**Assignment No 3**

**Cryptography and Network Security Lab (5CS453)**

**Name: Sumit Narake PRN: 2020BTECS00023**

**Class: Final Year - CSE**

**Title:**

**Encryption and Decryption using Playfair Cipher Technique.**

**Aim:**

**To Study and Implement Encryption and Decryption using Playfair Cipher Technique.**

**Theory:**

**Playfair Cipher Technique**

🡪

The Playfair Cipher Technique is a substitution cipher that encrypts pairs of characters (digraphs) from the plaintext using a 5x5 key square matrix.

The matrix is constructed from a keyword, with duplicate letters removed and the keyword letters placed at the beginning.

Encryption involves applying rules based on the positions of the letters within the key square.

If the letters are in the same row, column, or form a rectangle, they are replaced by specific neighboring letters.

**Encryption**:

* Divide the plaintext into digraphs.
* Apply rules based on the positions of letters in the key square to replace each digraph.

**Decryption**:

* Divide the ciphertext into digraphs.
* Apply the rules in reverse to each digraph to retrieve the original plaintext.

**Advantages**:

* Enhanced security due to digraphs and key square usage.
* Reduces susceptibility to frequency analysis.
* Key square generation is straightforward using a keyword.

**Disadvantages**:

* Complexity increases with handling various cases (same row, column, rectangle).
* Security depends on the keyword and arrangement of the key square.

**Code:**

**Encryption:**

def toLowerCase(text):

    return text.lower()

def removeSpaces(text):

    newText = ""

    for i in text:

        if i == " ":

            continue

        else:

            newText = newText + i

    return newText

# Function to group 2 elements of a string as a list element

def Diagraph(text):

    Diagraph = []

    group = 0

    for i in range(2, len(text), 2):

        Diagraph.append(text[group:i])

        group = i

    Diagraph.append(text[group:])

    return Diagraph

# Function to fill a letter in a string element (If 2 letters in the same string matches)

def FillerLetter(text):

    k = len(text)

    if k % 2 == 0:

        for i in range(0, k, 2):

            if text[i] == text[i+1]:

                new\_word = text[0:i+1] + str('x') + text[i+1:]

                new\_word = FillerLetter(new\_word)

                break

            else:

                new\_word = text

    else:

        for i in range(0, k-1, 2):

            if text[i] == text[i+1]:

                new\_word = text[0:i+1] + str('x') + text[i+1:]

                new\_word = FillerLetter(new\_word)

                break

            else:

                new\_word = text

    return new\_word

alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',

        'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

# Function to generate the 5x5 key square matrix

def generateKeyTable(word, alphabets):

    key\_letters = []

    for i in word:

        if i not in key\_letters:

            key\_letters.append(i)

    compElements = []

    for i in key\_letters:

        if i not in compElements:

            compElements.append(i)

    for i in alphabets:

        if i not in compElements:

            compElements.append(i)

    matrix = []

    while compElements != []:

        matrix.append(compElements[:5])

        compElements = compElements[5:]

    return matrix

def search(mat, element):

    for i in range(5):

        for j in range(5):

            if(mat[i][j] == element):

                return i, j

def encrypt\_RowRule(matr, e1r, e1c, e2r, e2c):

    char1 = ''

    if e1c == 4:

        char1 = matr[e1r][0]

    else:

        char1 = matr[e1r][e1c+1]

    char2 = ''

    if e2c == 4:

        char2 = matr[e2r][0]

    else:

        char2 = matr[e2r][e2c+1]

    return char1, char2

def encrypt\_ColumnRule(matr, e1r, e1c, e2r, e2c):

    char1 = ''

    if e1r == 4:

        char1 = matr[0][e1c]

    else:

        char1 = matr[e1r+1][e1c]

    char2 = ''

    if e2r == 4:

        char2 = matr[0][e2c]

    else:

        char2 = matr[e2r+1][e2c]

    return char1, char2

def encrypt\_RectangleRule(matr, e1r, e1c, e2r, e2c):

    char1 = ''

    char1 = matr[e1r][e2c]

    char2 = ''

    char2 = matr[e2r][e1c]

    return char1, char2

def encrypt(Matrix, plainList):

    CipherText = []

    for i in range(0, len(plainList)):

        c1 = 0

        c2 = 0

        ele1\_x, ele1\_y = search(Matrix, plainList[i][0])

        ele2\_x, ele2\_y = search(Matrix, plainList[i][1])

        if ele1\_x == ele2\_x:

            c1, c2 = encrypt\_RowRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

            # Get 2 letter cipherText

        elif ele1\_y == ele2\_y:

            c1, c2 = encrypt\_ColumnRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

        else:

            c1, c2 = encrypt\_RectangleRule(

                Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

        cipher = c1 + c2

        CipherText.append(cipher)

    return CipherText

# Main code

msg = str(input("Enter the message to be encrypted:\n"))

msg = removeSpaces(toLowerCase(msg))

PlainTextList = Diagraph(FillerLetter(msg))

if len(PlainTextList[-1]) != 2:

    PlainTextList[-1] = PlainTextList[-1]+'z'

key = str(input("Enter the key for encryption: "))

key = toLowerCase(key)

Matrix = generateKeyTable(key, alphabets)

CipherList = encrypt(Matrix, PlainTextList)

CipherText = ""

for i in CipherList:

    CipherText += i

print("Cipher text is:", CipherText)

**Decryption:**

def toLowerCase(txt):

    return txt.lower()

def removeSpaces(txt):

    # Remove all spaces in a string can be extended to remove punctuation

    return ''.join(txt.split())

# Generates the 5x5 key square

def generateKeyTable(key):

    keyT = [['' for i in range(5)] for j in range(5)]

    dicty = {chr(i + 97): 0 for i in range(26)}

    for i in range(len(key)):

        if key[i] != 'j':

            dicty[key[i]] = 2

    dicty['j'] = 1

    i, j, k = 0, 0, 0

    while k < len(key):

        if dicty[key[k]] == 2:

            dicty[key[k]] -= 1

            keyT[i][j] = key[k]

            j += 1

            if j == 5:

                i += 1

                j = 0

        k += 1

    for k in dicty.keys():

        if dicty[k] == 0:

            keyT[i][j] = k

            j += 1

            if j == 5:

                i += 1

                j = 0

    return keyT

# Search for the characters of a digraph in the key square and return their position

def search(keyT, a, b):

    arr = [0, 0, 0, 0]

    if a == 'j':

        a = 'i'

    elif b == 'j':

        b = 'i'

    for i in range(5):

        for j in range(5):

            if keyT[i][j] == a:

                arr[0], arr[1] = i, j

            elif keyT[i][j] == b:

                arr[2], arr[3] = i, j

    return arr

# Function to find the modulus with 5

def mod5(a):

    if a < 0:

        a += 5

    return a % 5

# Decryption

def decrypt(str, keyT):

    ps = len(str)

    i = 0

    while i < ps:

        a = search(keyT, str[i], str[i+1])

        if a[0] == a[2]:

            str = str[:i] + keyT[a[0]

                                ][mod5(a[1]-1)] + keyT[a[0]][mod5(a[3]-1)] + str[i+2:]

        elif a[1] == a[3]:

            str = str[:i] + keyT[mod5(a[0]-1)][a[1]] + \

                keyT[mod5(a[2]-1)][a[1]] + str[i+2:]

        else:

            str = str[:i] + keyT[a[0]][a[3]] + keyT[a[2]][a[1]] + str[i+2:]

        i += 2

    return str

# Main code

c\_txt = str(input("Enter the text to be decrypted:\n"))

c\_txt = removeSpaces(toLowerCase(c\_txt))

key = str(input("Enter the key for decryption: "))

key = removeSpaces(toLowerCase(key))

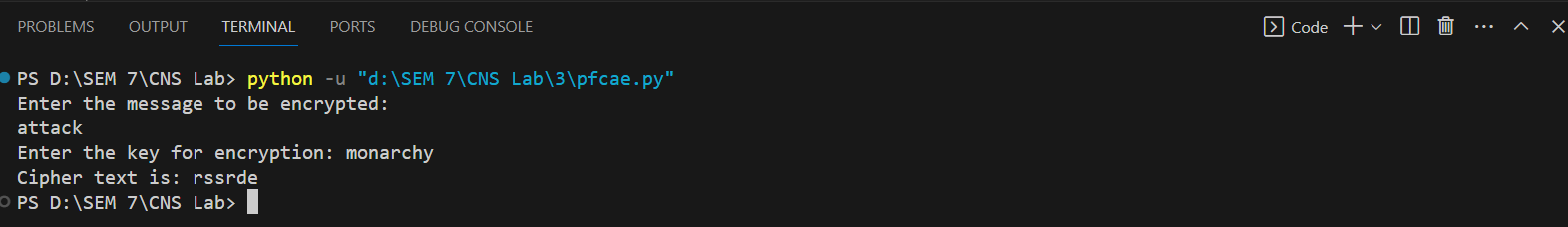
keyT = generateKeyTable(key)

# Decrypted text

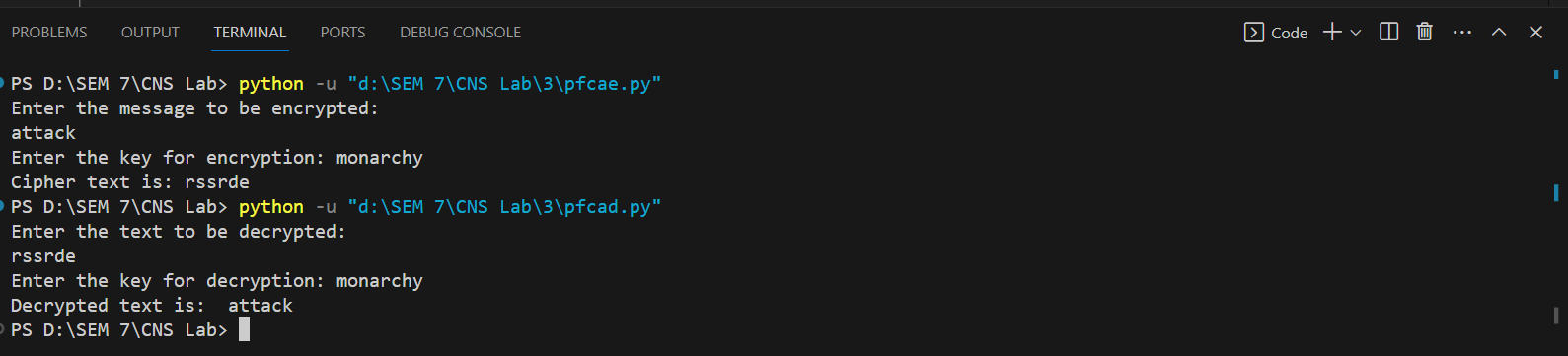
print("Decrypted text is: ", decrypt(c\_txt, keyT))

**Output:**

**Encryption:**

****

**Decryption:**

****