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.

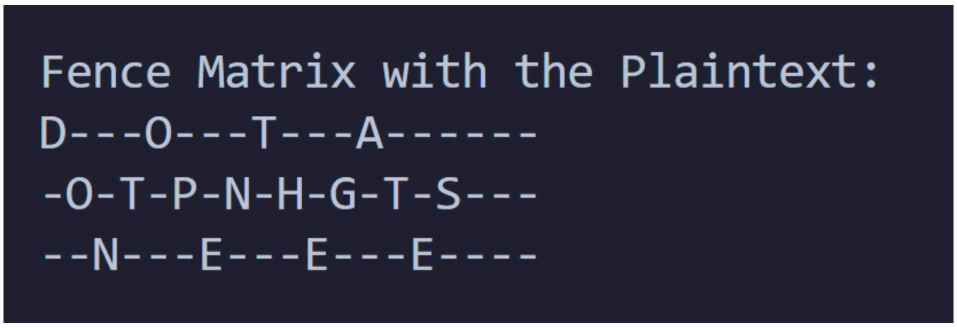
Illustration:

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.



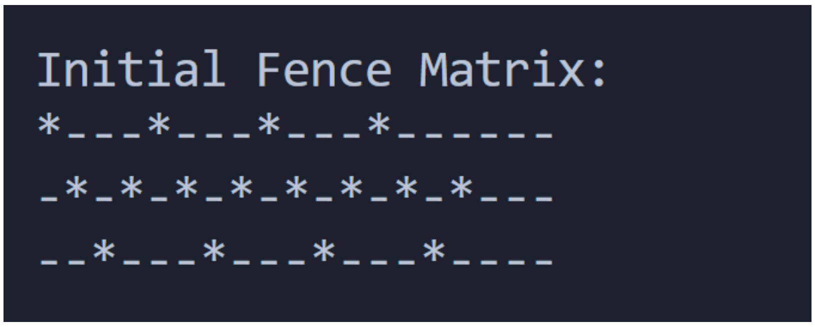
```
Fence Matrix with the Plaintext:
D---O---T---A------
-O-T-P-N-H-G-T-S---
--N---E---E---E----
```

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

Here, it would become – ‘DOTAOTPNHGTSNEEE’.

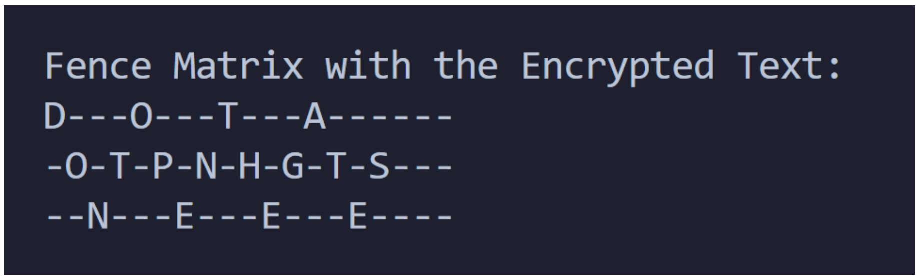
Decryption

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.



```
Initial Fence Matrix:
*___*___*___*______
_*_*_*_*_*_*_*_*___
__*___*___*___*____
```

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



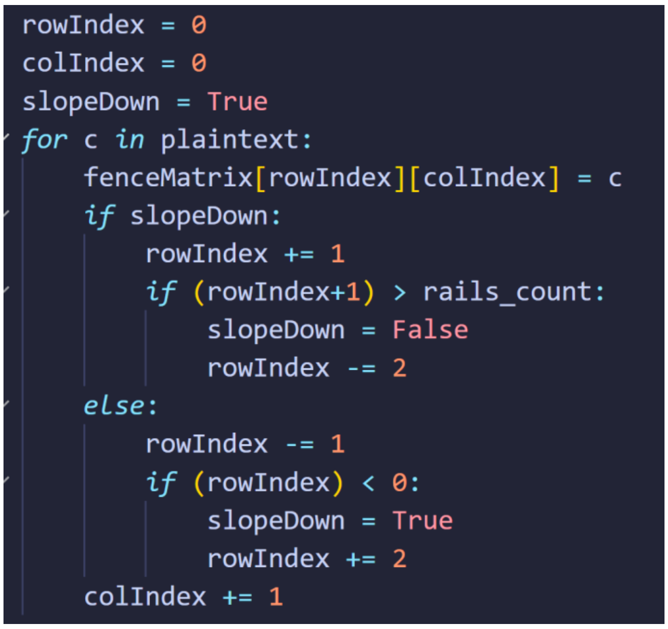
```
Fence Matrix with the Encrypted Text:
D---O---T---A------
-O-T-P-N-H-G-T-S---
--N---E---E---E----
```

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.

<u>Code</u>:

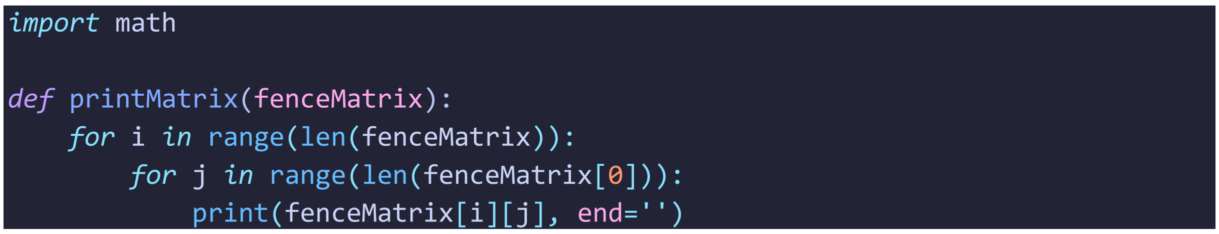
The crux of the code would be the code to traverse in the rail fence pattern. The function looks like:



```python
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
```

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:



```python
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
```

```python
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:

```python
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 '*'
```

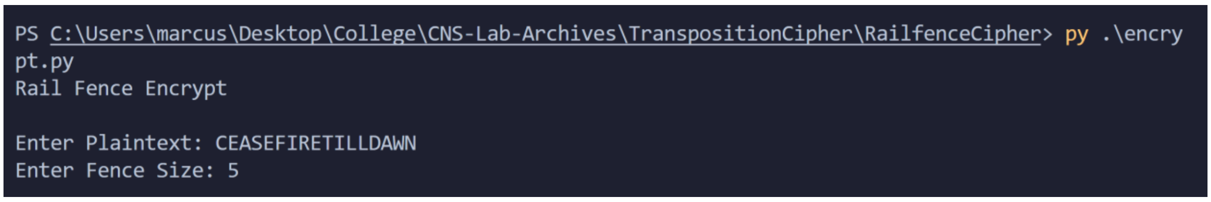
```python
rowIndex = 0
colIndex = 0
slopeDown = True
for c in range(len(encryptedText)):
    fenceMatrix[rowIndex][colIndex] = '*'
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            slopeDown = False
            rowIndex -= 2
    else:
        rowIndex -= 1
        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
```

```python
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.

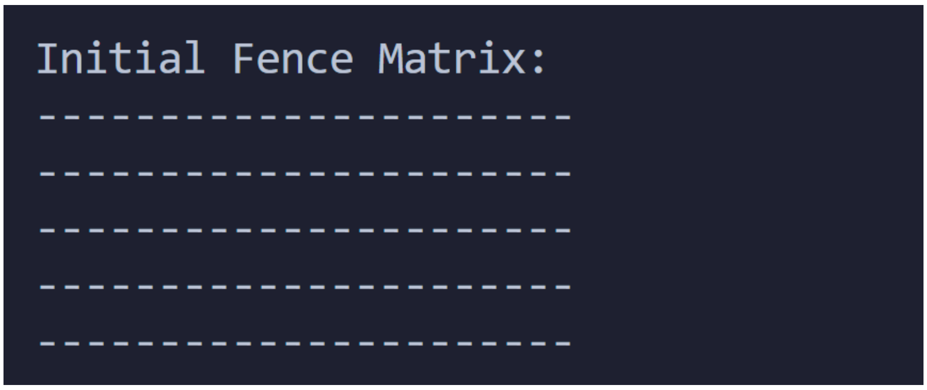
Say we wish to encrypt: ‘CEASEFIRETILLDAWN’
and we take our rail-size as 5

We run our code for encryption:



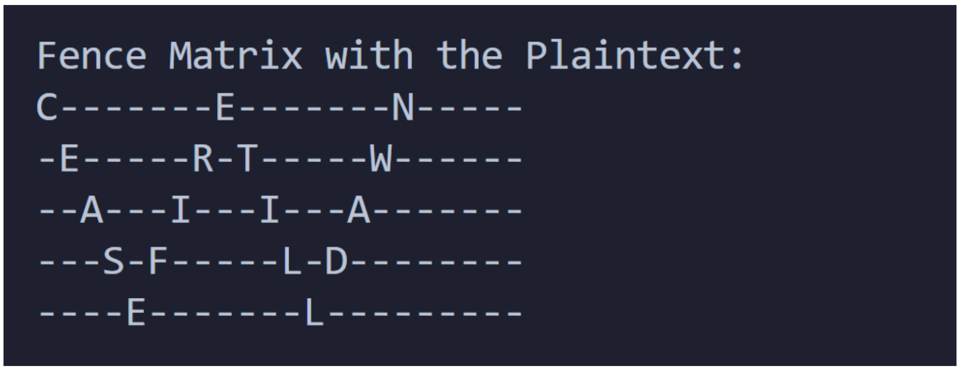
```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\encry
pt.py
Rail Fence Encrypt

Enter Plaintext: CEASEFIRETILLDAWN
Enter Fence Size: 5
```

Then, it shows us our initial fence:



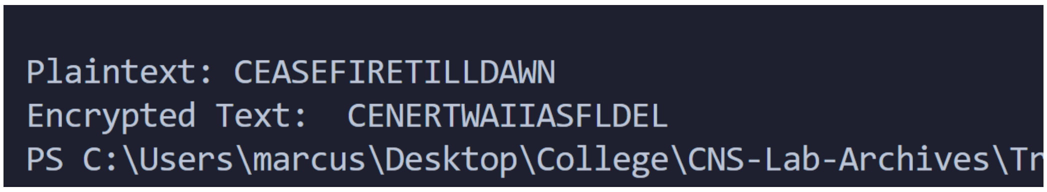
```
Initial Fence Matrix:
----------------------
----------------------
----------------------
----------------------
----------------------
```

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



```
Fence Matrix with the Plaintext:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------
```

Finally, we get the encrypted string as:

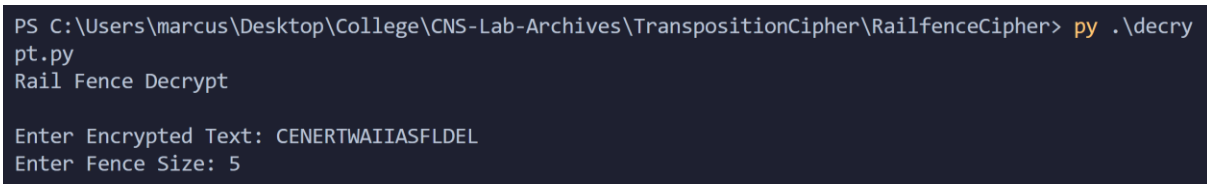


```
Plaintext: CEASEFIRETILLDAWN
Encrypted Text:  CENERTWAIIASFLDEL
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Tr
```

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

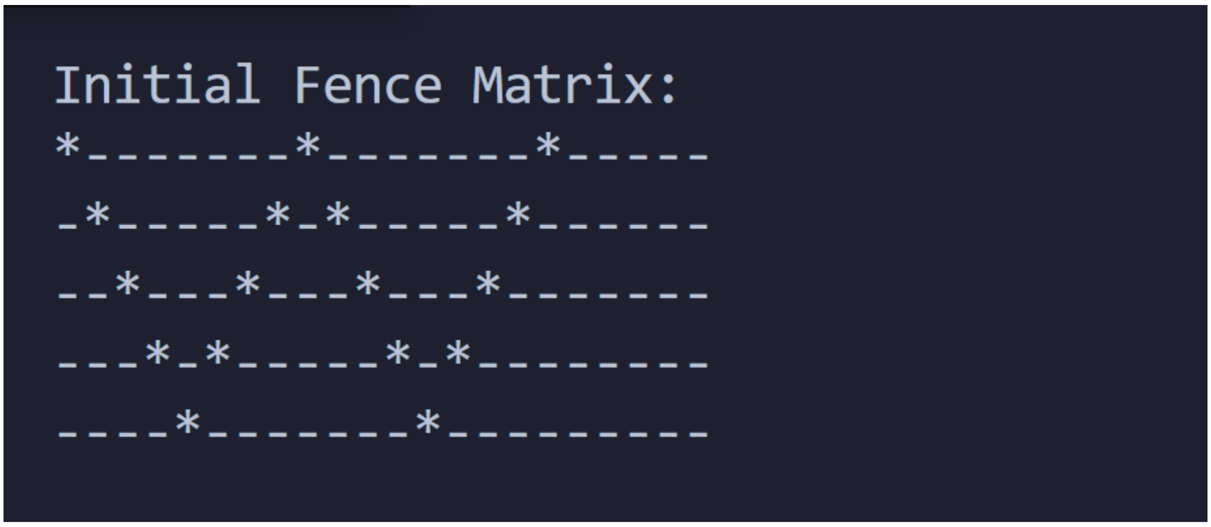
Now, we decrypt using the code.

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



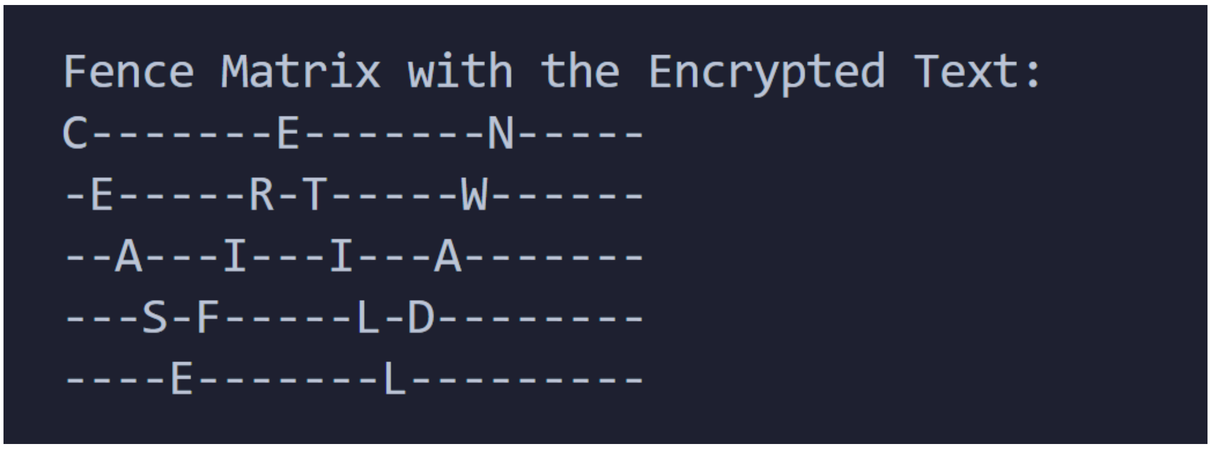
```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\decry
pt.py
Rail Fence Decrypt

Enter Encrypted Text: CENERTWAIIASFLDEL
Enter Fence Size: 5
```

We look at our initial fence matrix:



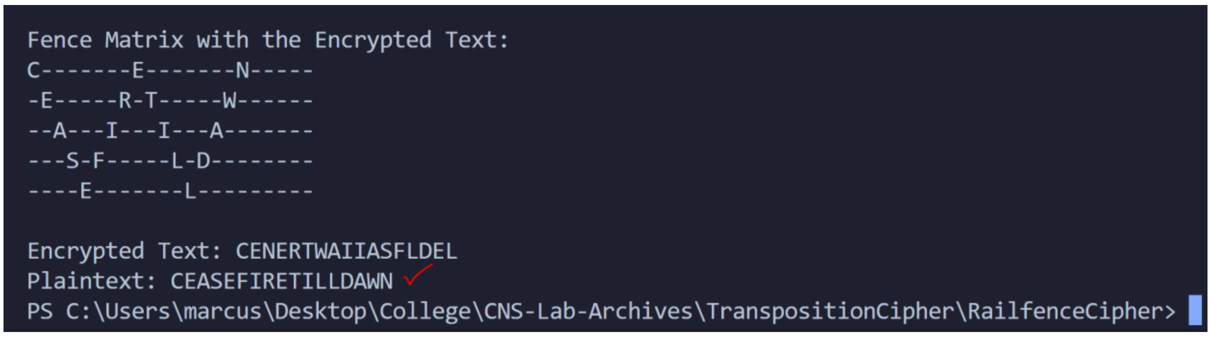
```
Initial Fence Matrix:
*_______*_______*_____
_*_____*_*_____*______
__*___*___*___*_______
___*_*_____*_*________
____*_______*_________
```

Then one that is filled with our encrypted text:



```
Fence Matrix with the Encrypted Text:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------
```

Finally, we get our plaintext:



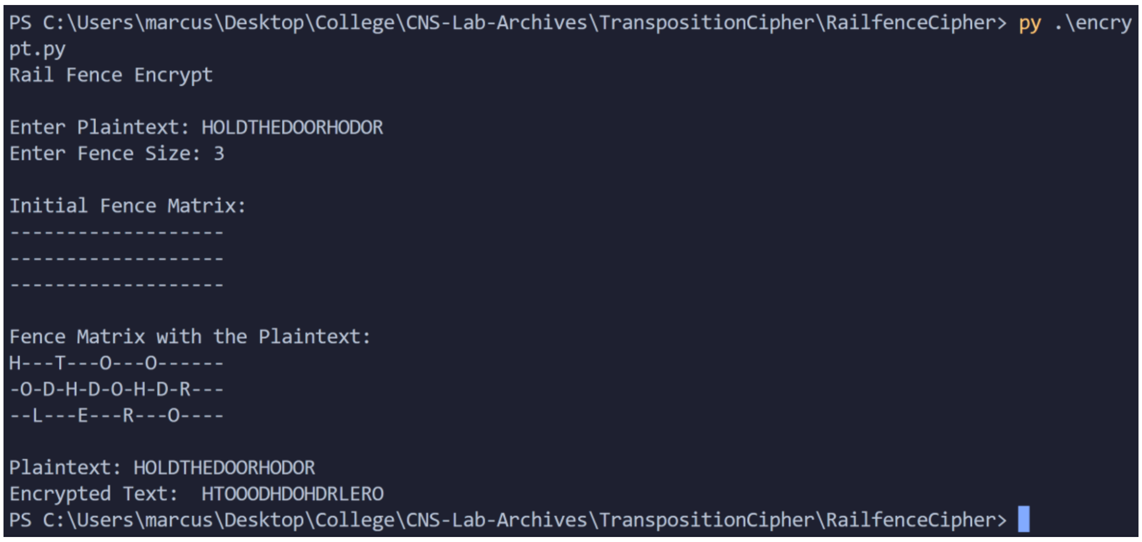
```
Fence Matrix with the Encrypted Text:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------

Encrypted Text: CENERTWAIIASFLDEL
Plaintext: CEASEFIRETILLDAWN ✓
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

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.

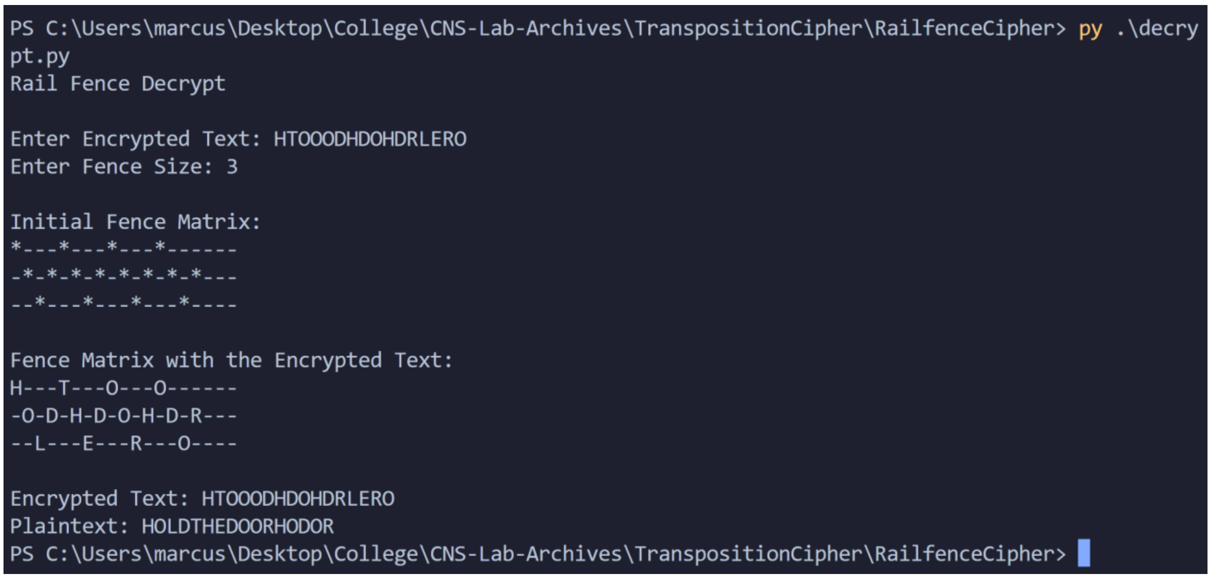
Using our encryption code:



```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\encry
pt.py
Rail Fence Encrypt

Enter Plaintext: HOLDTHEDOORHODOR
Enter Fence Size: 3

Initial Fence Matrix:
----------------
----------------
----------------

Fence Matrix with the Plaintext:
H---T---O---O------
-O-D-H-D-O-H-D-R---
--L---E---R---O----

Plaintext: HOLDTHEDOORHODOR
Encrypted Text:  HTOOODHDOHDRLERO
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

The encrypted text would be: ‘HTOOODHDOHDRLERO’.

Now, we decrypt:



```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\decry
pt.py
Rail Fence Decrypt

Enter Encrypted Text: HTOOODHDOHDRLERO
Enter Fence Size: 3

Initial Fence Matrix:
*---*---*---*------
-*-*-*-*-*-*-*-*---
--*---*---*---*----

Fence Matrix with the Encrypted Text:
H---T---O---O------
-O-D-H-D-O-H-D-R---
--L---E---R---O----

Encrypted Text: HTOOODHDOHDRLERO
Plaintext: HOLDTHEDOORHODOR
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

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.