**Lab 6**

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

**"XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILYVVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW"**

from collections import Counter

# Ciphertext to analyze

ciphertext = "XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILYVVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW"

# Frequency analysis function

def frequency\_analysis(text):

# Clean text: remove non-alphabetic characters and convert to upper case

text = ''.join(filter(str.isalpha, text)).upper()

# Count letter frequency

freq = Counter(text)

return freq.most\_common()

# Function to create a simple substitution based on frequency

def substitute\_text(ciphertext, substitution):

return ''.join(substitution.get(char, char) for char in ciphertext)

# Perform frequency analysis

freq = frequency\_analysis(ciphertext)

# Display the frequency of letters

print("Letter Frequency:")

for letter, count in freq:

print(f"{letter}: {count}")

# Hypothetical substitution mapping based on frequency analysis

substitution\_mapping = {

'X': 'T',

'L': 'H',

'I': 'E',

'S': 'A',

'Y': 'O',

'W': 'N',

'M': 'I',

'R': 'S',

'A': 'R',

'J': 'D',

'V': 'L',

'P': 'U',

'Q': 'Y',

'T': 'G', # Example (may not appear in the text)

# Add more mappings as needed

}

# Substitute the ciphertext using the mapping

plaintext = substitute\_text(ciphertext, substitution\_mapping)

print("\nPossible Plaintext:")

print(plaintext)



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

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

# Remove spaces and convert to uppercase

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

num\_rows = len(key)

rail = [['\n' for \_ in range(len(plaintext))] for \_ in range(num\_rows)]

# Fill the rail matrix

row, step = 0, 1

for char in plaintext:

rail[row].append(char)

if row == 0:

step = 1 # Move down

elif row == num\_rows - 1:

step = -1 # Move up

row += step

# Read off the characters in the zig-zag pattern

ciphertext = ''.join(''.join(row).replace('\n', '') for row in rail)

return ciphertext

# Input parameters

plaintext = "HELLO WORLD"

key = "ONE"

# Encrypt the plaintext

ciphertext = rail\_fence\_encrypt(plaintext, key)

print("Encrypted Ciphertext:", ciphertext)



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

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

# Remove spaces and convert to uppercase

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

# Calculate the number of columns

key\_length = len(key)

num\_columns = (len(plaintext) + key\_length - 1) // key\_length # Ceiling division

# Create a grid with padding (if necessary)

grid = [''] \* key\_length

for i in range(len(plaintext)):

grid[i % key\_length] += plaintext[i]

# Sort the key and get the order of columns

sorted\_key = sorted((char, index) for index, char in enumerate(key))

order = [index for char, index in sorted\_key]

# Read the columns in the order specified by the sorted key

ciphertext = ''.join(grid[i] for i in order)

return ciphertext

# Input parameters

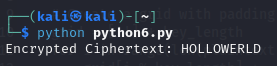
plaintext = "HELLO WORLD"

key = "COLUMN"

# Encrypt the plaintext

ciphertext = columnar\_transposition\_encrypt(plaintext, key)

print("Encrypted Ciphertext:", ciphertext)



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

def rail\_fence\_decrypt(ciphertext, key):

# Calculate the length of the ciphertext

n = len(ciphertext)

# Create an empty rail

rail = [['\n' for \_ in range(n)] for \_ in range(key)]

# Determine the direction of filling the rail

row, col = 0, 0

down = True # Initially moving downwards

# Mark the positions where characters will be placed

for i in range(n):

if row == 0:

down = True # Change direction to down

if row == key - 1:

down = False # Change direction to up

rail[row][col] = '\*'

col += 1

# Move up or down

row += 1 if down else -1

# Fill the rail with the ciphertext characters

index = 0

for i in range(key):

for j in range(n):

if rail[i][j] == '\*' and index < n:

rail[i][j] = ciphertext[index]

index += 1

# Read the rail in zig-zag to get the plaintext

result = []

row, col = 0, 0

for i in range(n):

if row == 0:

down = True # Change direction to down

if row == key - 1:

down = False # Change direction to up

result.append(rail[row][col])

col += 1

# Move up or down

row += 1 if down else -1

return ''.join(result)

# Input parameters

ciphertext = "HOLELWRDLO"

key = 3

# Decrypt the ciphertext

plaintext = rail\_fence\_decrypt(ciphertext, key)

print("Decrypted Plaintext:", plaintext)



**===============================================================================**

**"It does not matter how slowly you go so long as you do not stop." —Confucius**

**===============================================================================**