**Batch: Roll No.: 16010121045**

**Experiment No.: 1**

|  |
| --- |
| **Title: Encryption-Decryption programs using classical cryptography** |

**Objective:**

To write a program to convert plain text into cipher text using Caesar cipher and Transposition cipher.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **1** | **Explain fundamentals of Information Security and cryptography** |

**Books/ Journals/ Websites referred:**

<https://www.britannica.com/topic/transposition-cipher>

**Abstract**:-

In this experiment we need to write programs to encrypt the plaintext into ciphertext and to decrypt the ciphertext back to the plaintext using Caesar cipher and Transposition cipher.

**Related Theory:**

**Substitutional Cipher:**

1. **Caesar Cipher**

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar.

The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

For example,

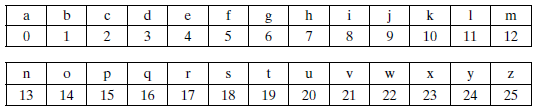


Note that the alphabet is wrapped around, so that the letter following Z is A.

We can define the transformation by listing all possibilities, as follows:



Let us assign a numerical equivalent to each letter:



The algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:

*C* = E(3, *p*) = (*p* + 3) mod 26

A shift may be of any amount, so the general Caesar algorithm is

C = E(k, p) = (p + k) mod 26

where k takes on a value in the range 1 to 25.

The decryption algorithm is simply

p = D(k, C) = (C - k) mod 26

If it is known that a given cipher text is a Caesar cipher, then a brute-force cryptanalysis is easily performed. Simply try all the 25 possible keys.

1. **Play fair cipher:**

Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a **digraph** (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5\*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then **J** is replaced by **I**. It means treat I and J as the same letter, accordingly.

Since Playfair cipher encrypts the message **digraph by digraph**. Therefore, the Playfair cipher is an example of a **digraph substitution cipher**.

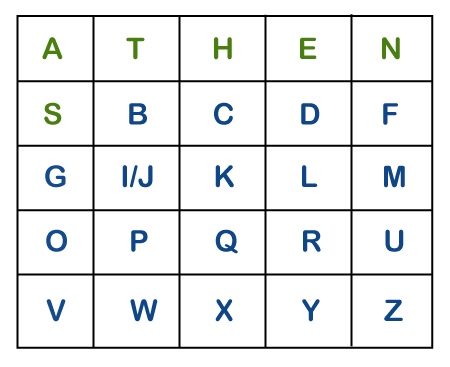
**Encryption**

1. First, split the plaintext into **digraphs** (pair of two letters). If the plaintext has the odd number of letters, append the letter **Z** at the end of the plaintext. It makes the plaintext of **even** For example, the plaintext **MANGO** has five letters. So, it is not possible to make a digraph. Since, we will append a letter **Z** at the end of the plaintext, i.e. **MANGOZ.**

2. After that, break the plaintext into **digraphs** (pair of two letters). If any letter appears twice (side by side), put **X** at the place of the second occurrence. Suppose, the plaintext is **COMMUNICATE** then its digraph becomes **CO MX MU NI CA TE**. Similarly, the digraph for the plaintext **JAZZ** will be **JA ZX ZX**, and for plaintext **GREET**, the digraph will be **GR EX ET.**

3. To determine the cipher (encryption) text, first, build a 5\*5 key-matrix or key-table and filled it with the letters of alphabets, as directed below:

* Fill the first row (left to right) with the letters of the given keyword (**ATHENS**). If the keyword has duplicate letters (if any) avoid them. It means a letter will be considered only once. After that, fill the remaining letters in alphabetical order. Let's create a 5\*5 key-matrix for the keyword **ATHENS**.

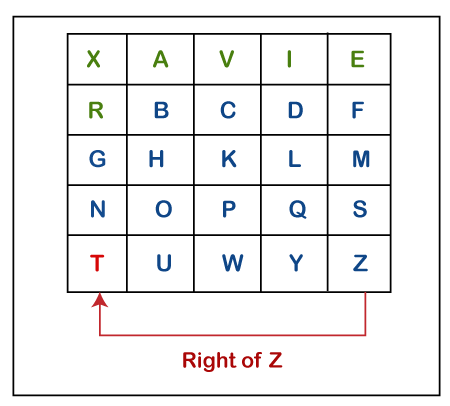


Note that in the above matrix any letter is not repeated. The letters in the first row (in green color) represent the keyword and the remaining letters sets in alphabetical order.

4. There may be the following three conditions:

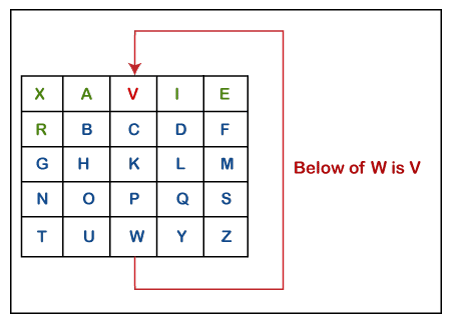
**i) If a pair of letters (digraph) appears in the same row**

In this case, replace each letter of the digraph with the letters immediately to their right. If there is no letter to the right, consider the first letter of the same row as the right letter. Suppose, **Z** is a letter whose right letter is required, in such case, **T** will be right to Z.



**ii) If a pair of letters (digraph) appears in the same column**

In this case, replace each letter of the digraph with the letters immediately below them. If there is no letter below, wrap around to the top of the same column. Suppose, **W** is a letter whose below letter is required, in such case, **V** will be below W.

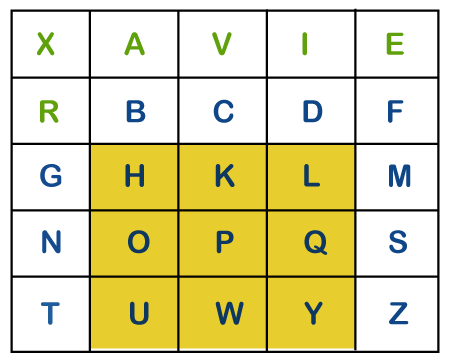


**iii) If a pair of letters (digraph) appears in a different row and different column**

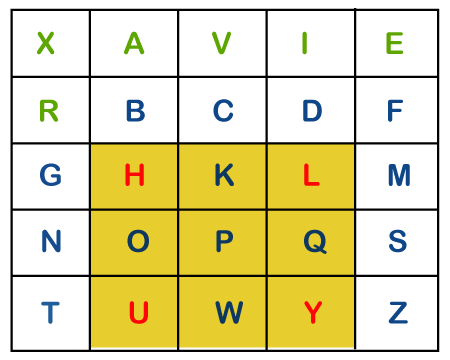
In this case, select a 3\*3 matrix from a 5\*5 matrix such that pair of letters appear in the 3\*3 matrix. Since they occupy two opposite corners of a square within the matrix. The other corner will be a cipher for the given digraph.

In other words, we can also say that intersection of H and Y will be the cipher for the first letter and

Suppose, a digraph is **HY** and we have to find a cipher for it. We observe that both H and Y are placed in different rows and different columns. In such cases, we have to select a 3\*3 matrix in such a way that both H and Y appear in the 3\*3 matrix (highlighted with yellow color). Now, we will consider only the selected matrix to find the cipher.



Now to find the cipher for HY, we will consider the diagonal **opposite** to HY, i.e. **LU.** Therefore, the cipher for **H** will be **L,** and the cipher for **Y** will be **U**.

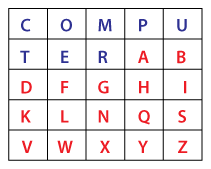


**Decryption**

Suppose, the plaintext is **COMMUNICATION** and the key that we will use to encipher the plaintext is **COMPUTER**. The key can be any word or phrase. Let's encipher the message **COMMUNICATION**.

1. First, split the plaintext into digraph (by rule 2) i.e. **CO MX MU NI CA TE**.

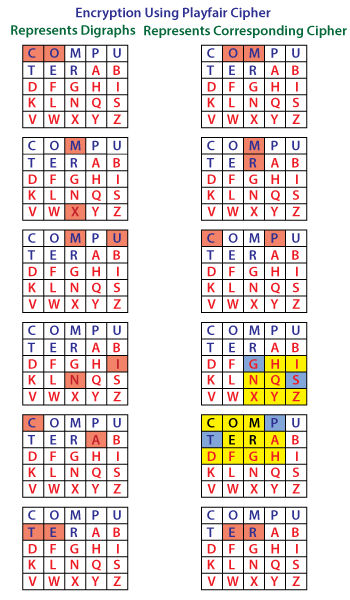
2. Construct a 5\*5 key-matrix (by rule 3). In our case, the key is **COMPUTER**.



3. Now, we will traverse in key-matrix pair by pair and find the corresponding encipher for the pair.

* The first digraph is **CO**. The pair appears in the same row. By using **Rule 4(i) CO** gets encipher into **OM**.
* The second digraph is **MX**. The pair appears in the same column. By using **Rule 4(ii) MX** gets encipher into **RM**.
* The third digraph is **MU**. The pair appears in the same row. By using **Rule 4(i) MU** gets encipher into **PC**.
* The fourth digraph is **NI**. The pair appears in different rows and different columns. By using **Rule 4(iii) NI** gets encipher into **SG**.
* The fifth digraph is **CA**. The pair appears in different rows and different columns. By using **Rule 4(iii) CA** gets encipher into **PT**.
* The sixth digraph is **TE**. The pair appears in the same row. By using **Rule 4(i) TE** gets encipher into **ER**.

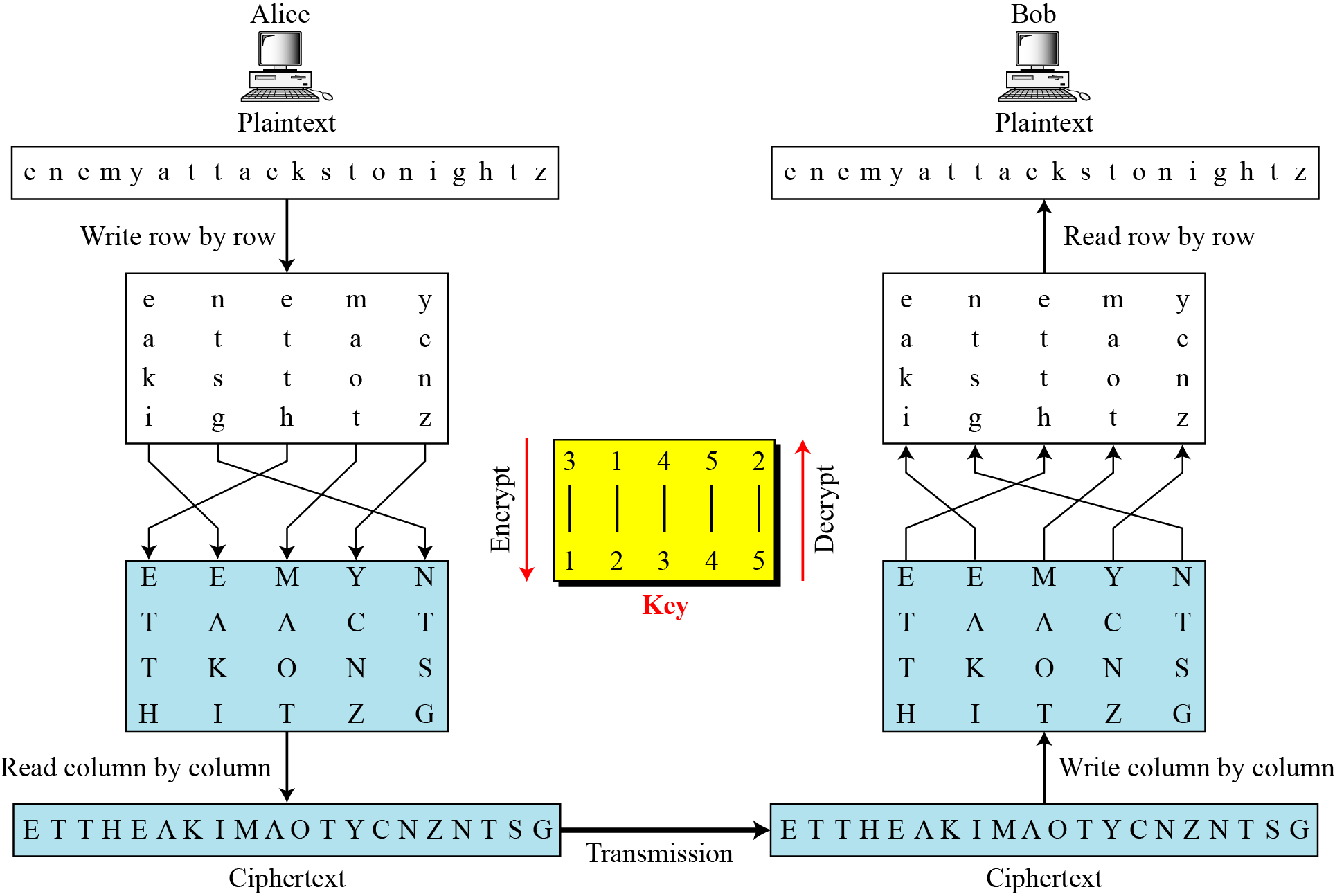
Therefore, the plaintext **COMMUNICATE** gets encipher (encrypted) into **OMRMPCSGPTER.**



**Transposition cipher**

1. **Columnar Cipher:**

The logic behind transposition cipher is as shown in the given figure:



1. **Rail Fence Cipher:**

The logic behind Rail Fence cipher is as shown in the given figure:

Table

Description automatically generated

**Implementation Details:**

**Code:**

Rail fence Cipher

def encrypt(*msg*, *key*):

arr = [['\n' *for* i *in* range(*key*)]*for* j *in* range(len(*msg*))]

dir = True

j = 0

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

arr[i][j] = *msg*[i]

*if* (j == *key*-1):

dir = False

*elif* (j == 0):

dir = True

*if* (dir):

j += 1

*else*:

j -= 1

eMsg = ''

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

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

*if* (arr[i][j] != '\n'):

eMsg += arr[i][j]

*return* eMsg

def decrypt(*eMsg*, *key*):

arr = [['\n' *for* i *in* range(*key*)]*for* j *in* range(len(*eMsg*))]

dir = True

j = 0

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

arr[i][j] = '#'

*if* j == 0:

dir = True

*elif* j == *key*-1:

dir = False

*if* dir:

j += 1

*else*:

j -= 1

msg = ''

pos = 0

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

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

*if* (arr[i][j] == '#'):

arr[i][j] = *eMsg*[pos]

pos += 1

j = 0

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

msg += arr[i][j]

*if* (j == 0):

dir = True

*elif* (j == *key*-1):

dir = False

*if* (dir):

j += 1

*else*:

j -= 1

*return* msg

print('''Select an Option

(1) Encrypt Using Rail Fence Cipher

(2) Decrypt Using Rail Fence Cipher

(3) Exit''')

n = input()

msg = input('Enter Message: ')

key = int(input('Enter Key: '))

*if* (n == '1'):

print("Encrypted Message:", encrypt(msg, key))

*elif* (n == '2'):

*pass*

print("Decrypted Message:", decrypt(msg, key))

*else*:

print("Quitting application")

Columnar Keyed cipher

*import* math

def eSorting(*key*, *seq*):

sortedKey = ''.join(sorted(*key*))

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

*seq*.append(*key*.find(sortedKey[i]))

def dSorting(*key*, *seq*):

sortedKey = ''.join(sorted(*key*))

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

*seq*.append(sortedKey.find(*key*[i]))

def encrypt(*msg*, *key*):

x = int(len(*key*)) *# No of columns*

y = int(math.ceil((len(*msg*)/len(*key*)))) *# No of rows*

arr = [['\n' *for* i *in* range(x)]*for* j *in* range(y)]

padding = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

pos, pPos = 0, 0

*for* i *in* range(y):

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

*try*:

arr[i][j] = *msg*[pos]

pos += 1

*except*:

arr[i][j] = padding[pPos]

pPos += 1

seq = []

eSorting(*key*, seq)

eMsg = ''

*for* i *in* seq:

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

eMsg += arr[j][i]

*return* eMsg

def decrypt(*msg*, *key*):

x = int(len(*key*)) *# No of columns*

y = int(math.ceil((len(*msg*)/len(*key*)))) *# No of rows*

arr = [['\n' *for* i *in* range(x)]*for* j *in* range(y)]

pos = 0

*for* i *in* range(x):

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

arr[j][i] = *msg*[pos]

pos += 1

seq = []

dSorting(*key*, seq)

dMsg = ''

*for* i *in* range(y):

*for* j *in* seq:

dMsg += arr[i][j]

*return* dMsg

print('''Select an Option

(1) Encrypt Using Columnar Fence Cipher

(2) Decrypt Using Columnar Fence Cipher

(3) Exit''')

n = input()

msg = input('Enter Message: ').upper()

key = input('Enter Keyword: ').upper()

*if* (n == '1'):

*try*:

print("Encrypted Message:", encrypt(msg, key))

*except*:

print("The Give Message cannot be encrypted using the given key")

*elif* (n == '2'):

*try*:

print("Decrypted Message:", decrypt(msg, key))

*except*:

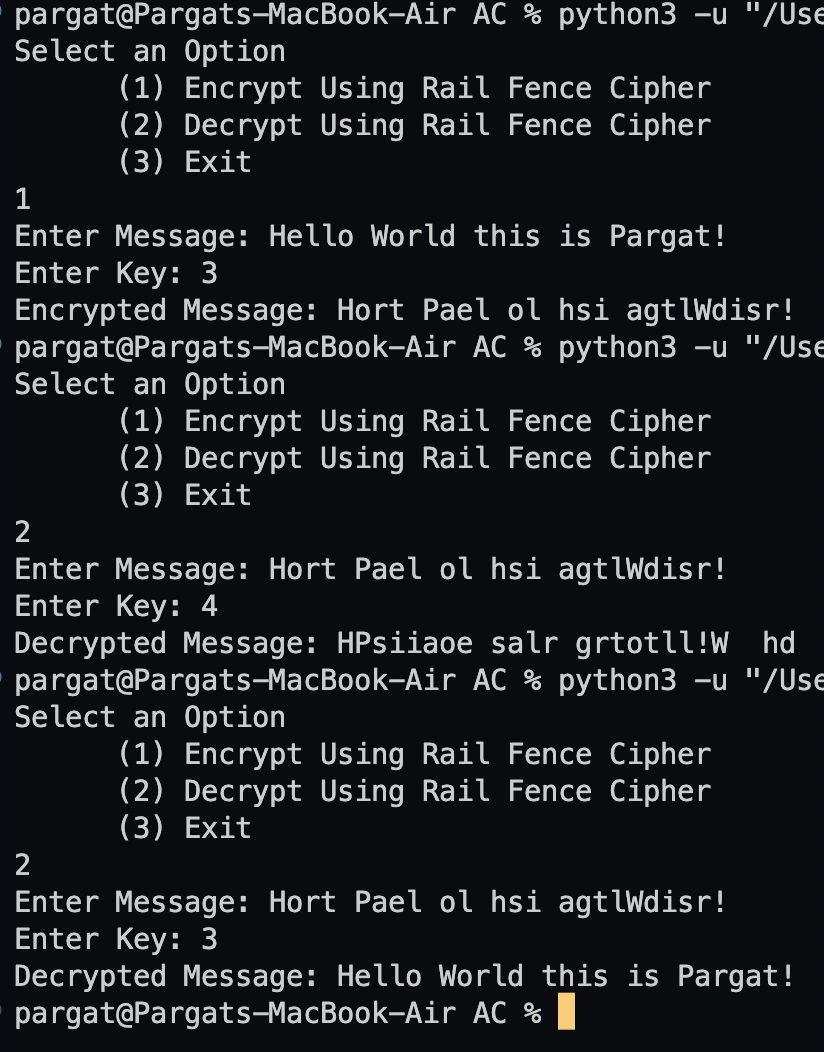
print("The Give Message cannot be decrypted using the given key")

*else*:

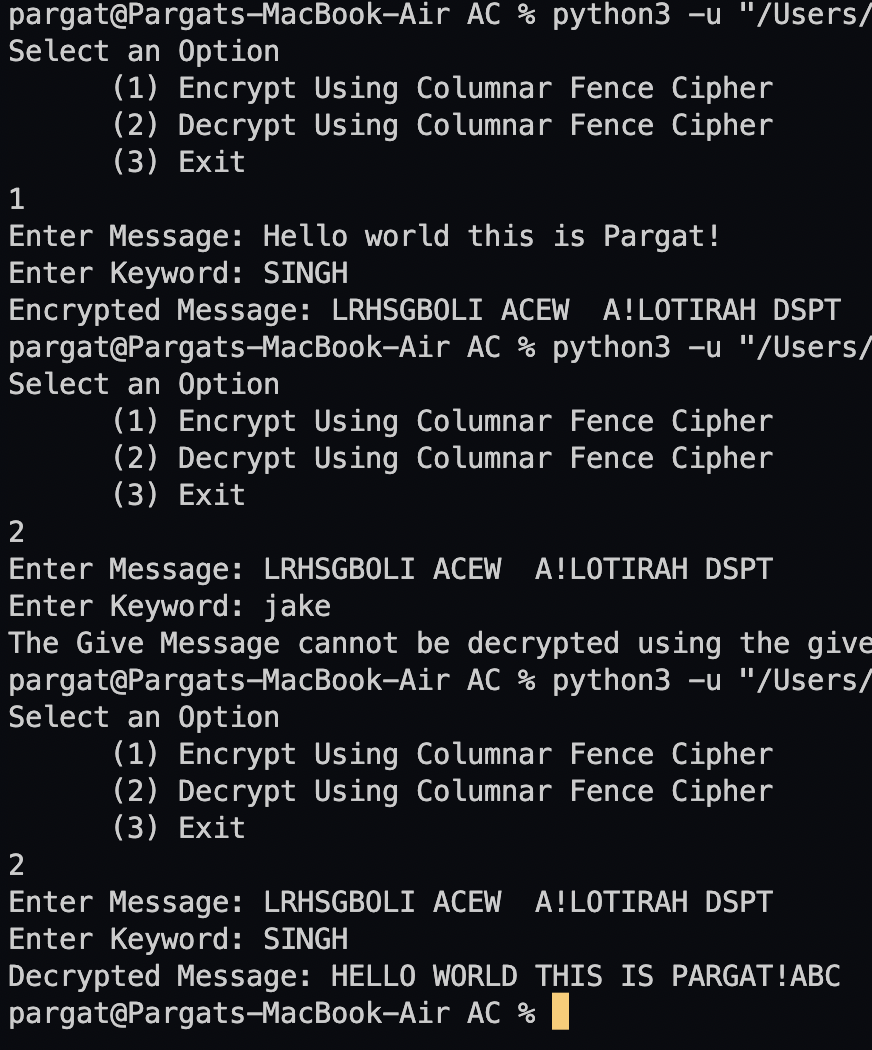
print("Quitting application")

**Sample Output screenshots:**

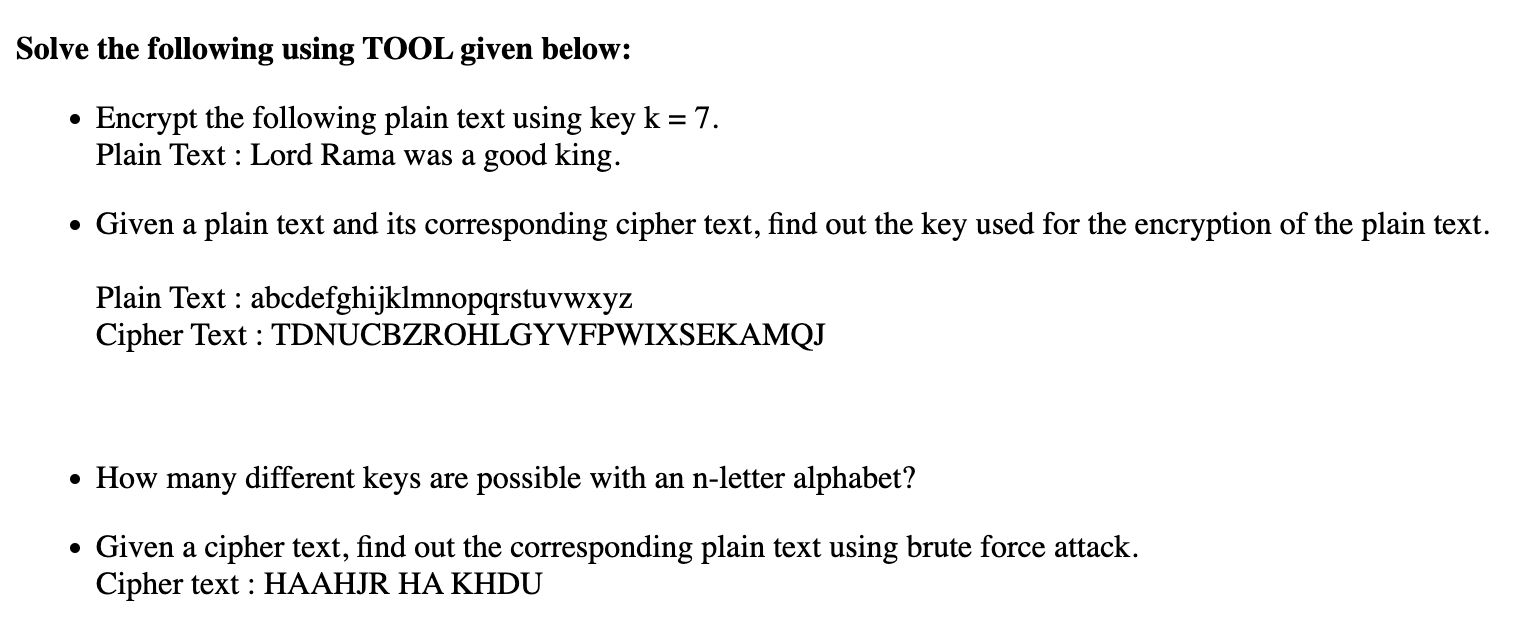
Rail Fence Cipher

****

Columnar Cipher

****

**Vlab:**

****

**Ans 1)**

svyk yhth dhz h nvvk rpun.

**Ans 2)**

The Given Combination for Plain Text and Cypher Text is not a Shift Cipher. Hence there is no shift key for the problem.

**Ans 3)**

25 keys are possible for any shift cipher as there are only 26 alphabets in the English language that we can shift around.

**Ans 4)**

ATTACK AT DAWN.

**Conclusion:-**

**Learnt about the various techniques which can be used to Encrypt and Decypt a given plain or Cypher Text.**