**CSA5103 CRYPTOGRAPHY AND NETWORK SECURITY FOR CYBER SECURITY**

**1.Write a Python program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.**

def caesar\_cipher\_encrypt(text, shift):

result = ""

for char in text:

if char.isalpha():

start = ord('A') if char.isupper() else ord('a')

shifted = (ord(char) - start + shift) % 26 + start

result += chr(shifted)

else:

result += char

return result

def main():

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

while True:

try:

k = int(input("Enter the key value (1-25): "))

if 1 <= k <= 25:

break

else:

print("Key must be between 1 and 25.")

except ValueError:

print("Please enter a valid integer.")

encrypted\_text = caesar\_cipher\_encrypt(text, k)

print(f"\nEncrypted Text with key {k}: {encrypted\_text}")

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

main()

**2.Write a Python program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.**

def monoalphabetic\_encrypt(plaintext, key):

alphabet = 'abcdefghijklmnopqrstuvwxyz'

key\_map = {plain: cipher for plain, cipher in zip(alphabet, key.lower())}

ciphertext = ''

for char in plaintext:

if char.isalpha():

is\_upper = char.isupper()

mapped\_char = key\_map[char.lower()]

ciphertext += mapped\_char.upper() if is\_upper else mapped\_char

else:

ciphertext += char

return ciphertext

def main():

plaintext = input("Enter the plaintext: ")

while True:

key = input("Enter the 26-letter key (cipher alphabet): ")

if len(key) == 26 and len(set(key.lower())) == 26 and key.isalpha():

break

else:

print("Invalid key. It must be 26 unique letters.")

encrypted = monoalphabetic\_encrypt(plaintext, key)

print(f"\nEncrypted Text:\n{encrypted}")

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

main()

**3.Write a python program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.**

def generate\_matrix(keyword):

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

matrix = ''.join(dict.fromkeys(keyword.upper().replace('J', 'I') + alphabet).keys())

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

def find\_position(matrix, char):

for r, row in enumerate(matrix):

if char in row: return r, row.index(char)

def preprocess\_text(text):

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

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

return ''.join([text[i] + ('X' if i + 1 < len(text) and text[i] == text[i+1] else '') for i in range(0, len(text), 2)])

def encrypt(plaintext, matrix):

return ''.join([matrix[r1][(c1 + 1) % 5] + matrix[r2][(c2 + 1) % 5] if r1 == r2 else matrix[(r1 + 1) % 5][c1] + matrix[(r2 + 1) % 5][c2] for (a, b) in zip(plaintext[::2], plaintext[1::2]) for r1, c1 in [find\_position(matrix, a)] for r2, c2 in [find\_position(matrix, b)]])

def main():

keyword, plaintext = input("Enter keyword: "), input("Enter plaintext: ")

matrix = generate\_matrix(keyword)

processed\_text = preprocess\_text(plaintext)

print("\nMatrix:\n" + '\n'.join(' '.join(row) for row in matrix))

print(f"\nEncrypted Text: {encrypt(processed\_text, matrix)}")

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

main()

**4.a. Write a C program to Encrypt the message “meet me at the usual place at ten rather than eight**

**oclock” using the Hill cipher with the key.**

**9 4**

**5 7**

**a. Show your calculations and the result.**

**b. Show the calculations for the corresponding decryption of the ciphertext to recover the original**

**plaintext.**

def mod\_inverse(a, m):

a = a % m

for x in range(1, m):

if (a \* x) % m == 1:

return x

return None

def matrix\_mod\_inv(matrix, modulus):

a, b = matrix[0]

c, d = matrix[1]

det = (a \* d - b \* c) % modulus

det\_inv = mod\_inverse(det, modulus)

if det\_inv is None:

raise Exception("Matrix is not invertible")

inv\_matrix = [

[(d \* det\_inv) % modulus, (-b \* det\_inv) % modulus],

[(-c \* det\_inv) % modulus, (a \* det\_inv) % modulus]

]

return inv\_matrix

def text\_to\_numbers(text):

return [ord(c) - ord('a') for c in text]

def numbers\_to\_text(numbers):

return ''.join([chr(n + ord('a')) for n in numbers])

def hill\_encrypt(plaintext, key\_matrix):

plaintext = ''.join([c for c in plaintext.lower() if c.isalpha()])

if len(plaintext) % 2 != 0:

plaintext += 'x'

ciphertext = ''

for i in range(0, len(plaintext), 2):

pair = text\_to\_numbers(plaintext[i:i+2])

c1 = (key\_matrix[0][0]\*pair[0] + key\_matrix[0][1]\*pair[1]) % 26

c2 = (key\_matrix[1][0]\*pair[0] + key\_matrix[1][1]\*pair[1]) % 26

ciphertext += numbers\_to\_text([c1, c2])

return ciphertext

def hill\_decrypt(ciphertext, key\_matrix):

inv\_key = matrix\_mod\_inv(key\_matrix, 26)

plaintext = ''

for i in range(0, len(ciphertext), 2):

pair = text\_to\_numbers(ciphertext[i:i+2])

p1 = (inv\_key[0][0]\*pair[0] + inv\_key[0][1]\*pair[1]) % 26

p2 = (inv\_key[1][0]\*pair[0] + inv\_key[1][1]\*pair[1]) % 26

plaintext += numbers\_to\_text([p1, p2])

return plaintext

key = [

[9, 4],

[5, 7]

]

message = "meet me at the usual place at ten rather than eight oclock"

cipher = hill\_encrypt(message, key)

print("Ciphertext:", cipher)

plain = hill\_decrypt(cipher, key)

print("Decrypted:", plain)

**5. Write a C program for one-time pad version of the Vigenère cipher. In this scheme, the key is a**

**stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of**

**plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5**

**letters, and so on.**

import random

import string

def generate\_key(length):

return [random.randint(0, 25) for \_ in range(length)]

def encrypt(plaintext, key):

ciphertext = ""

for i, char in enumerate(plaintext):

if char.isalpha():

offset = ord('A') if char.isupper() else ord('a')

shift = key[i]

encrypted\_char = chr((ord(char) - offset + shift) % 26 + offset)

ciphertext += encrypted\_char

else:

ciphertext += char

return ciphertext

plaintext = "cryptography"

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

key = generate\_key(len(plaintext))

ciphertext = encrypt(plaintext, key)

print("Plaintext: ", plaintext)

print("Key: ", key)

print("Ciphertext:", ciphertext)