**EXPERIMENT 3**

**Single Columnar Cipher:**

**Code:**

import math

def encrypt(message, key):

# Remove spaces and convert message to uppercase

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

# Determine number of columns and rows

cols = len(key)

rows = math.ceil(len(message) / cols)

# Add padding if necessary

message += 'X' \* (rows \* cols - len(message))

# Create a matrix and populate it row-wise

matrix = [list(message[i \* cols: (i + 1) \* cols]) for i in range(rows)]

# Get the order of columns based on the sorted key

sorted\_indices = sorted(range(len(key)), key=lambda k: key[k])

# Read the matrix column-wise according to sorted key order

ciphertext = ''.join(''.join(matrix[row][col] for row in range(rows)) for col in sorted\_indices)

return ciphertext

def decrypt(ciphertext, key):

# Determine number of columns and rows

cols = len(key)

rows = len(ciphertext) // cols

# Get the order of columns based on the sorted key

sorted\_indices = sorted(range(len(key)), key=lambda k: key[k])

# Create an empty matrix

matrix = [[''] \* cols for \_ in range(rows)]

# Fill the matrix column-wise using the sorted column order

idx = 0

for col in sorted\_indices:

for row in range(rows):

matrix[row][col] = ciphertext[idx]

idx += 1

# Read the matrix row-wise to reconstruct the plaintext

plaintext = ''.join(''.join(row) for row in matrix).rstrip('X') # Remove padding

return plaintext

# Take user input

message = input("Enter the plaintext: ")

key = input("Enter the key: ")

# Encrypt the message

cipher = encrypt(message, key)

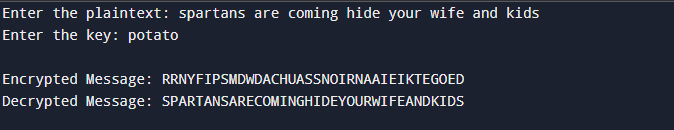
print("\nEncrypted Message:", cipher)

# Decrypt the message

decrypted = decrypt(cipher, key)

print("Decrypted Message:", decrypted)

**Output:**



**Double Transposition:**

**Code:**

import math

def columnar\_encrypt(message, key):

"""Encrypts a message using a single columnar transposition cipher."""

message = message.replace(" ", "").upper() # Remove spaces & convert to uppercase

cols = len(key)

rows = math.ceil(len(message) / cols)

# Add padding if necessary

message += 'X' \* (rows \* cols - len(message))

# Create a matrix row-wise

matrix = [list(message[i \* cols:(i + 1) \* cols]) for i in range(rows)]

# Get column order based on sorted key

sorted\_indices = sorted(range(len(key)), key=lambda k: key[k])

# Read the matrix column-wise based on key order

ciphertext = ''.join(''.join(matrix[row][col] for row in range(rows)) for col in sorted\_indices)

return ciphertext

def columnar\_decrypt(ciphertext, key):

"""Decrypts a message using a single columnar transposition cipher."""

cols = len(key)

rows = len(ciphertext) // cols

# Get column order based on sorted key

sorted\_indices = sorted(range(len(key)), key=lambda k: key[k])

# Create an empty matrix

matrix = [[''] \* cols for \_ in range(rows)]

# Fill the matrix column-wise using sorted key order

idx = 0

for col in sorted\_indices:

for row in range(rows):

matrix[row][col] = ciphertext[idx]

idx += 1

# Read the matrix row-wise to reconstruct the plaintext

plaintext = ''.join(''.join(row) for row in matrix).rstrip('X') # Remove padding

return plaintext

def double\_transposition\_encrypt(message, key1, key2):

"""Encrypts a message using double columnar transposition."""

first\_pass = columnar\_encrypt(message, key1)

second\_pass = columnar\_encrypt(first\_pass, key2)

return second\_pass

def double\_transposition\_decrypt(ciphertext, key1, key2):

"""Decrypts a message using double columnar transposition."""

first\_pass = columnar\_decrypt(ciphertext, key2)

second\_pass = columnar\_decrypt(first\_pass, key1)

return second\_pass

# Take user input

message = input("Enter the plaintext: ")

key1 = input("Enter the first key: ")

key2 = input("Enter the second key: ")

# Encrypt the message

ciphertext = double\_transposition\_encrypt(message, key1, key2)

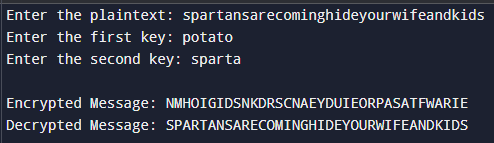
print("\nEncrypted Message:", ciphertext)

# Decrypt the message

decrypted\_text = double\_transposition\_decrypt(ciphertext, key1, key2)

print("Decrypted Message:", decrypted\_text)

**Output:**

****