Security Lab
Lab Assignment No. 3

Aim: Design and Implement a product cipher using Substitution ciphers.

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets.

The **Caesar Cipher** technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.

A **Product Cipher** combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components to make it resistant to cryptanalysis.

Substitution ciphers are probably the most common form of cipher. They work by replacing each letter of the plaintext (and sometimes punctuation marks and spaces) with another letter (or possibly even a random symbol).

Algorithm:

STEP 1: Ask the user to enter a plain text.
STEP 2: Generate a random number for a key to denote the required shift.
STEP 3: Call the function to encrypt the given plain text.
STEP 4: Traverse the given plain text one character at a time.
STEP 5: For each character, transform the given character as per the above required shift.
STEP 6: Return the encrypted text generated.
STEP 7: Decrypt the encrypted text using the key.
STEP 8: Generate a key in a cyclic manner until it's length isn't equal to the length of the original text.
STEP 9: Returns the encrypted text generated with the help of the key.
STEP 10: Decrypts the encrypted text and returns the original text.

Code:

```
# Function to encrypt a plain text using Caesar Cipher
def encryption(plain_text, shift):
    encrypted_text = ""
```

```
    # Traverse the plain text
    for i in range(len(plain_text)):
        char = plain_text[i]

        if (char.isalpha()):

            # Encryption of uppercase letters
            if (char.isupper()):
                encrypted_text += chr((ord(char) + shift - 65) % 26 + 65)

            # Encryption of lowercase letters
            else:
                encrypted_text += chr((ord(char) + shift - 97) % 26 + 97)

        # Keeping the whitespace as it is
        else:
            encrypted_text += char

    return encrypted_text

# Function to decrypt an encrypted text using Caesar Cipher
def decryption(key, message):
        message = message.upper()
        alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        result = ""

        # Traverse the encrypted text
        for letter in message:

                # Decryption of letters
                if letter in alpha:
                        letter_index = (alpha.find(letter) - key) % len(alpha)
                        result = result + alpha[letter_index]

                # Keeping the whitespace as it is
                else:
                        result = result + letter
```

```
        return result

# Function to generate a key for Vigenere Cipher
def vigenere_key(string, key):
        key = list(key)

        # Check whether the length of the key and string is same or not
        if len(string) == len(key):
                return(key)

        else:
                # Generate a key until its length isn't equal to the length of the original text
                for i in range(len(string) - len(key)):
                        key.append(key[i % len(key)])

        return("" . join(key))

# Function returning the encrypted text generated with the help of the key
def encrypt_vigenere(string, key):
   cipher_text = []

   for i in range(len(string)):
      if(string[i].isalpha()):
         x = (ord(string[i]) +
            ord(key[i])) % 26

         x += ord('A')
         cipher_text.append(chr(x))

      else:
         cipher_text.append(string[i])

   return("" . join(cipher_text))

# Function decrypting the encrypted text and returns the original text
def decrypt_vigenere(cipher_text, key):
   orig_text = []
```

```
    for i in range(len(cipher_text)):
        if(cipher_text[i].isalpha()):
            x = (ord(cipher_text[i]) -
                ord(key[i]) + 26) % 26

            x += ord('A')
            orig_text.append(chr(x))

        else:
            orig_text.append(cipher_text[i])

    return("" . join(orig_text))

plain_text_1 = input("Enter the plain text: ")
plain_text_1 = plain_text_1.upper()

shift = int(input("Enter shift value: "))

encrypted_text_1 = encryption(plain_text_1, shift)
N = len(encrypted_text_1)

print("Encrypted text 1: "+encrypted_text_1)

plain_text_2 = encrypted_text_1
keyword = input("Enter the key word: ")
keyword = keyword.upper()

key = vigenere_key(plain_text_2, keyword)

encrypted_text_2 = encrypt_vigenere(plain_text_2, key)
print("Encrypted text 2: "+encrypted_text_2)

decryptedText1 = decrypt_vigenere(encrypted_text_2, key)
print("Decrypted text 1: "+decryptedText1)
decryptedText2 = decryption(shift, decryptedText1)
print("Decrypted text 2: "+decryptedText2)
```

Output:

```
PS D:\CNS Lab Experiments> & "C:/Users/Ninad Rao/AppData/Local/Programs/Python/Python39/python.exe" "d:/CNS Lab Experiments/assignment3.py"
Enter the plain text: My name is Ninad Rao
Enter shift value: 5
Encrypted text 1: RD SFRJ NX SNSFI WFT
Enter the key word: nice
Encrypted text 2: EL WSZL AF WAAHM EHX
Decrypted text 1: RD SFRJ NX SNSFI WFT
Decrypted text 2: MY NAME IS NINAD RAO
```

Conclusion: Thus we understood to design and implement a Product cipher using Substitution ciphers.