Ex. No. 2	IMPLEMENT SUBSTITUTION CIPHER
Date of Exercise	07/08/2024

Aim

To perform encryption and decryption using

a) Caesar cipher
b) Monoalphabetic cipher
c) Affine cipher
d) Vigenere cipher

Description

To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by three. Each letter is replaced by the letter three letters ahead in the alphabet. A becomes D, B becomes E, and so on. For the last letters, we can think of the alphabet as a circle and "wrap around". W becomes Z, X becomes A, Y becomes B, and Z becomes C. To change a message back, each letter is replaced by the one three before it.

A cipher is a method used to encode or encrypt a message to keep its contents secret from unauthorized parties. It involves converting the original message, known as plaintext, into a coded format, called ciphertext, using a specific algorithm or set of rules. Only those who know the correct decryption method or key can convert the ciphertext back into the original plaintext.

Algorithm

STEP-1: Read the plain text from the user.

STEP-2: Read the key value from the user.

STEP-3: If the key is positive then encrypt the text by adding the key with

each character in the plain text.

STEP-4: Else subtract the key from the plain text.

STEP-5: Display the cipher text obtained above.

Program

1) Caesar Cipher

```
def encrypt(plaintext, shift):
    encrypted_text = []
    for char in plaintext:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr((ord(char) - base + shift) % 26 + base)
            encrypted_text.append(encrypted_char)
        else:
            encrypted_text.append(char)
    return ''.join(encrypted_text)
plaintext = input("Enter the plaintext: ")
shift = int(input("Enter the shift value (an integer): "))
encrypted_text = encrypt(plaintext, shift)
print("Encrypted Text:", encrypted_text)
```

2) Monoalphabetic Cipher

```
import string
class MonoalphabeticCipher:
    def __init__(self, key):
        self.key = key
        self.alphabet = string.ascii_lowercase
        self.encryption_map = self.create_encryption_map()
```

```
        self.decryption_map = {v: k for k, v in self.encryption_map.items()}
    def create_encryption_map(self):
        encryption_map = {}
        for i, letter in enumerate(self.alphabet):
            encryption_map[letter] = self.key[i]
        return encryption_map
    def encrypt(self, plaintext):
        plaintext = plaintext.lower()
        encrypted_text = []
        for char in plaintext:
            if char in self.encryption_map:
                encrypted_text.append(self.encryption_map[char])
            else:
                encrypted_text.append(char)  # Non-alphabetic characters remain the same
        return ''.join(encrypted_text)
    def decrypt(self, ciphertext):
        ciphertext = ciphertext.lower()
        decrypted_text = []
        for char in ciphertext:
            if char in self.decryption_map:
                decrypted_text.append(self.decryption_map[char])
            else:
                decrypted_text.append(char)  # Non-alphabetic characters remain the same
```

```
        return ''.join(decrypted_text)
key = input("Enter a 26-letter key (unique letters only): ").lower()
if len(key) != 26 or len(set(key)) != 26:
    print("Invalid key. The key must be 26 unique letters.")
cipher = MonoalphabeticCipher(key)
plaintext = input("Enter the plaintext: ")
ciphertext = cipher.encrypt(plaintext)
decrypted_text = cipher.decrypt(ciphertext)
print(f"Plaintext: {plaintext}")
print(f"Ciphertext: {ciphertext}")
print(f"Decrypted Text: {decrypted_text}")
```

3) Affine Cipher

```
def mod_inverse(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return -1
def encrypt_char(ch, a, b):
    if ch.isalpha():
        base = ord('A') if ch.isupper() else ord('a')
        return chr((a * (ord(ch) - base) + b) % 26 + base)
    return ch
```

```
def decrypt_char(ch, a, b):
    if ch.isalpha():
        base = ord('A') if ch.isupper() else ord('a')
        a_inverse = mod_inverse(a, 26)
        return chr((a_inverse * (ord(ch) - base - b + 26)) % 26 + base)
    return ch
def encrypt(plaintext, a, b):
    encrypted_text = []
    for ch in plaintext:
        encrypted_text.append(encrypt_char(ch, a, b))
    return ''.join(encrypted_text)
def decrypt(ciphertext, a, b):
    decrypted_text = []
    for ch in ciphertext:
        decrypted_text.append(decrypt_char(ch, a, b))
    return ''.join(decrypted_text)


plaintext = input("Enter the plaintext: ")
a = int(input("Enter the key 'a' (must be coprime with 26): "))
b = int(input("Enter the key 'b': "))
if mod_inverse(a, 26) == -1:
    print("Invalid key 'a'. Choose a key that is coprime with 26.")
encrypted_text = encrypt(plaintext, a, b)
```

```
print("Encrypted Text:", encrypted_text)
decrypted_text = decrypt(encrypted_text, a, b)
print("Decrypted Text:", decrypted_text)
```

4) Vigenere Cipher

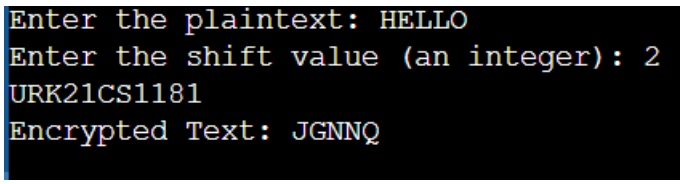
```
def encrypt(plaintext, key):
    encrypted_text = []
    key_length = len(key
    for i, char in enumerate(plaintext):
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            shift = ord(key[i % key_length].upper()) - ord('A')
            encrypted_char = chr((ord(char) - base + shift) % 26 + base)
            encrypted_text.append(encrypted_char)
        else:
            encrypted_text.append(char)
    return ''.join(encrypted_text)
def decrypt(ciphertext, key):
    decrypted_text = []
    key_length = len(key)
    for i, char in enumerate(ciphertext):
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
```

```
            shift = ord(key[i % key_length].upper()) - ord('A')
            decrypted_char = chr((ord(char) - base - shift + 26) % 26 + base)
            decrypted_text.append(decrypted_char)
        else:
            decrypted_text.append(char)
    return ''.join(decrypted_text)
plaintext = input("Enter the plaintext: ")
key = input("Enter the key: ")

encrypted_text = encrypt(plaintext, key)
print("Encrypted Text:", encrypted_text)
decrypted_text = decrypt(encrypted_text, key)
print("Decrypted Text:", decrypted_text)
```

Output Screenshot

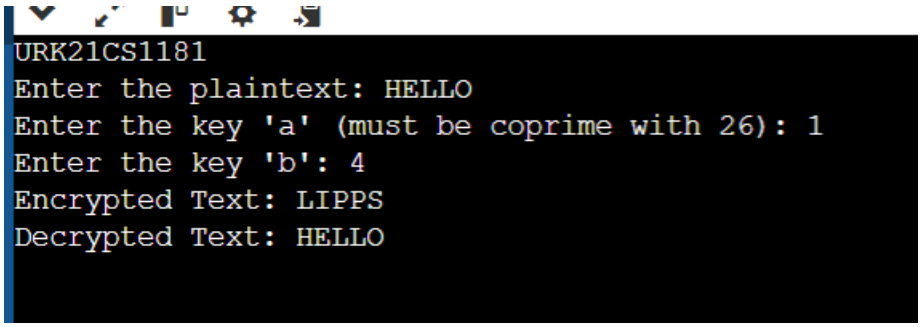
1) Caesar Cipher



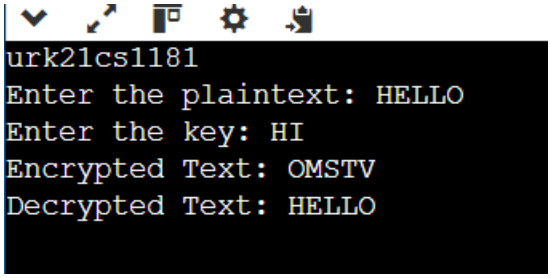
2) Monoalphabetic Cipher

```
URK21CS1181
Enter a 26-letter key (unique letters only): XBAJFLMWZQCYHNERVDIKGOPUTS
Enter the plaintext: hello
Plaintext: hello
Ciphertext: wfyye
Decrypted Text: hello
```

3) Affine Cipher



4) Vigenere Cipher



Result

Hence the python program to implement encryption decryption techniques have been coded and executed successfully.