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

```
~/College Work/Year 3 Semester 2/C/Assigment 1 on git master !343 ?17
└─ python3 playfair.py
Key: SCOPE
Plaintext: security
The encrypted text is: CSEQQKYP
The decrypted text is: SECURITY

~/College Work/Year 3 Semester 2/C/Assigment 1 on git master !343 ?17
└─ python3 playfair.py
Key: abhinav
Plaintext: papaya
The encrypted text is: UVUVUI
The decrypted text is: PAPAYA
```

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

```
~/College Work/Year 3 Semester 2/C/Assigment 1 on git master !343 ?17
└─ python3 hill.py
Key: GYBNQKURP
Plaintext: act
The encrypted text is: POH
The decrypted text is: ACT

~/College Work/Year 3 Semester 2/C/Assigment 1 on git master !343 ?17
└─ python3 hill.py
Key: GYBNQKURP
Plaintext: lad
The encrypted text is: RRF
The decrypted text is: LAD
```

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