**Cryptography and Network Security  
Lab Assignment - I**

**Program 1: Caesar Cipher**

**Pseudocode/Algorithm**

* **Start**: Display the program details and options menu for encryption, decryption, or exit.
* **Input Choice**: Take the user’s choice for encryption, decryption, or exiting the program.
* **Exit Check**: If the choice is to exit, terminate the program.
* **Input Text and Key**: Prompt the user to enter the plaintext and the key.
* **Encrypt/Decrypt**: Based on the choice, call the caesar function with the appropriate mode ('e' for encryption, 'd' for decryption).
* **Display Result**: Print the result of the encryption or decryption.
* **Repeat or End**: Return to step 1 unless the user chooses to exit.

**Code**

def caesar(text, key, chk):

    result=""

    for i in text:

        if i==' ':

            result+=' '

            continue

        if chk == 'e':

            result+=chr((ord(i)+key-97)%26+97)

        elif chk=='d':

            result+=chr((ord(i)-key-97)%26+97)

    return result

print("Siddhanth Monnappa\t22BCE3061")

while(True):

    print("--CAESAR CIPHER--")

    print("1) Encrypt Text")

    print("2) Decrypt Text")

    print("3) Exit")

    ch=input("Enter your choice: ")

    if ch=='3':

        break

    text=input("Enter PlainText: ")

    text=text.lower()

    key=int(input("Enter Key: "))

    if ch=='1':

        print(f"Caesar Cipher: {caesar(text, key, 'e')}")

    elif ch=='2':

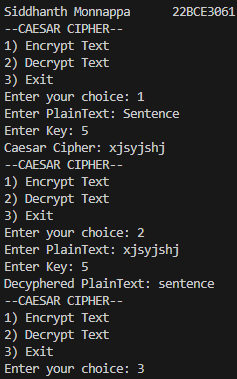
        print(f"Decyphered PlainText: {caesar(text, key, 'd')}")

**Code Screenshot**

**A screen shot of a computer program

Description automatically generated**

**Output**



**Program 2: Playfair Cipher**

**Pseudocode/Algorithm**

* **Start**: Display the program details and options menu for encryption, decryption, or exit.
* **Input Choice**: Take the user’s choice for encryption, decryption, or exit.
* **Exit Check**: If the choice is to exit, terminate the program.
* **Generate Key Matrix**: Take the user-input key, preprocess it, and generate a 5x5 key matrix by combining the key and the alphabet, excluding duplicates and replacing 'j' with 'i'.
* **Preprocess Text**: Preprocess the input text by converting it to lowercase, removing spaces, replacing 'j' with 'i', and appending 'x' if the text length is odd. Split the text into digraphs.
* **Encrypt/Decrypt**: Based on the choice, process each digraph using the encrypt or decrypt function, applying Playfair rules for rows, columns, and rectangles.
* **Display Result**: Print the resulting ciphertext (for encryption) or plaintext (for decryption) and return to step 1 unless the user chooses to exit.

**Code**

def encrypt(key\_matrix, d):

        a, b = d

        row\_a, col\_a=divmod(key\_matrix.index(a), 5)

        row\_b, col\_b=divmod(key\_matrix.index(b), 5)

        if row\_a==row\_b:

            col\_a=(col\_a+1)%5

            col\_b=(col\_b+1)%5

        elif col\_a==col\_b:

            row\_a=(row\_a+1)%5

            row\_b=(row\_b+1)%5

        else:

            col\_a, col\_b=col\_b, col\_a

        return key\_matrix[row\_a\*5+col\_a]+key\_matrix[row\_b\*5+col\_b]

def decrypt(key\_matrix, d):

        a, b = d

        row\_a, col\_a=divmod(key\_matrix.index(a), 5)

        row\_b, col\_b=divmod(key\_matrix.index(b), 5)

        if row\_a==row\_b:

            col\_a=(col\_a-1)%5

            col\_b=(col\_b-1)%5

        elif col\_a==col\_b:

            row\_a=(row\_a-1)%5

            row\_b=(row\_b-1)%5

        else:

            col\_a, col\_b=col\_b, col\_a

        return key\_matrix[row\_a\*5+col\_a]+key\_matrix[row\_b\*5+col\_b]

print("Siddhanth Monnappa\t22BCE3061\n")

while(True):

    print("--PLAYFAIR ENCRYPTION--")

    print("1) Encrypt Text")

    print("2) Decrypt Text")

    print("3) Exit")

    ch=int(input("Enter Choice: "))

    if ch==3:

        break

    text=input("Enter Text: ")

    key=input("Enter Key: ")

    alpha='abcdefghiklmnopqrstuvwxyz'

    key=key.lower().replace(' ', '').replace('j', 'i')

    key\_matrix=''

    for alp in key+alpha:

        if alp not in key\_matrix:

            key\_matrix+=alp

    text=text.lower().replace(' ', '').replace('j', 'i')

    if len(text)%2==1:

        text+='x'

    digraph=[text[i:i+2] for i in range(0, len(text), 2)]

    result=''

    if(ch==1):

        for d in digraph:

            result+=encrypt(key\_matrix,d)

        print("Ciphertext:", result)

    elif(ch==2):

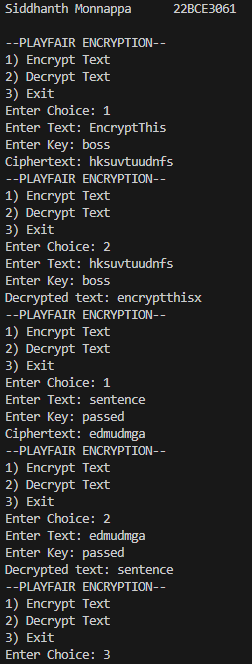
        for d in digraph:

            result+=decrypt(key\_matrix,d)

        print("Decrypted text:", result)

**Code Screenshot Output**

**A screen shot of a computer

Description automatically generated** ****

**Program 3: Hill Cipher**

**Pseudocode/Algorithm**

* **Start:** Display the program details and options menu for encryption, decryption, or exit.
* **Input Choice:** Take the user’s choice for encryption, decryption, or exit.
* **Exit Check:** If the choice is to exit, terminate the program.
* **Input Key:** Prompt the user for an alphabetic key of at least 4 characters.
* **Input Text:** Prompt the user to encrypt or decrypt.
* **Format Text:** Preprocess the text by converting it to uppercase, removing spaces, and appending "X" if the length is odd.
* **Key Matrix:** Generate a 2x2 key matrix from the first 4 characters of the key, converting letters to numeric values.
* **Encrypt/Decrypt Matrix:** If decrypting, calculate the inverse of the key matrix. If the inverse doesn’t exist, terminate with an error.
* **Text to Numbers:** Convert the text into numeric values using their alphabetic positions.
* **Matrix Multiplication:** Multiply the numeric text with the key matrix (for encryption) or its inverse (for decryption), modulo 26.
* **Numbers to Text:** Convert the resulting numbers back to text using their alphabetic positions.
* **Display Result:** Print the resulting ciphertext (for encryption) or plaintext (for decryption) and return to step 1 unless the user chooses to exit.

**Code**

def format\_text(text):

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

    if len(text)%2!=0:

        text+="X"

    return text

def numeric\_value(text):

    numbers=[]

    for char in text:

        if char.isalpha():

            numbers.append(ord(char)-ord("A"))

    return numbers

def key\_to\_matrix(key):

    key=key.upper().replace(" ", "")

    if len(key) < 4:

        print("Key Must be atleast 4 characters long")

        exit()

    key=key[:4]

    key\_numbers=numeric\_value(key)

    return [[key\_numbers[0], key\_numbers[1]],[key\_numbers[2], key\_numbers[3]],]

def num\_to\_text(num):

    text=""

    for n in num:

        text+=chr(n+ord("A"))

    return text

def mod\_inverse(a, m):

    for i in range(1, m):

        if (a\*i)%m==1:

            return i

    return None

def matrix\_multiply(matrix1, matrix2):

    result=[]

    for i in range(0, len(matrix1), 2):

        row\_result=[]

        for j in range(2):

            result\_val=sum(matrix1[i+x]\*matrix2[x][j] for x in range(2))%26

            row\_result.append(result\_val)

        result.extend(row\_result)

    return result

def inverse\_matrix(key\_matrix):

    det=(key\_matrix[0][0]\*key\_matrix[1][1]-key\_matrix[0][1]\*key\_matrix[1][0])%26

    inv\_det=mod\_inverse(det, 26)

    if inv\_det is None:

        return None

    adj\_mat=[[key\_matrix[1][1], -key\_matrix[0][1]],[-key\_matrix[1][0], key\_matrix[0][0]],]

    for i in range(2):

        for j in range(2):

            adj\_mat[i][j]=adj\_mat[i][j]%26

    inv\_mat=[[(inv\_det\*adj\_mat[i][j])%26 for j in range(2)]for i in range(2)]

    return inv\_mat

def encrypt(text, key):

    text=format\_text(text)

    key\_matrix=key\_to\_matrix(key)

    num=numeric\_value(text)

    result\_numbers=matrix\_multiply(num, key\_matrix)

    result\_text=num\_to\_text(result\_numbers)

    return result\_text

def decrypt(text, key):

    text=format\_text(text)

    key\_matrix=key\_to\_matrix(key)

    key\_matrix=inverse\_matrix(key\_matrix)

    if key\_matrix is None:

        return "Key matrix is not invertible"

    num=numeric\_value(text)

    result\_numbers=matrix\_multiply(num, key\_matrix)

    result\_text=num\_to\_text(result\_numbers)

    return result\_text

print("Siddhanth Monnappa\t22BCE3061\n")

while(True):

    print("--HILL CIPHER--")

    print("1) Encrypt Text")

    print("2) Decrypt Text")

    print("3) Exit")

    ch=int(input("Enter Choice: "))

    if ch==3:

        break

    key = input("Enter an alphabetic key (min 4 characters): ")

    text = input("Enter the text: ")

    if (ch==1):

        print(f"Ciphertext: {encrypt(text, key)}")

    elif (ch==2):

        print(f"Decrypted Text: {decrypt(text, key)}")

**Code Screenshot**

**A screen shot of a computer screen

Description automatically generatedA screen shot of a computer screen

Description automatically generated**

**Output**

**A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated**

**Program 4: Vigenère Cipher**

**Pseudocode/Algorithm**

* **Start**: Display the program details and options menu for encryption, decryption, or exit.
* **Input Choice**: Take the user’s choice for encryption, decryption, or exit.
* **Exit Check**: If the choice is to exit, terminate the program.
* **Input Text and Key**: Prompt the user to input the text and the key for encryption or decryption.
* **Generate Key**: Extend the key to match the length of the text, keeping non-alphabetic characters in place.
* **Iterate Over Text**: Loop through each character in the text.
* **Check Alphabetic Characters**: If the character is alphabetic, calculate its new position based on the key and mode (encrypt or decrypt).
* **Handle Non-Alphabetic Characters**: If the character is not alphabetic, retain it in the result.
* **Build Result**: Combine the processed characters to form the encrypted or decrypted text.
* **Display Result**: Print the resulting ciphertext or plaintext and return to step 1 unless the user chooses to exit.

**Code**

def generate\_key(text, key):

    key=key.upper()

    ext\_key=""

    key\_index=0

    for char in text:

        if char.isalpha():

            ext\_key+=key[key\_index%len(key)]

            key\_index+=1

        else:

            ext\_key+=char

    return ext\_key

def vigenere(text, key, chk):

    result=""

    key=generate\_key(text, key)

    for i, char in enumerate(text):

        if char.isalpha():

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

            key\_shift=ord(key[i].upper())-ord('A')

            if chk=="e":

                result+=chr((ord(char)-base+key\_shift)%26+base)

            else:

                result+=chr((ord(char)-base-key\_shift)%26+base)

        else:

            result+=char

    return result

print("Siddhanth Monnappa\t22BCE3061\n")

while(True):

    print("--VIGENERE ENCRYPTION--")

    print("1) Encrypt Text")

    print("2) Decrypt Text")

    print("3) Exit")

    ch=int(input("Enter Choice: "))

    if ch==3:

        break

    text = input("Enter the text: ")

    key = input("Enter the key: ")

    if ch==1:

        print(f"Encrypted: {vigenere(text,key,"e")}")

    elif ch==2:

        print(f"Decrypted: {vigenere(text,key,"d")}")

**Code Screenshot Output**

**A screen shot of a computer screen

Description automatically generated** **A screen shot of a computer

Description automatically generated**