21BDS0340

Abhinav Dinesh Srivatsa

Cryptography and Network Security Lab

BCSE309P

Assessment – I

**Question 1**

Implement the Playfair cipher without a standard cryptographic library

**Aim**

To implement the Playfair cipher substitution technique in python

**Algorithm**

1. Read the key as input from the user
2. Create a 5x5 grip of the alphabets
3. Read the plain text as input
4. Split the plain text message into pairs
5. If a pair has the same letter, break into a single letter and add ‘Z’
6. If both letters are in the same column, shift downwards
7. If both letters are in the same row, shift right
8. Else, replace with the letters that complete the rectangle the letters form

**Code**

def get\_next\_letter(used\_letter\_list, key):

for letter in key:

if letter not in used\_letter\_list:

used\_letter\_list.append(letter)

return letter

for letter in "ABCDEFGHIKLMNOPQRSTUVWXYZ":

if letter not in used\_letter\_list:

used\_letter\_list.append(letter)

return letter

def create\_matrix(key):

unique\_letters = []

matrix = []

for x in range(5):

letters = []

for y in range(5):

letters.append(get\_next\_letter(unique\_letters, key))

matrix.append(letters)

return matrix

def split\_pairs(text):

if len(text) % 2 == 1:

text += "Z"

text\_list = []

for x in range(0, len(text), 2):

if text[x] == text[x+1]:

text\_list.append("X" + text[x])

else:

text\_list.append(text[x:x+2])

return text\_list

def get\_letter\_position(letter, matrix):

for i in range(len(matrix)):

for j in range(len(matrix[i])):

if letter == matrix[i][j]:

return i, j

def encrypt(plaintext\_list, matrix):

encrypted = ""

for pair in plaintext\_list:

let1 = pair[0]

let2 = pair[1]

x1, y1 = get\_letter\_position(let1, matrix)

x2, y2 = get\_letter\_position(let2, matrix)

if y1 == y2:

encrypted += matrix[(x1 + 1) % 5][y1]

encrypted += matrix[(x2 + 1) % 5][y2]

elif x1 == x2:

encrypted += matrix[x1][(y1 + 1) % 5]

encrypted += matrix[x2][(y2 + 1) % 5]

else:

encrypted += matrix[x1][y2]

encrypted += matrix[x2][y1]

return encrypted

def decrypt(encrypted\_list, matrix):

decrypted = ""

for pair in encrypted\_list:

let1 = pair[0]

let2 = pair[1]

x1, y1 = get\_letter\_position(let1, matrix)

x2, y2 = get\_letter\_position(let2, matrix)

if y1 == y2:

decrypted += matrix[(x1 - 1) % 5][y1]

decrypted += matrix[(x2 - 1) % 5][y2]

elif x1 == x2:

decrypted += matrix[x1][(y1 - 1) % 5]

decrypted += matrix[x2][(y2 - 1) % 5]

else:

decrypted += matrix[x1][y2]

decrypted += matrix[x2][y1]

return decrypted

key = input('Key: ')

plaintext = input('Plaintext: ')

# uppercasing the letters

key = key.upper()

# replacing all J with I

key = key.replace('J', 'I')

# generating the encryption matrix

matrix = create\_matrix(key)

# converting plaintext to uppercase

plaintext = plaintext.upper()

# getting plaintext split into pairs

plaintext\_list = split\_pairs(plaintext)

# encrypting the plaintext

encrypted = encrypt(plaintext\_list, matrix)

print(f'The encrypted text is: {encrypted}')

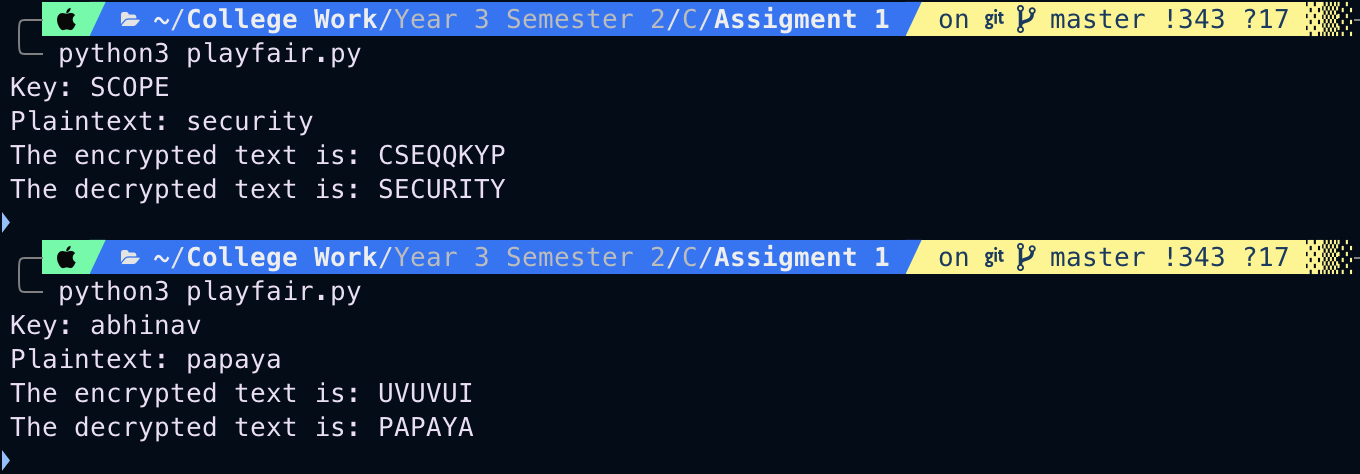
# getting encrypted split into pairs

encrypted\_list = split\_pairs(encrypted)

decrypted = decrypt(encrypted\_list, matrix)

print(f'The decrypted text is: {decrypted}')

**Input/Output**



**Question 2**

Implement the Hill cipher without a standard cryptographic library

**Aim**

To implement the Hill cipher substitution technique in python

**Algorithm**

1. Obtain a plain text message to encode in standard English with no spaces
2. Split the plain text into groups of 3, add as suffix X to fill
3. Convert each group of letters into vectors
4. Replace each letter by its corresponding position in alphabet
5. Create a key 3x3 matrix
6. Multiply the matrix and the vector to obtain a cipher vector
7. Convert each vector into characters, this is the cipher text

**Code**

import math

def determinant(matrix):

size = len(matrix)

if size == 1:

return matrix[0][0]

if size == 2:

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

det = 0

for i in range(size):

new\_matrix = []

for x in range(1, size):

new\_row = []

for y in range(size):

if y == i:

continue

new\_row.append(matrix[x][y])

new\_matrix.append(new\_row)

det += matrix[0][i] \* (-1) \*\* (i % 2) \* determinant(new\_matrix)

return det

def matrix\_mod\_inverse(matrix, det, mod):

inverse = []

for x in range(len(matrix)):

r = []

for y in range(len(matrix[x])):

new\_matrix = []

for i in range(len(matrix)):

if i == x:

continue

l = []

for j in range(len(matrix)):

if j == y:

continue

l.append(matrix[i][j])

new\_matrix.append(l)

r.append((-1) \*\* (x \* len(matrix) + y)

\* determinant(new\_matrix) % mod)

inverse.append(r)

transpose = []

for x in range(len(inverse)):

l = []

for y in range(len(inverse[x])):

l.append((inverse[y][x] \* det) % 26)

transpose.append(l)

return transpose

def mod\_inverse(num, mod):

num = num % mod

i = 0

while (i \* num) % mod != 1:

i += 1

continue

return i

def letters\_to\_numbers(text):

text = text.upper()

return [(ord(let) - 65) % 26 for let in text]

def numbers\_to\_letters(nums):

return [chr(n + 65) for n in nums]

def list\_to\_square\_matrix(l):

size = int(math.sqrt(len(l)))

matrix = []

for x in range(size):

vals = []

for y in range(size):

vals.append(l[x \* size + y])

matrix.append(vals)

return matrix

def multiply(matrix, arr):

res = []

for i in matrix:

s = 0

for index in range(len(i)):

s = (s + i[index] \* arr[index]) % 26

res.append(s)

return res

key = input("Key: ")

plaintext = input("Plaintext: ")

# getting numeric list of key

key\_letter\_list = letters\_to\_numbers(key)

# getting numeric list of plaintext

text\_letter\_list = letters\_to\_numbers(plaintext)

# generating matrix from key list values

matrix = list\_to\_square\_matrix(key\_letter\_list)

# performing matrix multiplication to encrypt

encypted\_nums = multiply(matrix, text\_letter\_list)

encrypted = numbers\_to\_letters(encypted\_nums)

print(f'The encrypted text is: {"".join(encrypted)}')

# finding determinant and its mod inverse for decryption

det = determinant(matrix)

positive\_determinant = det if det > 0 else -det

mod\_inverse\_determinant = mod\_inverse(det, 26)

# getting the matrix inverse for decryption

inverse = matrix\_mod\_inverse(matrix, mod\_inverse\_determinant, 26)

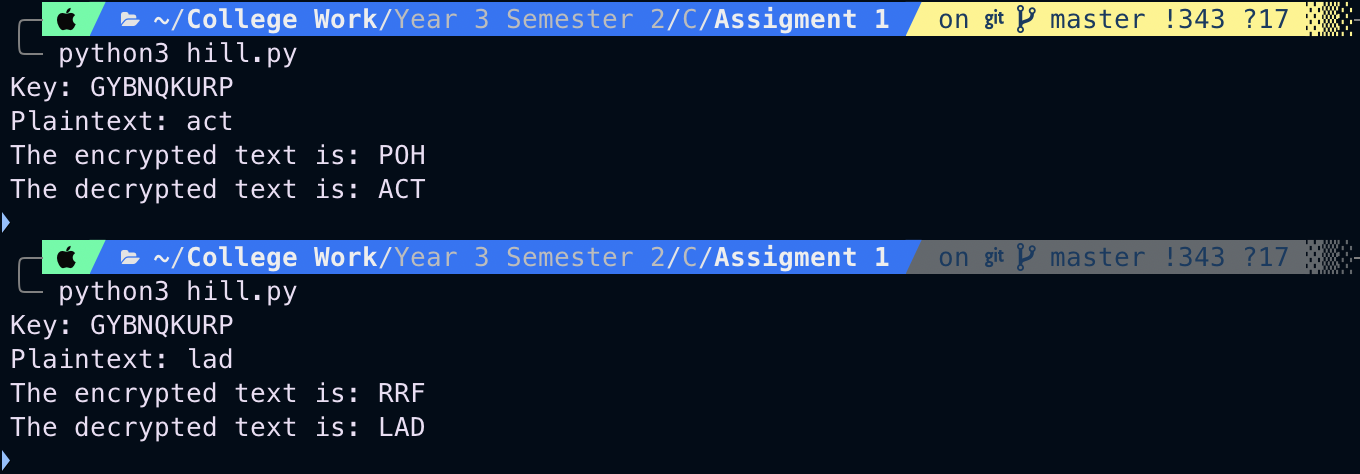
# performing the decryption

decrypted\_nums = multiply(inverse, encypted\_nums)

decrypted = numbers\_to\_letters(decrypted\_nums)

print(f'The decrypted text is: {"".join(decrypted)}')

**Input/Output**

****

**Question 3**

Implement the Vigenère cipher without a standard cryptographic library

**Aim**

To implement the Vigenère cipher substitution technique in python

**Algorithm**

1. Take input for the key and the plain text
2. Convert the key and plain text to their corresponding numeric values
3. The cipher numbers can be obtained by performing a modulo addition on the key and the plain text
4. Convert the cipher numbers to letters to obtain the cipher text

**Code**

def letters\_to\_numbers(text):

text = text.upper()

return [(ord(let) - 65) % 26 for let in text]

def numbers\_to\_letters(nums):

return [chr(n + 65) for n in nums]

def encrypt(key, plaintext):

key\_size = len(key)

encrypted = []

for i in range(len(plaintext)):

encrypted.append((plaintext[i] + key[i % key\_size]) % 26)

return encrypted

def decrypt(key, ciphertext):

key\_size = len(key)

decrypted = []

for i in range(len(ciphertext)):

decrypted.append((ciphertext[i] - key[i % key\_size]) % 26)

return decrypted

key = input("Key: ")

plaintext = input("Plaintext: ")

key\_letter\_list = letters\_to\_numbers(key)

plaintext\_letter\_list = letters\_to\_numbers(plaintext)

encrypted\_nums = encrypt(key\_letter\_list, plaintext\_letter\_list)

encrypted = numbers\_to\_letters(encrypted\_nums)

print(f'The encrypted text is: {"".join(encrypted)}')

decrypted\_nums = decrypt(key\_letter\_list, encrypted\_nums)

decrypted = numbers\_to\_letters(decrypted\_nums)

print(f'The decrypted text is: {"".join(decrypted)}')

**Input/Output**

