1. **Assume you intercepted the following ciphertext. Using a statistical attack, find the plaintext**

**"XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILYVVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW"**

cipher = "XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILYVVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW"

frequency = {}

for char in cipher:

if char in frequency:

frequency[char] += 1

else:

frequency[char] = 1

frequency = dict(sorted(frequency.items(), key=lambda item: item[1], reverse=True))

english\_frequency = "ETAOINSHRDLCUMWFGYPBVKJXQZ"

replacement\_mapping = {}

for i, char in enumerate(frequency):

replacement\_mapping[char] = english\_frequency[i]

plain\_text = "".join(replacement\_mapping.get(char, char) for char in cipher)

print(frequency)

print(replacement\_mapping)

print(cipher)

print(plain\_text)

1. **Write a Python script to encrypt using Rail Fence (Zig zag ) with three rows and with key (ONE).**

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

# Create the rail fence matrix

fence = [[' ' for \_ in range(len(message))] for \_ in range(3)]

# Fill the rail fence

row, direction = 0, 1

for i, char in enumerate(message):

fence[row][i] = char

row += direction

if row == 0 or row == 2:

direction \*= -1

# Read off the fence

cipher = ''

for i, letter in enumerate(key):

for j in range(len(message)):

if fence[i][j] != ' ':

cipher += fence[i][j]

return cipher

# Example usage

message = input("Enter the message to encrypt: ")

key = "ONE"

encrypted\_message = rail\_fence\_encrypt(message, key)

print(f"Encrypted message: {encrypted\_message}")

1. **Write a python script to encrypt columnar transposition**

def columnar\_transposition\_encrypt(message, key):

# Remove spaces and convert to uppercase

message = ''.join(message.split()).upper()

key = ''.join(key.split()).upper()

# Calculate number of rows needed

num\_rows = -(-len(message) // len(key)) # Ceiling division

# Pad the message if necessary

message += 'X' \* (num\_rows \* len(key) - len(message))

# Create the grid

grid = [[''] \* len(key) for \_ in range(num\_rows)]

# Fill the grid with the message

index = 0

for row in range(num\_rows):

for col in range(len(key)):

grid[row][col] = message[index]

index += 1

# Sort the columns based on the key

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

# Read off the columns to get the ciphertext

ciphertext = ''

for col in sorted\_columns:

ciphertext += ''.join(grid[row][col] for row in range(num\_rows))

return ciphertext

# Example usage

message = input("Enter the message to encrypt: ")

key = input("Enter the encryption key: ")

encrypted\_message = columnar\_transposition\_encrypt(message, key)

print(f"Encrypted message: {encrypted\_message}")

1. **Write a Python script to decrypt Rail Fence Cipher.**

