

LAJPAT RAI | BSCS-21f-11

CIS

Sir Atif Jamil

16-Nov-2024

Assignment # 01

**Computer and information security**

Dawood University of Engineering and Technology

**(1) mono alphabetic**

## **a) Additive Cypher**

def preprocess\_text(text, remove\_spaces=True):

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

    if remove\_spaces:

        text = text.replace(" ", "")

    return text.upper()

def additive\_encrypt(plain\_text, key):

    plain\_text = preprocess\_text(plain\_text)

    cipher\_text = ""

    for char in plain\_text:

        shifted = (ord(char) - ord('A') + key) % 26

        encrypted\_char = chr(shifted + ord('A'))

        cipher\_text += encrypted\_char

    return cipher\_text

def additive\_decrypt(cipher\_text, key):

    cipher\_text = preprocess\_text(cipher\_text)

    plain\_text = ""

    for char in cipher\_text:

        shifted = (ord(char) - ord('A') - key) % 26

        decrypted\_char = chr(shifted + ord('A'))

        plain\_text += decrypted\_char

    return plain\_text

plain\_text = input("Enter plain text: ")

key = int(input("Enter key (0-25): "))

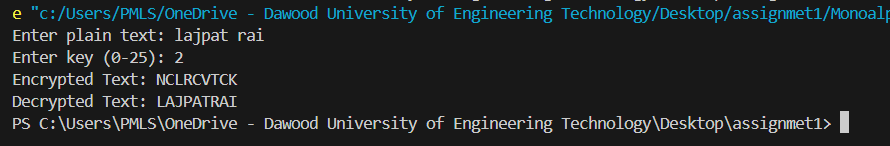
encrypted\_text = additive\_encrypt(plain\_text, key)

decrypted\_text = additive\_decrypt(encrypted\_text, key)

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

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

**OUTPUT**

****

## **b) Multiplicative Cypher**

def mod\_inverse(a, m):

    for x in range(m):

        if (a \* x) % m == 1:

            return x

    return -1

def preprocess\_text(text):

    return ''.join([c.upper() for c in text if c.isalpha()])

def multiplicative\_encrypt(plain\_text, key):

    plain\_text = preprocess\_text(plain\_text)

    cipher\_text = ""

    for char in plain\_text:

        encrypted\_char = chr(((ord(char) - ord('A')) \* key % 26) + ord('A'))

        cipher\_text += encrypted\_char

    return cipher\_text

def multiplicative\_decrypt(cipher\_text, key):

    cipher\_text = preprocess\_text(cipher\_text)

    inverse\_key = mod\_inverse(key, 26)

    if inverse\_key == -1:

        raise ValueError("No modular inverse exists for the provided key.")

    plain\_text = ""

    for char in cipher\_text:

        decrypted\_char = chr(((ord(char) - ord('A')) \* inverse\_key % 26) + ord('A'))

        plain\_text += decrypted\_char

    return plain\_text

plain\_text = input("Enter plain text: ")

key = int(input("Enter key (must be coprime with 26): "))

encrypted\_text = multiplicative\_encrypt(plain\_text, key)

decrypted\_text = multiplicative\_decrypt(encrypted\_text, key)

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

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

**A screen shot of a computer

Description automatically generatedOUTPUT**

## **c) Affine Cypher**

def preprocess\_text(text):

    return ''.join([c.upper() for c in text if c.isalpha()])

def mod\_inverse(a, m):

    for i in range(1, m):

        if (a \* i) % m == 1:

            return i

    return None

def affine\_encrypt(plain\_text, key1, key2):

    plain\_text = preprocess\_text(plain\_text)

    cipher\_text = ''.join([chr(((ord(c) - ord('A')) \* key1 + key2) % 26 + ord('A')) for c in plain\_text])

    return cipher\_text

def affine\_decrypt(cipher\_text, key1, key2):

    cipher\_text = preprocess\_text(cipher\_text)

    inverse\_key1 = mod\_inverse(key1, 26)

    if inverse\_key1 is None:

        return "Error: No modular inverse found for the key"

    plain\_text = ''.join([chr((inverse\_key1 \* (ord(c) - ord('A') - key2)) % 26 + ord('A')) for c in cipher\_text])

    return plain\_text

plain\_text = input("Enter plain text: ")

key1 = int(input("Enter key1 (multiplicative key): "))

key2 = int(input("Enter key2 (additive key): "))

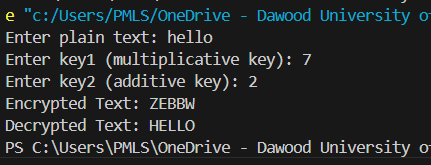
encrypted\_text = affine\_encrypt(plain\_text, key1, key2)

decrypted\_text = affine\_decrypt(encrypted\_text, key1, key2)

print(f"Encrypted Text: {encrypted\_text}")

print(f"Decrypted Text: {decrypted\_text}")

**OUTPUT**

****

**(2) poly alphabetic**

## **a) Playfair Cypher**

import numpy as np

# Helper function to preprocess the text

def preprocess\_text(text, remove\_spaces=True):

    text = text.replace(" ", "") if remove\_spaces else text

    return ''.join([c.upper() for c in text if c.isalpha()])

# Helper function to create the 5x5 Playfair key matrix

def create\_playfair\_matrix(key):

    key = preprocess\_text(key)

    key\_matrix = []

    used\_letters = set()

    for char in key + "ABCDEFGHIKLMNOPQRSTUVWXYZ":  # J is usually merged with I

        if char not in used\_letters and char != 'J':

            key\_matrix.append(char)

            used\_letters.add(char)

    return np.array(key\_matrix).reshape(5, 5)

# Helper function to format plaintext for Playfair cipher

def format\_playfair\_text(text):

    text = preprocess\_text(text)

    formatted\_text = ""

    i = 0

    while i < len(text):

        if i == len(text) - 1:

            formatted\_text += text[i] + 'X'

            i += 1

        elif text[i] == text[i + 1]:

            formatted\_text += text[i] + 'X'

            i += 1

        else:

            formatted\_text += text[i] + text[i + 1]

            i += 2

    return formatted\_text

# Function to find position of a letter in the matrix

def find\_position(matrix, letter):

    for row in range(5):

        for col in range(5):

            if matrix[row][col] == letter:

                return row, col

    return None, None

# Playfair encryption

def playfair\_encrypt(plain\_text, key):

    matrix = create\_playfair\_matrix(key)

    plain\_text = format\_playfair\_text(plain\_text)

    cipher\_text = ""

    for i in range(0, len(plain\_text), 2):

        r1, c1 = find\_position(matrix, plain\_text[i])

        r2, c2 = find\_position(matrix, plain\_text[i + 1])

        if r1 == r2:

            cipher\_text += matrix[r1][(c1 + 1) % 5] + matrix[r2][(c2 + 1) % 5]

        elif c1 == c2:

            cipher\_text += matrix[(r1 + 1) % 5][c1] + matrix[(r2 + 1) % 5][c2]

        else:

            cipher\_text += matrix[r1][c2] + matrix[r2][c1]

    return cipher\_text

# Playfair decryption

def playfair\_decrypt(cipher\_text, key):

    matrix = create\_playfair\_matrix(key)

    plain\_text = ""

    for i in range(0, len(cipher\_text), 2):

        r1, c1 = find\_position(matrix, cipher\_text[i])

        r2, c2 = find\_position(matrix, cipher\_text[i + 1])

        if r1 == r2:

            plain\_text += matrix[r1][(c1 - 1) % 5] + matrix[r2][(c2 - 1) % 5]

        elif c1 == c2:

            plain\_text += matrix[(r1 - 1) % 5][c1] + matrix[(r2 - 1) % 5][c2]

        else:

            plain\_text += matrix[r1][c2] + matrix[r2][c1]

    return plain\_text

# Input and Output demonstration

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

    plain\_text = input("Enter the plain text: ")

    key = input("Enter the key: ")

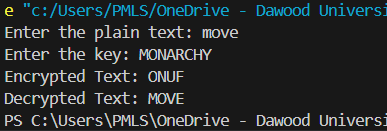
    encrypted\_text = playfair\_encrypt(plain\_text, key)

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

    decrypted\_text = playfair\_decrypt(encrypted\_text, key)

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

**OUTPUT**

****

## **b) Autokey Cypher**

def preprocess\_text(text):

    return ''.join([c.upper() for c in text if c.isalpha()])

def autokey\_encrypt(plain\_text, key):

    plain\_text = preprocess\_text(plain\_text)

    key\_stream = (key + plain\_text)[:len(plain\_text)]

    cipher\_text = ""

    for p, k in zip(plain\_text, key\_stream):

        encrypted\_char = chr((ord(p) - ord('A') + ord(k) - ord('A')) % 26 + ord('A'))

        cipher\_text += encrypted\_char

    return cipher\_text

def autokey\_decrypt(cipher\_text, key):

    cipher\_text = preprocess\_text(cipher\_text)

    key\_stream = key

    plain\_text = ""

    for i in range(len(cipher\_text)):

        decrypted\_char = chr((ord(cipher\_text[i]) - ord('A') - (ord(key\_stream[i]) - ord('A'))) % 26 + ord('A'))

        plain\_text += decrypted\_char

        key\_stream += decrypted\_char

    return plain\_text

plain\_text = input("Enter plain text: ")

key = input("Enter key: ")

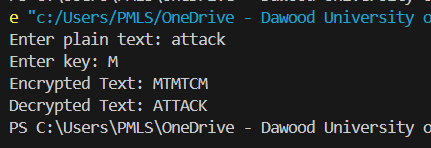
encrypted\_text = autokey\_encrypt(plain\_text, key)

decrypted\_text = autokey\_decrypt(encrypted\_text, key)

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

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

**OUTPUT**

****

## **c) Vigenère Cypher**

def preprocess\_text(text):

    return ''.join([c.upper() for c in text if c.isalpha()])

def vigenere\_encrypt(plain\_text, key):

    plain\_text = preprocess\_text(plain\_text)

    key = (key \* ((len(plain\_text) // len(key)) + 1))[:len(plain\_text)]

    cipher\_text = ''.join([chr((ord(p) - ord('A') + ord(k) - ord('A')) % 26 + ord('A')) for p, k in zip(plain\_text, key)])

    return cipher\_text

def vigenere\_decrypt(cipher\_text, key):

    cipher\_text = preprocess\_text(cipher\_text)

    key = (key \* ((len(cipher\_text) // len(key)) + 1))[:len(cipher\_text)]

    plain\_text = ''.join([chr((ord(c) - ord('A') - (ord(k) - ord('A'))) % 26 + ord('A')) for c, k in zip(cipher\_text, key)])

    return plain\_text

plain\_text = input("Enter plain text: ")

key = input("Enter key: ")

encrypted\_text = vigenere\_encrypt(plain\_text, key)

decrypted\_text = vigenere\_decrypt(encrypted\_text, key)

print(f"Encrypted Text: {encrypted\_text}")

print(f"Decrypted Text: {decrypted\_text}")

**OUTPUT**

**A screenshot of a computer screen

Description automatically generated**