Ishaan Mehta E18CSE069 EB02 LabWeek2

Affine Cypher

In [4]:

```
# Defining Affine encrypt function

# This function takes the keys a,b and the total length allowed 'm' and returns a function to calculate
d encrypted index on a int
affineEncrypt = lambda a, b, m : lambda p : (a*p+b) % m
#takes the encryption function and text characters in num iterable format and returns chars for m = 26
 with lowercase input
encrypt = lambda eFun, numText : [chr(eFun(i)+97) for i in numText]

#Defining Affine decrpyt function

getInv = lambda a, m : [i for i in range(9999) if (a*i)%m == 1][0]
affineDecrpyt = lambda aInv, b, m : lambda c : (aInv*(c-b))%m

decrypt = lambda dFun, numText : [chr(dFun(i)+97) for i in numText]
```

In [5]:

```
import sys
#Taking all the Inputs

#defining 'm'
m = 26

# Getting the keys
a, b = int(input("Enter key a:")), int(input("Enter key b:"))
if a%m == 0 or m%a == 0:
    print(f'Invalid Cypher Key. Needs to be prime relative to m:{m}')
    sys.exit("Invalid Input")

# Asking whether to encrypt or decrypt
state = int(input("Enter 0 to encrypt and 1 to decrypt or 2 for both"))

#Getting the text to encrypt or decrpyt
text = str(input("Enter the text to encrypt or decrpt (in range a-b) (A-B will be converted to lowercas
e)")).lower()
if [i for i in text if ord(i) < 97 or ord(i) > 97+26]:
    print(f'Some character is out of range for Affine encryption with current m={m}. Enter in range a-
b')
    sys.exit('Invalid input')

# Converting to numeric
convNum = lambda t : [ord(i)-97 for i in text]
text = convNum(text)
# Initializing encrypt
eFun = affineEncrypt(a, b, m)
aInv = getInv(a, m)
dFun = affineDecrpyt(aInv, b, m)
```

```
Enter key a:3
Enter key b:6
Enter 0 to encrypt and 1 to decrypt or 2 for both2
Enter the text to encrypt or decrpt (in range a-b) (A-B will be converted to lowercase)ishaan
```

In [6]:

```
print(f'a: {a}, b: {b}, m: {m}')

# Encrypt
if state == 0 or state == 2:
    encText = "".join(encrypt(eFun, text))
    print(f'Encrypted Text: {encText}')
    text = [ord(i)-97 for i in encText]
# Decrypt
if state == 1 or state == 2:
    decText = "".join(decrypt(dFun, text))
    print(f'Decrypted Text: {decText}')
```

```
a: 3, b: 6, m: 26
Encrypted Text: eibggt
Decrypted Text: ishaan
```

PlayFair Cypher

In [10]:

```
import numpy as np
import string
class Playfair:
    def __init__(self, key, verbose = False):
        self._key = str(key).lower()
        self.verbose = verbose
        self._Grid = np.array([['']*5]*5 , dtype="str")
        self._visited = {c : False for c in string.ascii_lowercase if c != 'j'}
        self._initGrid()

    def _initGrid(self):
        i,j= self._fillGrid(0,0, self._key)
        self._fillGrid(i,j, string.ascii_lowercase)
        if self.verbose:
            print(self._Grid)

    def _fillGrid(self, i, j, s):
        for c in s:
            if i > 4:
                break
            if j > 4:
                j = 0
                i+=1
            if c == 'j':
                c = 'i'
            if self._visited[c] == False:
                self._Grid[i,j] = c
                self._visited[c] = True
                j+=1
        return i,j


    def encrypt(self, text):
        text = str(text).lower()
        if len(text)%2 != 0:
            text += 'z'
        text = text.replace('j','i')
        diG = [text[i]+text[i+1] for i in range(0,len(text),2)]
        if self.verbose:
            print(f'Following is the digraph: {diG}\n')
        encDig = map(self._encryptor, diG)
        return "".join(list(encDig))

    def decrpyt(self, text):
        text = str(text).lower()
        if len(text)%2 != 0:
            text += 'z'
        text = text.replace('j','i')
        diG = [text[i]+text[i+1] for i in range(0,len(text),2)]
        if self.verbose:
            print(f'Following is the digraph: {diG}\n')
        encDig = map(self._decryptor, diG)
        return "".join(list(encDig))

    def _encryptor(self, d):
        if self.verbose:
            print(f'Mapping: {d[0]},{d[1]}')
        pos1 = np.where(self._Grid == d[0])
        pos2 = np.where(self._Grid == d[1])
        i1,j1,i2,j2 = pos1[0], pos1[1], pos2[0], pos2[1]

        #same column
        if j1 == j2:
            i1+=1
            i2+=1
        #same row
        elif i1 == i2:
            j1+=1
            j2+=1
        #diagonal
        else:
            t = j1
            j1 = j2
            j2 = t
        i1 = 0 if i1 > 4 else i1
        j1 = 0 if j1 > 4 else j1
        i2 = 0 if i2 > 4 else i2
        j2 = 0 if j2 > 4 else j2
        if self.verbose:
            print(f'Mapped to : {self._Grid[i1,j1][0]} ,{self._Grid[i2,j2][0]}\n')
        return self._Grid[i1,j1][0]+self._Grid[i2,j2][0]

    def _decryptor(self, d):
        if self.verbose:
            print(f'Mapping: {d[0]},{d[1]}')
        pos1 = np.where(self._Grid == d[0])
        pos2 = np.where(self._Grid == d[1])
        i1,j1,i2,j2 = pos1[0], pos1[1], pos2[0], pos2[1]

        #same column
        if j1 == j2:
            i1-=1
            i2-=1
        #same row
        elif i1 == i2:
            j1-=1
            j2-=1
        #diagonal
        else:
            t = j1
            j1 = j2
            j2 = t
        i1 = 4 if i1 < 0 else i1
        j1 = 4 if j1 < 0 else j1
        i2 = 4 if i2 < 0 else i2
        j2 = 4 if j2 < 0 else j2
        if self.verbose:
            print(f'Mapped to : {self._Grid[i1,j1][0]} ,{self._Grid[i2,j2][0]}\n')
        return self._Grid[i1,j1][0]+self._Grid[i2,j2][0]
```

In [15]:

```
key = str(input('Enter key text:'))
text = str(input('Enter text to encrypt:'))
```

```
Enter key text:monarchy
Enter text to encrypt:instruments
```

In [17]:

```
cypher = Playfair(key)
print(f'Original Message: {text}')
res = cypher.encrypt(text)
print(f'Encrypted Message: {res}')
```

```
Original Message: instruments
Encrypted Message: gatlmzclrqtx
```

In [ ]: