**Python-based Text Encryption and Decryption Toolki**

**#Part 1: Encryption/Decryption using Polyalphabetic Ciphers:**

# Function to encrypt plaintext using a substitution cipher

def encrypt\_substitution(plaintext, cipher):

encrypted\_text = ""

for char in plaintext:

if char.isalpha():

idx = ord(char.lower()) - ord('a')

encrypted\_text += cipher[idx]

else:

encrypted\_text += char

return encrypted\_text

# Function to decrypt ciphertext using a substitution cipher

def decrypt\_substitution(ciphertext, cipher):

decrypted\_text = ""

for char in ciphertext:

if char.isalpha():

idx = cipher.index(char.lower())

decrypted\_text += chr(idx + ord('a'))

else:

decrypted\_text += char

return decrypted\_text

# Function to apply multiple substitution ciphers based on a cycling pattern

def encrypt\_with\_multiple\_ciphers(text, ciphers):

result = ""

num\_ciphers = len(ciphers)

for i, char in enumerate(text):

cipher\_idx = i % num\_ciphers

cipher = ciphers[cipher\_idx]

result += encrypt\_substitution(char, cipher)

return result

def main():

# Example usage

M1\_cipher = "defghijklmnopqrstuvwxyzabc"

M2\_cipher = "dkvqfibjwpescxhtmynuoralgz"

M3\_cipher = "wxyzabcdefghijklmnopqrstuv"

input\_text = "hello world"

# Encrypt the input text using multiple ciphers based on the given cycling pattern

encrypted\_text = encrypt\_with\_multiple\_ciphers(input\_text, [M2\_cipher, M3\_cipher, M2\_cipher, M1\_cipher, M3\_cipher])

print("Encrypted text:", encrypted\_text)

# Decrypt the encrypted text using the same cycling pattern and ciphers

decrypted\_text = encrypt\_with\_multiple\_ciphers(encrypted\_text, [M2\_cipher, M3\_cipher, M2\_cipher, M1\_cipher, M3\_cipher])

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

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

main()

**Part 2: Encryption/Decryption using Rail Fence Cipher**

# Function to encrypt plaintext using Rail Fence Cipher

def encrypt\_rail\_fence(plaintext, depth):

rails = [[] for \_ in range(depth)]

rail\_idx = 0

direction = 1

for char in plaintext:

rails[rail\_idx].append(char)

if rail\_idx == 0:

direction = 1

elif rail\_idx == depth - 1:

direction = -1

rail\_idx += direction

encrypted\_text = "".join("".join(rail) for rail in rails)

return encrypted\_text

# Function to decrypt ciphertext using Rail Fence Cipher

def decrypt\_rail\_fence(ciphertext, depth):

rail\_lengths = [0] \* depth

rail\_idx = 0

direction = 1

for i in range(len(ciphertext)):

rail\_lengths[rail\_idx] += 1

if rail\_idx == 0:

direction = 1

elif rail\_idx == depth - 1:

direction = -1

rail\_idx += direction

rails = [ciphertext[:rail\_lengths[i]] for i in range(depth)]

ciphertext = ciphertext[rail\_lengths[0]:]

for i in range(1, depth - 1):

rails[i] += ciphertext[:rail\_lengths[i]]

ciphertext = ciphertext[rail\_lengths[i]:]

rails[depth - 1] += ciphertext

decrypted\_text = ""

rail\_idx = 0

direction = 1

for \_ in range(len(ciphertext)):

decrypted\_text += rails[rail\_idx][0]

rails[rail\_idx] = rails[rail\_idx][1:]

if rail\_idx == 0:

direction = 1

elif rail\_idx == depth - 1:

direction = -1

rail\_idx += direction

return decrypted\_text

def main():

# Example usage

input\_text = "hello world"

depth = 3

# Encrypt the input text using Rail Fence Cipher

encrypted\_text = encrypt\_rail\_fence(input\_text, depth)

print("Encrypted text:", encrypted\_text)

# Decrypt the encrypted text using Rail Fence Cipher

decrypted\_text = decrypt\_rail\_fence(encrypted\_text, depth)

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

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

main()