AIM: Implementing Substitution and Transposition Ciphers - Design and implement algorithms to encrypt and decrypt messages using classical substitution and transposition techniques.

Program 1: Write a python program to implement Ceaser Cipher.

```
def encrypt(string, shift):
    cipher = ''
    for char in string:
        if char == ' ':
            cipher = cipher + char
        elif char.isupper():
            cipher = cipher + chr((ord(char) + shift - 65) % 26 + 65)
        else:
            cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)
    return cipher

def decrypt(string, shift):
    cipher = ''
    for char in string:
        if char == ' ':
            cipher = cipher + char
        elif char.isupper():
            cipher = cipher + chr((ord(char) + (26-shift) - 65) % 26 + 65)
        else:
            cipher = cipher + chr((ord(char) + (26-shift) - 97) % 26 + 97)
    return cipher

text = input("Enter String : ")
s = int(input("enter Shift Number : "))

option = int(input("1. For Encrypt \n2. For Decrypt\n Enter Your choice : "))

print("Original String : ", text)
if( option == 1):
    print("After Encryption : ", encrypt(text, s))
else:
    print("After Decryption : ", decrypt(text, s))
```

Output:

```
Enter String : good
enter Shift Number : 2
1. For Encrypt
2. For Decrypt
 Enter Your choice : 1
Original String :  good
After Encryption :  iqqf
```

Program 2: Write a python program to implement Mono-alphabetic Cipher.

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
keyword = "ZYXWVUTSRQPONMLKJIHGFEDCBA"
def encrypt(Plaintext):
    result = ""
    for char in Plaintext:
        if char in alphabet:
            num = alphabet.find(char)
            result += keyword[num]
        else:
            result += char
    print("Encrypted Text:", result)
def decrypt(Ciphertext):
    result = ""
    for char in Ciphertext:
        if char in keyword:
            num = keyword.find(char)
            result += alphabet[num]
        else:
            result += char
    print("Decrypted Text:", result)
while True:
    try:
        n = int(input("Enter Value:\n1) Encrypt Text\n2) Decrypt Text\n3) See Key\n4) Exit\nChoice:
"))
    except ValueError:
        print("Invalid input; please enter a number between 1 and 4.")
        continue
    if n == 1:
        Plaintext = input("Enter Text to Encrypt: ")
        encrypt(Plaintext.upper())
    elif n == 2:
        Ciphertext = input("Enter Text to Decrypt: ")
        decrypt(Ciphertext.upper())
    elif n == 3:
        print("Substitution Key (Keyword):", keyword)
    elif n == 4:
        print("Exiting the program.")
        break
    else:
        print("Invalid Input; Enter Again!!")
```

Output:

```
Enter Value:
1) Encrypt Text
2) Decrypt Text
3) See Key
4) Exit
Choice: 1
Enter Text to Encrypt: hello
Encrypted Text: SVOOL
```

Program 3: Write a python program to implement Playfair Cipher.

```
key = input("Enter key : ")
key = key.replace(" ", "")
key = key.upper()
def matrix(x, y, initial):
    return [[initial for i in range(x)] for j in range(y)]
result = list()
for c in key:
    if c not in result:
        if c == 'J':
            result.append('I')
        else:
            result.append(c)
flag = 0
for i in range(65, 91):
    if chr(i) not in result:
        if i == 73 and chr(74) not in result:
            result.append("I")
            flag = 1
        elif flag == 0 and i == 73 or i == 74:
            pass
        else:
            result.append(chr(i))
k = 0
my_matrix = matrix(5, 5, 0)
for i in range(0, 5):
    for j in range(0, 5):
        my_matrix[i][j] = result[k]
        k += 1
def locindex(c):
    loc = list()
    if c == 'J':
        c = 'I'
    for i, j in enumerate(my_matrix):
        for k, l in enumerate(j):
            if c == l:
                loc.append(i)
                loc.append(k)
                return loc
def encrypt():
    msg = str(input("ENTER MSG : "))
    msg = msg.upper()
```

```
    msg = msg.replace(" ", "")
    i = 0
    for s in range(0, len(msg) + 1, 2):
        if s < len(msg) - 1:
            if msg[s] == msg[s + 1]:
                msg = msg[:s + 1] + 'X' + msg[s + 1:]
    if len(msg) % 2 != 0:
        msg = msg[:] + 'X'
    print("CIPHER TEXT:", end=' ')
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            print("{}{}".format(my_matrix[(loc[0] + 1) % 5][loc[1]], my_matrix[(loc1[0] + 1) % 5][loc1[1]]),
end=' ')
        elif loc[0] == loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1] + 1) % 5], my_matrix[loc1[0]][(loc1[1] + 1) % 5]),
end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]], my_matrix[loc1[0]][loc[1]]), end=' ')
        i = i + 2
def decrypt():
    msg = str(input("ENTER CIPHER TEXT:"))
    msg = msg.upper()
    msg = msg.replace(" ", "")
    print("PLAIN TEXT:", end=' ')
    i = 0
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            print("{}{}".format(my_matrix[(loc[0] - 1) % 5][loc[1]], my_matrix[(loc1[0] - 1) % 5][loc1[1]]),
end=' ')
        elif loc[0] == loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1] - 1) % 5], my_matrix[loc1[0]][(loc1[1] - 1) % 5]),
end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]], my_matrix[loc1[0]][loc[1]]), end=' ')
```

```
        i = i + 2
while (1):
    choice = int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT \n Enter Your Choice: \n "))

    if choice == 1:
        encrypt()
    elif choice == 2:
        decrypt()
    elif choice == 3:
        exit()
    else:
        print("Choose correct choice")
```

Output:

```
Enter key : 2

 1.Encryption
 2.Decryption:
 3.EXIT
 Enter Your Choice:
 1
ENTER MSG : good morning
CIPHER TEXT: IM TI NK SM HO HW
```

Program 4: Write a python program to implement Vernam Cipher.

```
def Vernam(Plain,Key,Flag):
    result=""
    for i in range(len(Plain)):
        char=Plain[i]
        if(Flag):
            result+=chr((ord(char)-97 +ord(Key[i])-97)%26 +97)
        else:
            result += chr((ord(char) - ord(Key[i])+26) % 26 + 97)
    return result

if __name__=="__main__":
    Key=''.join(input("Enter Key: ").lower().split())
    Plain=''.join(input("Enter Plaintext: ").lower().split())
    if(len(Key)!=len(Plain)):
        print("Invalid Key!")
        exit(None)
    CipherText=Vernam(Plain,Key,True)
    print("CipherText: ",CipherText)
    print("PlainBack: ",Vernam(CipherText,Key,False))
```

Output:

```
Enter Key: ABCD
Enter Plaintext: BLUE
CipherText:  bmwh
PlainBack:  blue
```

Program 5: Write a python program to implement Simple Columnar Transposition Cipher.

```
import math

key = "HACK"



def encryptMessage(msg):
    cipher = ""

    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    col = len(key)

    row = int(math.ceil(msg_len / col))

    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1

    return cipher



def decryptMessage(cipher):
    msg = ""

    k_indx = 0
```

```
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    col = len(key)

    row = int(math.ceil(msg_len / col))

    key_lst = sorted(list(key))


    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]


    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

        for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1

    try:
        msg = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError("This program cannot",
                        "handle repeating words.")

    null_count = msg.count('_')

    if null_count > 0:
        return msg[: -null_count]

    return msg


msg = "Come Home Tomorrow"

cipher = encryptMessage(msg)
print("Encrypted Message: {}".
```

```
    format(cipher))

print("Decryped Message: {}".
    format(decryptMessage(cipher)))
```

Output:

```
Encrypted Message: oH owmoTr_C emoemor_
Decryped Message: Come Home Tomorrow
```

Program 6: Write a python program to implement Railfence Cipher.

```
def encryptRailFence(text, key):
    rail = [['\n' for i in range(len(text))]
                for j in range(key)]
    dir_down = False
    row, col = 0, 0
    for i in range(len(text)):
        if (row == 0) or (row == key - 1):
            dir_down = not dir_down        rail[row][col] = text[i]
        col += 1        if dir_down:
            row += 1
        else:
            row -= 1
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return("" . join(result))
def decryptRailFence(cipher, key):
    rail = [['\n' for i in range(len(cipher))]
                for j in range(key)]
    dir_down = None
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
        rail[row][col] = '*'
        col += 1
        if dir_down:
            row += 1
        else:
            row -= 1
    index = 0
    for i in range(key):
        for j in range(len(cipher)):
            if ((rail[i][j] == '*') and
            (index < len(cipher))):
                rail[i][j] = cipher[index]
                index += 1
```

```
    result = []
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key-1:
            dir_down = False
        if (rail[row][col] != '*'):
            result.append(rail[row][col])
            col += 1
        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))
if __name__ == "__main__":
    print(encryptRailFence("attack at once", 2))
    print(encryptRailFence("defend the east wall", 3))
    print(decryptRailFence("atc toctaka ne", 2))
    print(decryptRailFence("dnhaweedtees alf  tl", 3))
```

Output:

```
atc toctaka ne
dnhaweedtees alf  tl
attack at once
defend the east wall
```