise to generate the Cipher-text.

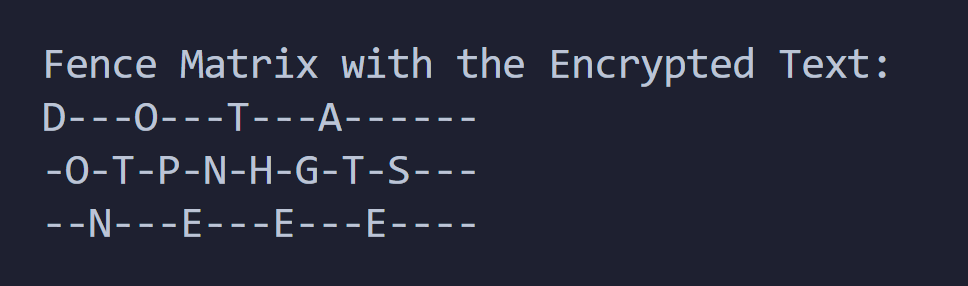
Here, it would become – ‘DOTAOTPNHGTSNEEE’.

Decryption

Say we wish to decrypt our Ciphertext of ‘DOTAXOTPNHGTSNEEE’ with the rail size of 3. We shall first make the structure of our fence, replacing the characters with stars.



Then we replace the ‘\*’ with the cipher-text character going row-wise. This would make it:

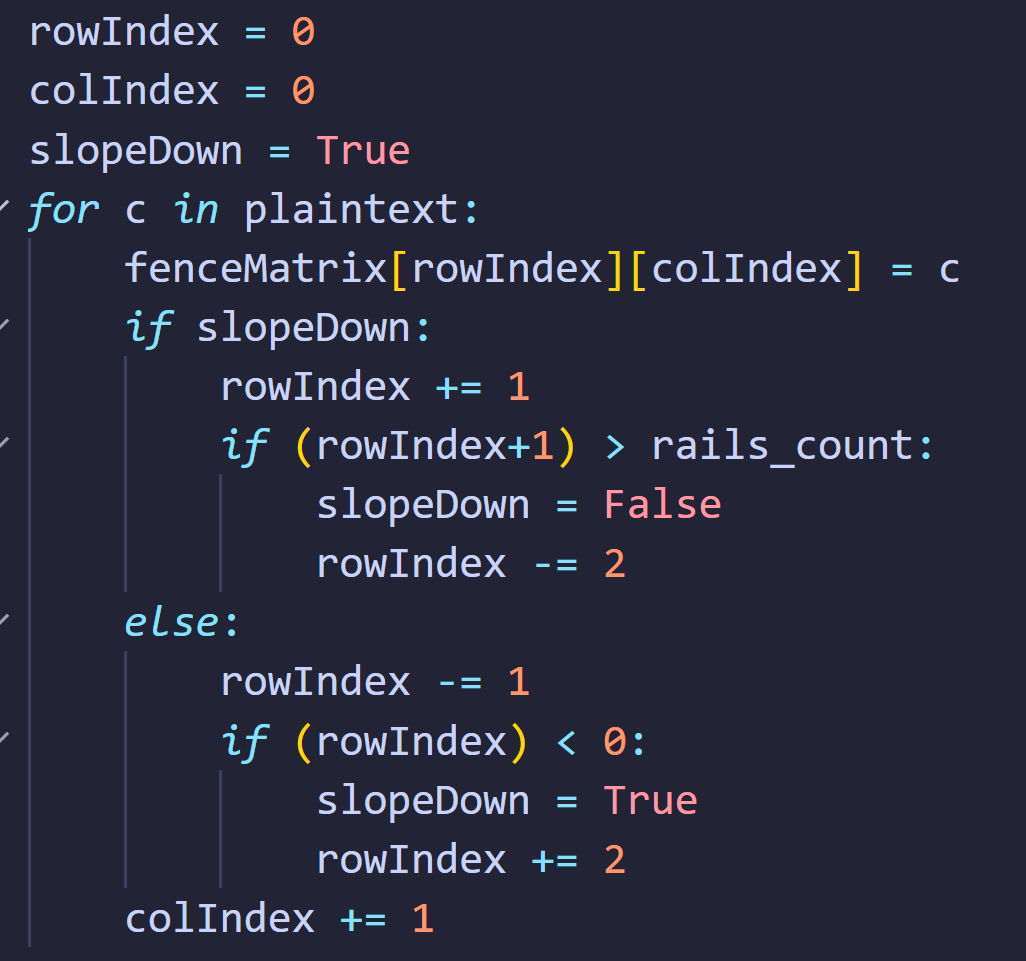


Now, in the pattern we made the ‘\*’, we shall traverse the fence to get our initial plaintext. Therefore, we get, ‘DONTOPENTHEGATES’.

We illustrate more examples using the code.

Code:

The crux of the code would be the code to traverse in the rail fence pattern. The function looks like:



Besides, the code to generate the matrix, traverse the elements and generating the cipher-text would be easy to implement. Similar is the case with decryption.

Code for Encryption:

*import* math

*def* printMatrix(fenceMatrix):

*for* i *in* range(len(fenceMatrix)):

*for* j *in* range(len(fenceMatrix[0])):

            print(fenceMatrix[i][j], end='')

        print()

*def* getEncryptedTextFromFenceMatrix(fenceMatrix):

    encText = ""

*for* i *in* range(len(fenceMatrix)):

*for* j *in* range(len(fenceMatrix[0])):

*if* fenceMatrix[i][j] != '-':

                encText += fenceMatrix[i][j]

*return* encText

print("Rail Fence Encrypt\n")

print("Enter Plaintext: ", end='')

plaintext = input()

print("Enter Fence Size: ", end='')

rails\_count = int(input())

fenceMatrix = []

*# No. of Rows -> No. of Rails*

*# No. of Cols -> Size of Plaintext + No. Rails*

noOfCols = len(plaintext) + rails\_count

*for* i *in* range(rails\_count):

    individualFence = []

*for* j *in* range(noOfCols):

        individualFence.append("-")

    fenceMatrix.append(individualFence)

print("\nInitial Fence Matrix:")

printMatrix(fenceMatrix)

rowIndex = 0

colIndex = 0

slopeDown = True

*for* c *in* plaintext:

    fenceMatrix[rowIndex][colIndex] = c

*if* slopeDown:

        rowIndex += 1

*if* (rowIndex+1) > rails\_count:

            slopeDown = False

            rowIndex -= 2

*else*:

        rowIndex -= 1

*if* (rowIndex) < 0:

            slopeDown = True

            rowIndex += 2

    colIndex += 1

*while* True:

    fenceMatrix[rowIndex][colIndex] = '-'

*if* slopeDown:

        rowIndex += 1

*if* (rowIndex+1) > rails\_count:

*break*

*else*:

        rowIndex -= 1

*if* (rowIndex) < 0:

*break*

    colIndex += 1

print("\nFence Matrix with the Plaintext:")

printMatrix(fenceMatrix)

print()

print("Plaintext:", plaintext)

print("Encrypted Text: ",getEncryptedTextFromFenceMatrix(fenceMatrix))

Code for Decryption:

*import* math

*def* printMatrix(fenceMatrix):

*for* i *in* range(len(fenceMatrix)):

*for* j *in* range(len(fenceMatrix[0])):

            print(fenceMatrix[i][j], end='')

        print()

print("Rail Fence Decrypt\n")

print("Enter Encrypted Text: ", end='')

encryptedText = input()

print("Enter Fence Size: ", end='')

rails\_count = int(input())

fenceMatrix = []

*# No. of Rows -> No. of Rails*

*# No. of Cols -> Size of encryptedText + No. Rails*

noOfCols = len(encryptedText) + rails\_count

*for* i *in* range(rails\_count):

    individualFence = []

*for* j *in* range(noOfCols):

        individualFence.append("-")

    fenceMatrix.append(individualFence)

*# We first mark the indices with '\*'*

rowIndex = 0

colIndex = 0

slopeDown = True

*for* c *in* range(len(encryptedText)):

    fenceMatrix[rowIndex][colIndex] = '\*'

*if* slopeDown:

        rowIndex += 1

*if* (rowIndex+1) > rails\_count:

            slopeDown = False

            rowIndex -= 2

*else*:

        rowIndex -= 1

*if* (rowIndex) < 0:

            slopeDown = True

            rowIndex += 2

    colIndex += 1

print("\nInitial Fence Matrix:")

printMatrix(fenceMatrix)

*# Replace \* with the Encrypted Text*

encTextIndex = 0

*for* i *in* range(rails\_count):

*for* j *in* range(noOfCols):

*if* fenceMatrix[i][j] == '\*':

            fenceMatrix[i][j] = encryptedText[encTextIndex]

            encTextIndex += 1

plaintext = ''

rowIndex = 0

colIndex = 0

slopeDown = True

*# Now we iterate in the rails and create the plain-text*

*for* c *in* range(len(encryptedText)):

    plaintext+=fenceMatrix[rowIndex][colIndex]

*if* slopeDown:

        rowIndex += 1

*if* (rowIndex+1) > rails\_count:

            slopeDown = False

            rowIndex -= 2

*else*:

        rowIndex -= 1

*if* (rowIndex) < 0:

            slopeDown = True

            rowIndex += 2

    colIndex += 1

print("\nFence Matrix with the Encrypted Text:")

printMatrix(fenceMatrix)

print()

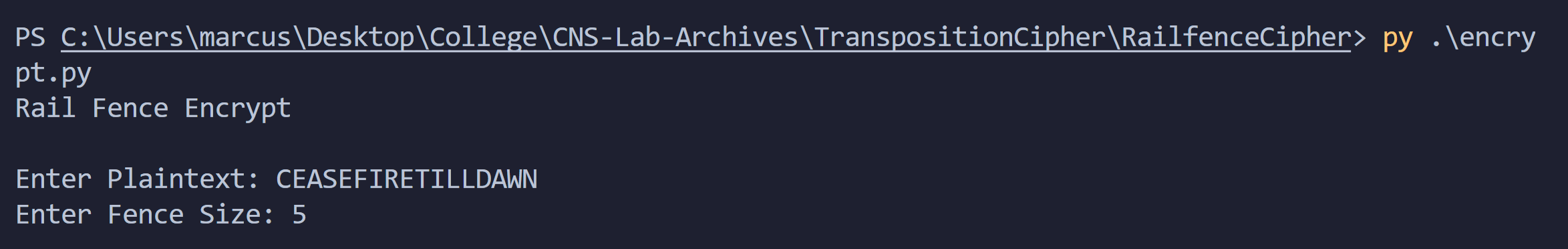
print("Encrypted Text:", encryptedText)

print("Plaintext:", plaintext)

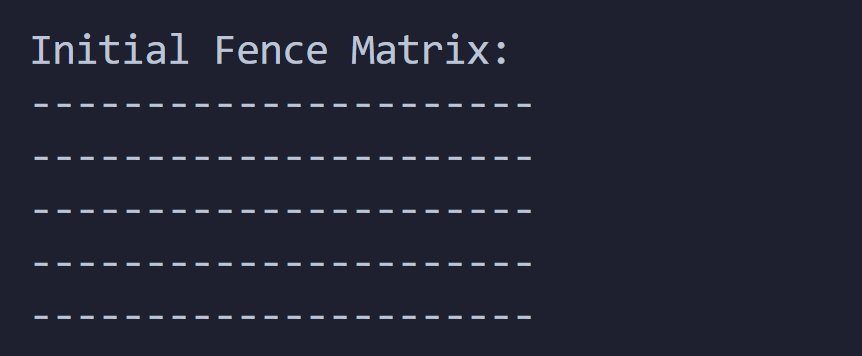
We now solve some examples with the code.

Say we wish to encrypt: ‘CEASEFIRETILLDAWN’  
and we take our rail-size as 5

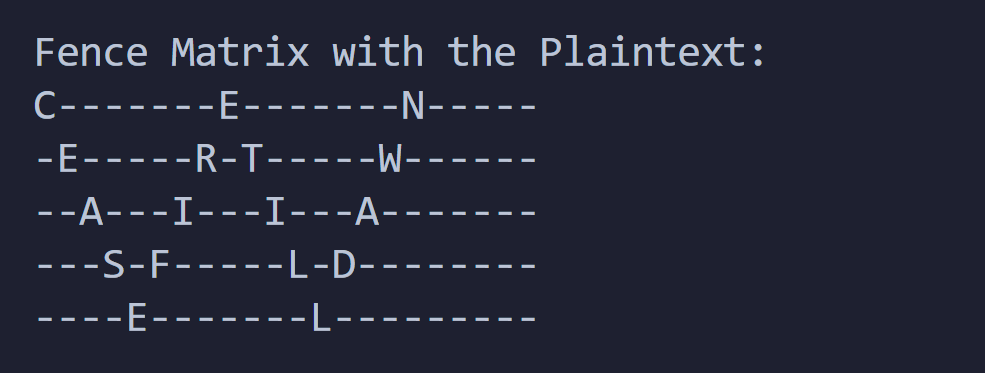
We run our code for encryption:



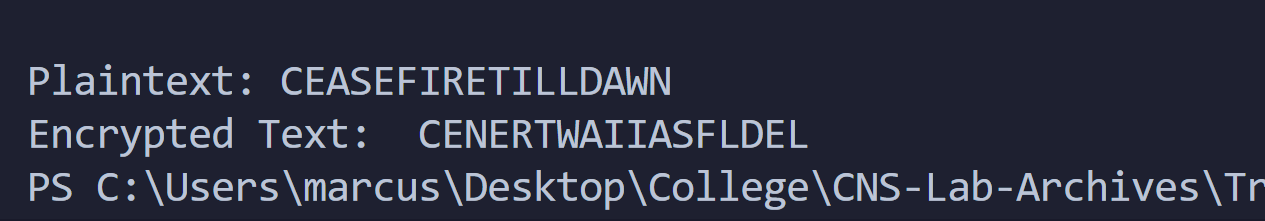
Then, it shows us our initial fence:



Then, it traverses and generates the zig-zag pattern:



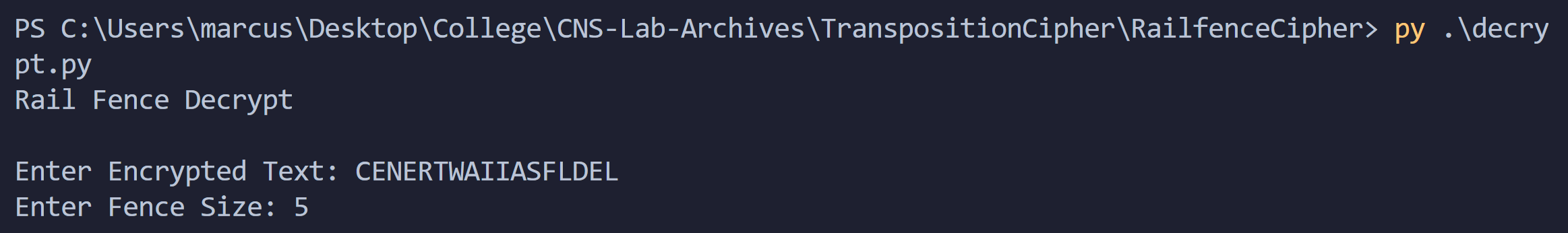
Finally, we get the encrypted string as:

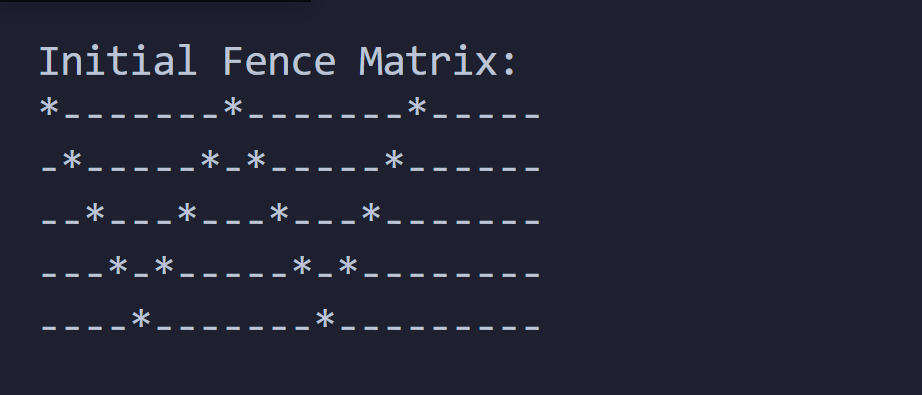


Therefore, the ciphertext for ‘CEASEFIRETILLDAWN’ with rail-size 5 would be ‘CENERTWAIIASFLDEL’.

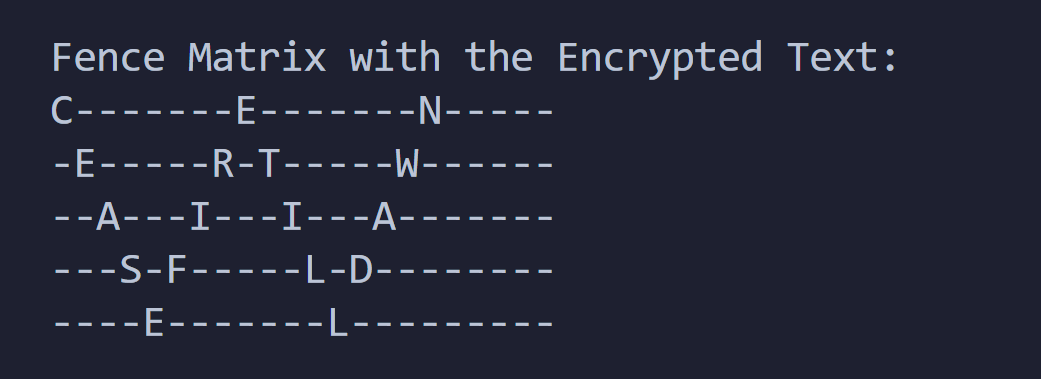
Now, we decrypt using the code.

We run using encrypted text ‘CENERTWAIIASFLDEL’ and rail-size 5.

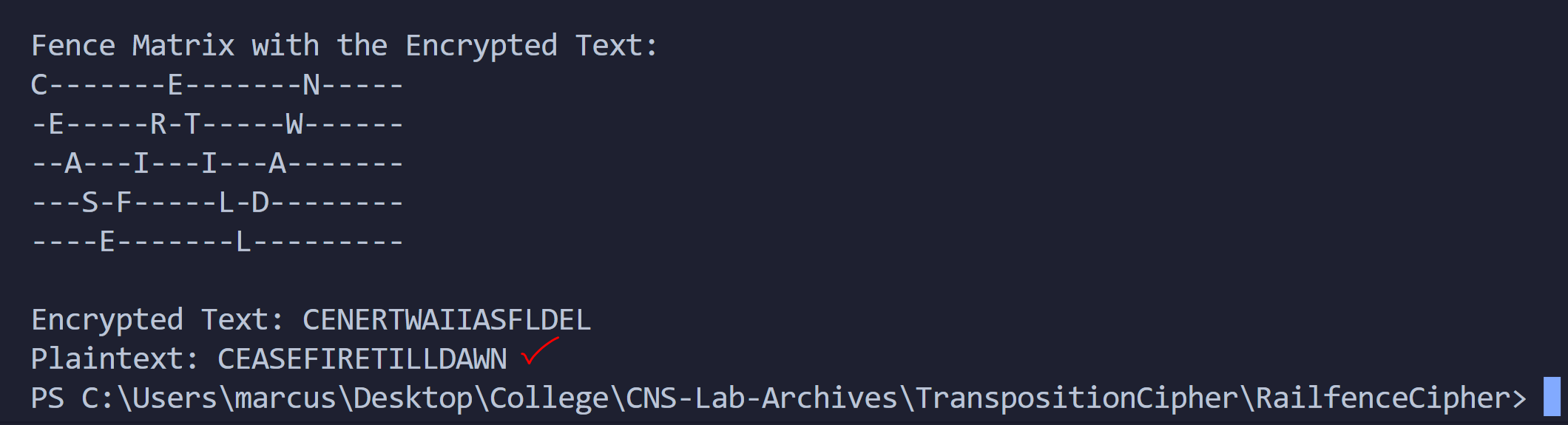


We look at our initial fence matrix:  


Then one that is filled with our encrypted text:



Finally, we get our plaintext:

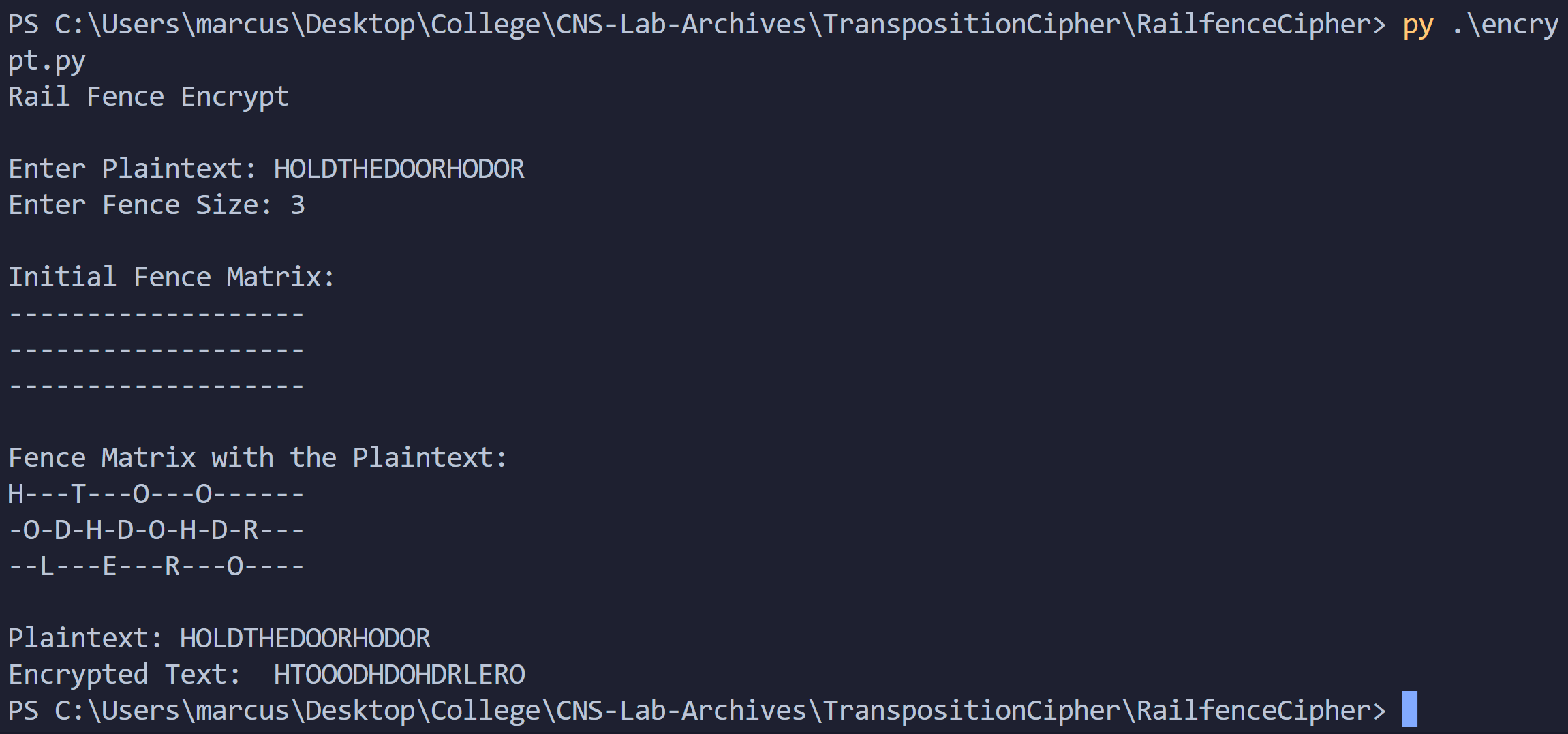


Therefore, decrypting ‘CENERTWAIIASFLDEL’ with rail-size 5 we get ‘CEASEFIRETILLDAWN’.

Let’s take another example:

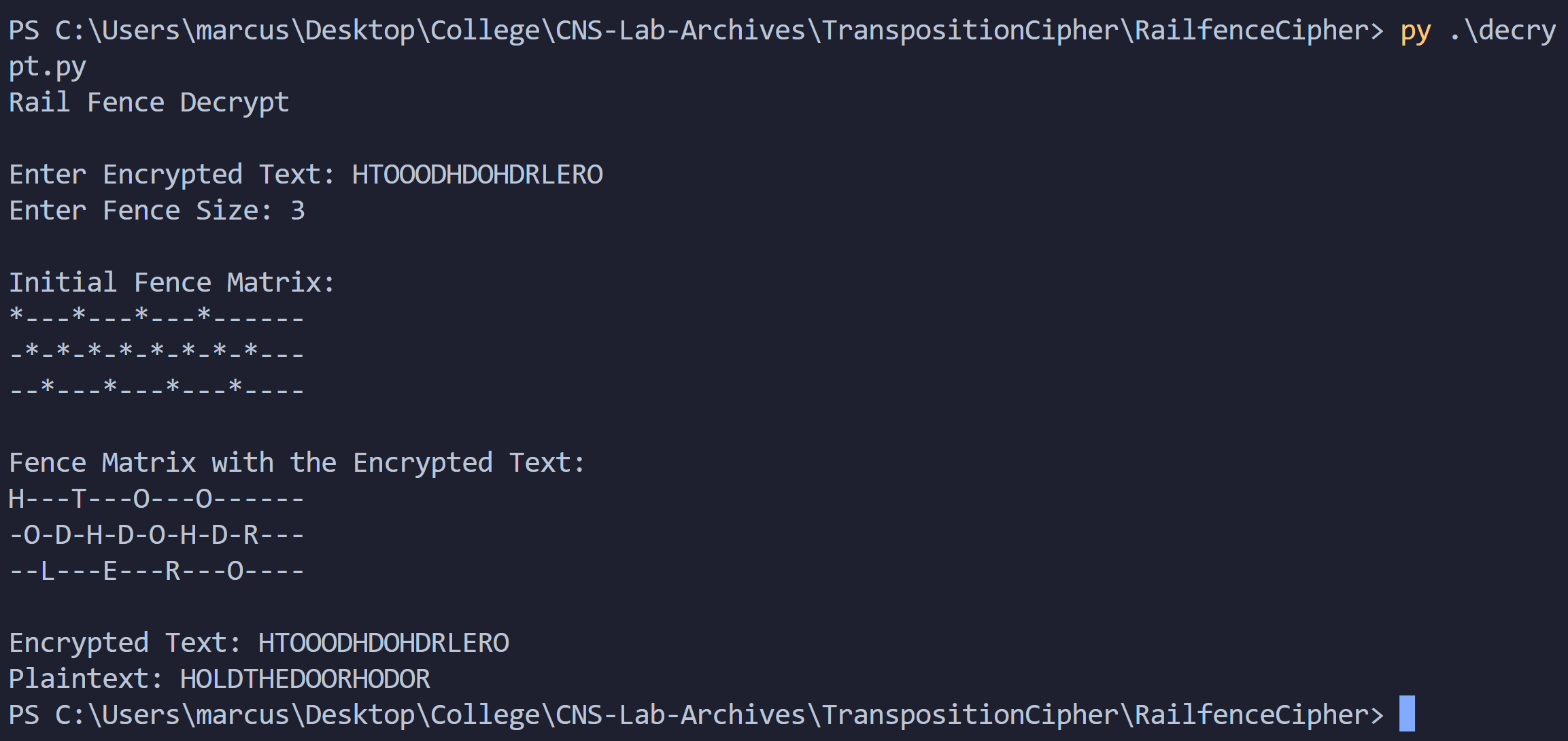
We wish to encrypt ‘HOLDTHEDOORHODOR’ with rail-size of 3.

Using our encryption code:



The encrypted text would be: ‘HTOOODHDOHDRLERO’.

Now, we decrypt:



We get our plaintext back.

Thus, we demonstrated the working of the code with examples.

Conclusion:

Thus, the Rail Fence Cipher algorithm was studied and demonstrated with the code. It is observed that it is a difficult algorithm to decipher when the number of words increases with a large value of key.