Assignment No: 1

Title: Implement Encryption Algorithms

Name: Shubhankar jakate
PRN : 22310371
Roll No = 382019

Objective: The key objective of this assignment as follows
1. Study of Information security and its models
2. Study of Encryption algorithms
2. Study of different encryption algorithms for text, image and file.

Theory:

Caesar Cipher
- Type: Substitution Cipher
- Working: Each letter in the plaintext is shifted a fixed number of positions down the alphabet.
- Example: With a shift of 3, A → D, B → E, etc.
- Weakness: Easy to break using brute force as there are only 25 possible shifts.

Playfair Cipher
- Type: Digraph Substitution Cipher
- Working: Encrypts pairs of letters using a 5x5 matrix of letters generated from a keyword.
- Rules:
    - Same row: Replace with letters to the right.
    - Same column: Replace with letters below.
    - Rectangle: Replace with opposite corners.
- Advantage: Harder to crack than Caesar since it works on letter pairs.

Hill Cipher
- Type: Polygraphic Substitution Cipher
- Working: Uses matrix multiplication to transform a block of plaintext into ciphertext using a key matrix.
- Mathematics: Based on linear algebra over modulo 26 arithmetic.
- Strength: More secure for block encryption but vulnerable to known plaintext attacks.

Vigenère Cipher
- Type: Polyalphabetic Substitution Cipher
- Working: Encrypts text using a repeating keyword, shifting each letter by positions based on the corresponding keyword letter.
- Example: Plaintext: ATTACK, Key: LEMON → A+L, T+E, T+M...
- Advantage: More secure than Caesar due to varying shifts.

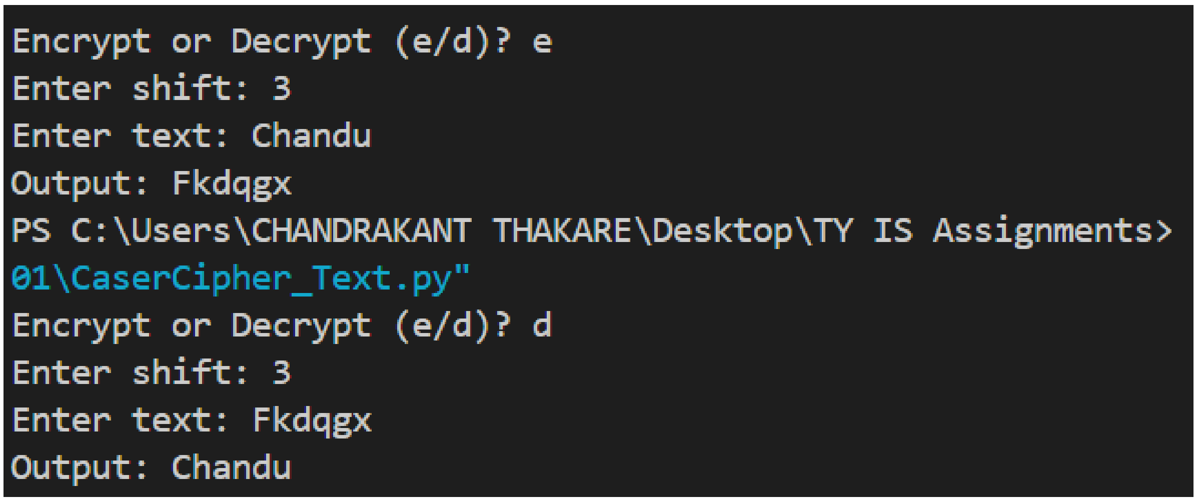
Code:

import numpy as np

# Caesar Cipher

```
def caesar_encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            result += chr((ord(char) - offset + shift) % 26 + offset)
        else:
            result += char
    return result

def caesar_decrypt(text, shift):
    return caesar_encrypt(text, -shift)

if __name__ == "__main__":
    mode = input("Encrypt or Decrypt (e/d)? ").lower()
    shift = int(input("Enter shift: "))
    text = input("Enter text: ")

    output = caesar_encrypt(text, shift) if mode == "e" else
caesar_decrypt(text, shift)
    print("Output:", output)
```

# OUPUT :



```
Encrypt or Decrypt (e/d)? e
Enter shift: 3
Enter text: Chandu
Output: Fkdqgx
PS C:\Users\CHANDRAKANT THAKARE\Desktop\TY IS Assignments>
01\CaserCipher_Text.py"
Encrypt or Decrypt (e/d)? d
Enter shift: 3
Enter text: Fkdqgx
Output: Chandu
```

# Playfair Cipher

```python
def generate_playfair_matrix(key):
    key = key.replace(" ", "").upper().replace('J', 'I')
    matrix = []
    seen = set()
    for char in key:
        if char.isalpha() and char not in seen:
            seen.add(char)
            matrix.append(char)
    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":
        if char not in seen:
            matrix.append(char)
    return [matrix[i:i+5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
                return i, j

def playfair_prepare_text(text):
    text = text.upper().replace("J", "I").replace(" ", "")
    prepared = ''
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'X'
        if a == b:
            prepared += a + 'X'
            i += 1
        else:
            prepared += a + b
            i += 2
    if len(prepared) % 2 != 0:
        prepared += 'X'
    return prepared

def playfair_encrypt(text, key):
    matrix = generate_playfair_matrix(key)
    text = playfair_prepare_text(text)
    cipher = ''
    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        r1, c1 = find_position(matrix, a)
        r2, c2 = find_position(matrix, b)
        if r1 == r2:
            cipher += matrix[r1][(c1 + 1) % 5]
            cipher += matrix[r2][(c2 + 1) % 5]
        elif c1 == c2:
            cipher += matrix[(r1 + 1) % 5][c1]
            cipher += matrix[(r2 + 1) % 5][c2]
        else:
            cipher += matrix[r1][c2]
            cipher += matrix[r2][c1]
    return cipher

def playfair_decrypt(cipher, key):
    matrix = generate_playfair_matrix(key)
    plain = ''
    for i in range(0, len(cipher), 2):
        a, b = cipher[i], cipher[i+1]
        r1, c1 = find_position(matrix, a)
        r2, c2 = find_position(matrix, b)
        if r1 == r2:
            plain += matrix[r1][(c1 - 1) % 5]
            plain += matrix[r2][(c2 - 1) % 5]
        elif c1 == c2:
            plain += matrix[(r1 - 1) % 5][c1]
            plain += matrix[(r2 - 1) % 5][c2]
        else:
            plain += matrix[r1][c2]
            plain += matrix[r2][c1]
    return plain

def clean_playfair_decryption(text):
    cleaned = ''
    i = 0
    while i < len(text):
        if i < len(text)-2 and text[i] == text[i+2] and text[i+1] == 'X':
            cleaned += text[i]
            i += 2
        else:
            cleaned += text[i]
            i += 1
    if cleaned.endswith('X'):
        cleaned = cleaned[:-1]
    return cleaned

while True:
    print("\n===== Playfair Cipher Menu =====")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Exit")
    choice = input("Enter your choice: ")

    if choice == '1':
        key = input("Enter key: ")
        text = input("Enter plaintext: ")
        encrypted = playfair_encrypt(text, key)
        print("Encrypted Text:", encrypted)

    elif choice == '2':
        key = input("Enter key: ")
        text = input("Enter ciphertext: ")
        decrypted = playfair_decrypt(text, key)
        cleaned = clean_playfair_decryption(decrypted)
        print("Decrypted Text:", cleaned)

    elif choice == '3':
        print("Exiting Playfair Cipher.")
        break

    else:
        print("Invalid choice. Try again.")
```

# OUTPUT :

```
===== Playfair Cipher Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter key: Tutorial
Enter plaintext: Hello World
Encrypted Text: KFBWBUXUUCBZ

===== Playfair Cipher Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter key: Tutorial
Enter ciphertext: KFBWBUXUUCBZ
Decrypted Text: HELLOWORLD
```

# Hill Cipher (2x2)

```python
import numpy as np

def mod_inverse(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

def hill_encrypt(text, key_matrix):
    text = text.upper().replace(" ", "")
    while len(text) % 2 != 0:
        text += 'X'
    result = ''
    for i in range(0, len(text), 2):
        pair = np.array([[ord(text[i]) - 65], [ord(text[i+1]) - 65]])
        res = np.dot(key_matrix, pair) % 26
        result += chr(res[0][0] + 65) + chr(res[1][0] + 65)
    return result

def hill_decrypt(cipher, key_matrix):
    det = int(np.round(np.linalg.det(key_matrix))) % 26
    det_inv = mod_inverse(det, 26)
    if det_inv is None:
        return "Matrix not invertible!"

    adj = np.array([[key_matrix[1][1], -key_matrix[0][1]],
                    [-key_matrix[1][0], key_matrix[0][0]]])
    inv_matrix = (det_inv * adj) % 26
    inv_matrix = inv_matrix.astype(int)

    result = ''
    for i in range(0, len(cipher), 2):
        pair = np.array([[ord(cipher[i]) - 65], [ord(cipher[i+1]) - 65]])
        res = np.dot(inv_matrix, pair) % 26
        result += chr(res[0][0] + 65) + chr(res[1][0] + 65)
    return result

while True:
    print("\n===== Hill Cipher (2x2) Menu =====")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Exit")
    choice = input("Enter your choice: ")

    if choice in ['1', '2']:
        print("Enter 4 integers for 2x2 key matrix (row-wise):")
        k = list(map(int, input().split()))
        if len(k) != 4:
            print("Invalid input. Try again.")
            continue
        key_matrix = np.array(k).reshape(2, 2)
        text = input("Enter text: ")

        if choice == '1':
            print("Encrypted Text:", hill_encrypt(text, key_matrix))
        else:
            print("Decrypted Text:", hill_decrypt(text, key_matrix))

    elif choice == '3':
        print("Exiting Hill Cipher.")
        break

    else:
        print("Invalid choice. Try again.")
```

# Hill Cipher (2x2) OUTPUT :

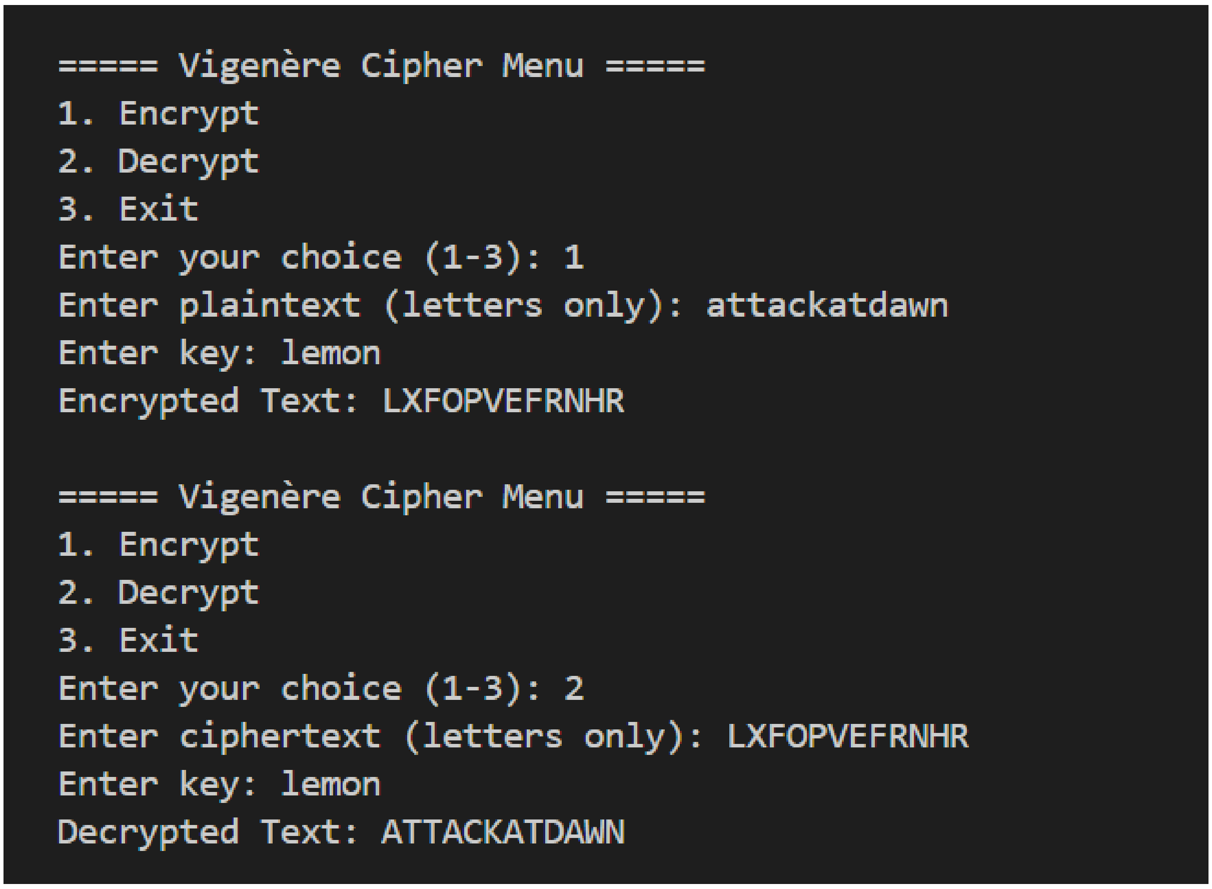
```
===== Hill Cipher (2x2) Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter 4 integers for 2x2 key matrix (row-wise):
3 3 2 5
Enter text: HELP
Encrypted Text: HIAT

===== Hill Cipher (2x2) Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter 4 integers for 2x2 key matrix (row-wise):
3 3 2 5
Enter text: HIAT
Decrypted Text: HELP
```

# Vigenère Cipher

```
def generate_key(text, key):
    key = list(key)
    if len(text) == len(key):
        return "".join(key)
    else:
        for i in range(len(text) - len(key)):
            key.append(key[i % len(key)])
    return "".join(key)

def encrypt(plaintext, key):
    cipher_text = ""
    for i in range(len(plaintext)):
        x = (ord(plaintext[i]) - ord('A') + ord(key[i]) - ord('A')) % 26
        x += ord('A')
        cipher_text += chr(x)
    return cipher_text
```

```python
def decrypt(ciphertext, key):
    orig_text = ""
    for i in range(len(ciphertext)):
        x = (ord(ciphertext[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text += chr(x)
    return orig_text

def to_uppercase(text):
    return text.upper().replace(" ", "")

while True:
    print("\n===== Vigenère Cipher Menu =====")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Exit")

    choice = input("Enter your choice (1-3): ")

    if choice == '1':
        plaintext = input("Enter plaintext (letters only): ")
        key = input("Enter key: ")
        plaintext = to_uppercase(plaintext)
        key = to_uppercase(key)
        full_key = generate_key(plaintext, key)
        cipher = encrypt(plaintext, full_key)
        print("Encrypted Text:", cipher)

    elif choice == '2':
        ciphertext = input("Enter ciphertext (letters only): ")
        key = input("Enter key: ")
        ciphertext = to_uppercase(ciphertext)
        key = to_uppercase(key)
        full_key = generate_key(ciphertext, key)
        original = decrypt(ciphertext, full_key)
        print("Decrypted Text:", original)

    elif choice == '3':
        print("Exiting program.")
        break

    else:
        print("Invalid choice. Please enter 1, 2, or 3.")
```

OUTPUT :



```
===== Vigenère Cipher Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice (1-3): 1
Enter plaintext (letters only): attackatdawn
Enter key: lemon
Encrypted Text: LXFOPVEFRNHR

===== Vigenère Cipher Menu =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice (1-3): 2
Enter ciphertext (letters only): LXFOPVEFRNHR
Enter key: lemon
Decrypted Text: ATTACKATDAWN
```