**Q Write a Python program to accept a plain text message and encrypt it using Caesar Cipher technique.**

def caesar\_encrypt(plaintext, shift):

ciphertext = []

for ch in plaintext:

if ch.isalpha():

base = 'A' if ch.isupper() else 'a'

ciphertext.append(chr((ord(ch) - ord(base) + shift) % 26 + ord(base)))

else:

ciphertext.append(ch)

return ''.join(ciphertext)

if \_\_name\_\_ == "\_\_main\_\_":

msg = input("Plaintext: ")

s = int(input("Shift (0-25): "))

print("Ciphertext:", caesar\_encrypt(msg, s))

**Q Write a Python program to implement Playfair Cipher technique for**

**Decryption.**

def table(key):

key = key.upper().replace('J','I')

seq = "".join(dict.fromkeys(key + "ABCDEFGHIKLMNOPQRSTUVWXYZ"))

return [list(seq[i:i+5]) for i in range(0,25,5)]

def pos(tab, ch):

for i in range(5):

for j in range(5):

if tab[i][j] == ch: return i,j

def decrypt(ct, key):

tab, ct = table(key), ct.upper().replace('J','I')

pt, i = "", 0

while i < len(ct):

a,b = ct[i], ct[i+1] if i+1<len(ct) else 'X'

r1,c1 = pos(tab,a); r2,c2 = pos(tab,b)

if r1==r2: pt += tab[r1][(c1-1)%5]+tab[r2][(c2-1)%5]

elif c1==c2: pt += tab[(r1-1)%5][c1]+tab[(r2-1)%5][c2]

else: pt += tab[r1][c2]+tab[r2][c1]

i += 2

return pt

if \_\_name\_\_=="\_\_main\_\_":

print("Plaintext:", decrypt(input("Cipher: "), input("Key: "))) #ct=bm

**Q Encrypt the message "HELLO" using the Simple Columnar Transposition**

**Technique with the key "4321".**

def columnar\_encrypt(text, key):

text = text.upper().replace(" ", "")

n = len(key)

rows = (len(text) + n - 1) // n

grid = [['X'] \* n for \_ in range(rows)]

k = 0

for r in range(rows):

for c in range(n):

if k < len(text):

grid[r][c] = text[k]

k += 1

order = sorted(list(enumerate(key)), key=lambda x: x[1])

cipher = ""

for idx, \_ in order:

for r in range(rows):

cipher += grid[r][idx]

return cipher

plaintext = "HELLO"

key = "4321"

print("Ciphertext:", columnar\_encrypt(plaintext, key))

**Q Encrypt the message "HELLO" using the Simple Columnar Transposition**

**Technique with the key "4321".**

def columnar\_encrypt(text, key):

text = text.upper().replace(" ", "")

n = len(key)

rows = (len(text) + n - 1) // n

grid = [['X'] \* n for \_ in range(rows)]

k = 0

for r in range(rows):

for c in range(n):

if k < len(text):

grid[r][c] = text[k]

k += 1

order = sorted(list(enumerate(key)), key=lambda x: x[1])

cipher = ""

for idx, \_ in order:

for r in range(rows):

cipher += grid[r][idx]

return cipher

plaintext = "HELLO"

key = "4321"

print("Ciphertext:", columnar\_encrypt(plaintext, key))

**QWrite a Python program to accept a plain text message and encrypt it using**

**Monoalphabetic Cipher.**

import string

def mono\_encrypt(text, key):

alpha = string.ascii\_uppercase

mapping = {alpha[i]: key[i] for i in range(26)}

result = ""

for ch in text.upper():

result += mapping.get(ch, ch) # substitute if alphabet, else keep same

return result

plaintext = input("Enter plaintext: ")

key = "QWERTYUIOPASDFGHJKLZXCVBNM" # substitution key (example)

print("Ciphertext:", mono\_encrypt(plaintext, key))

**QWrite a Python Program to implement Rail fence transposition technique to**

**encrypt plain text messages.**

def rail\_fence\_encrypt(text, key):

fence = [[] for \_ in range(key)]

rail, var = 0, 1

for ch in text:

fence[rail].append(ch)

rail += var

if rail == 0 or rail == key-1:

var = -var

return ''.join(''.join(row) for row in fence)

plaintext = input("Enter plaintext: ")

key = int(input("Enter key (number of rails): "))

cipher = rail\_fence\_encrypt(plaintext.replace(" ", ""), key)

print("Ciphertext:", cipher)

**Q Write a Python program to implement the DES algorithm to encrypt a**

**message.**

def toy\_des\_encrypt(key, text):

k = sum(ord(c) for c in key) % 256

return ''.join(chr((ord(c) ^ k) ^ (i & 0xFF)) for i, c in enumerate(text))

def toy\_des\_decrypt(key, text):

k = sum(ord(c) for c in key) % 256

return ''.join(chr((ord(c) ^ (i & 0xFF)) ^ k) for i, c in enumerate(text))

key = input("Enter key: ")

plaintext = input("Enter plaintext: ")

cipher = toy\_des\_encrypt(key, plaintext)

cipher\_hex = cipher.encode().hex() # convert ciphertext to hex for clean display

print("Ciphertext (hex):", cipher\_hex)

cipher\_bytes = bytes.fromhex(cipher\_hex)

plain = toy\_des\_decrypt(key, cipher\_bytes.decode())

print("Decrypted:", plain)

**Q Write a Python program to implement the Diffie Hellman Key ExchangeAlgorithms.**

p = int(input("Enter a prime number (p): "))

g = int(input("Enter a primitive root (g): "))

a = int(input("Alice private key: "))

b = int(input("Bob private key: "))

A = pow(g, a, p)

B = pow(g, b, p)

alice\_shared = pow(B, a, p)

bob\_shared = pow(A, b, p)

print("Alice's Public Key:", A)

print("Bob's Public Key:", B)

print("Shared Secret Key (Alice):", alice\_shared)

print("Shared Secret Key (Bob):", bob\_shared)

**Q Write a Python Program to implement Single Columnar transposition**

**technique to decrypt cipher text to plain text message.**

def single\_columnar\_decrypt(cipher, key):

n = len(key)

rows = (len(cipher) + n - 1) // n

order = sorted(list(enumerate(key)), key=lambda x: x[1])

grid = [[''] \* n for \_ in range(rows)]

k = 0

for idx, \_ in order:

for r in range(rows):

if k < len(cipher):

grid[r][idx] = cipher[k]

k += 1

return ''.join(''.join(row) for row in grid).rstrip('X')

ciphertext = input("Enter ciphertext: ") #ct=etts

key = input("Enter key (digits): ") #key=21

plaintext = single\_columnar\_decrypt(ciphertext, key)

print("Plaintext:", plaintext)