Caesar Cipher:

def encrypt(text,s):

    result = ""

    for char in text:

        if char.isnumeric():

            base = ord('0')

            result += chr((ord(char)-base + s)%10+base)

        else:

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

            result += chr((ord(char)-base + s)%26+base)

    return result

def decrypt(text,s):

    result = ""

    for char in text:

        if char.isnumeric():

            base = ord('0')

            result += chr((ord(char)-base - s)%10+base)

        else:

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

            result += chr((ord(char)-base - s)%26+base)

    return result

text = "BtechZ9"

s = 4

cipherText=encrypt(text,s)

print (f"Text:{text}\nShift:{s}\nCipher:{cipherText}\nDecrypted:{decrypt(cipherText,s)}")

PlayFair cipher:

def create\_matrix(key, list1=['A','B','C','D','E','F','G','H','I','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']):

    temp,matrix = [],[]

    for i in key:

        if i not in temp:

            temp.append(i)

    for i in list1:

        if i not in temp:

            temp.append(i)

    while temp != []:

        matrix.append(temp[:5])

        temp = temp[5:]

        # for i in range(5): temp.pop(0)

    return matrix

def addBuffer(message):

    for index in range(0, len(message)-1, 2):

        l1,l2 =message[index], message[index + 1]

        if l1 == l2:

            message = message[:index + 1] + "X" + message[index + 1:]

    if len(message)%2!=0:

        message+='X'

    return message

def indexOf(letter, matrix):

    for i in range(5):

        try:

            index = matrix[i].index(letter)

            return (i, index)

        except:

            continue

def playfair(key, message, encrypt=True):

    inc = 1 if encrypt else -1

    matrix = create\_matrix(key)

    message = message.replace(' ', '')

    message = addBuffer(message)

    cipher\_text = ''

    for (l1, l2) in zip(message[0::2], message[1::2]):

        row1, col1 = indexOf(l1, matrix)

        row2, col2 = indexOf(l2, matrix)

        if row1 == row2:

            cipher\_text += matrix[row1][(col1 + inc) % 5] + matrix[row2][(col2 + inc) % 5]

        elif col1 == col2:

            cipher\_text += matrix[(row1 + inc) % 5][col1] + matrix[(row2 + inc) % 5][col2]

        else:

            cipher\_text += matrix[row1][col2] + matrix[row2][col1]

    return cipher\_text

plainText = input("Enter the Plain text: ").upper()

key = input("Enter Key: ").upper().replace('J','I')

cipherText = playfair(key, plainText)

print(f'Encrypted:{cipherText}\nDecrypted:{playfair(key, cipherText, encrypt=False)}')

Vigenere Cipher:

plain\_txt = input("Enter plaintext : ").upper()

key = input("Enter Key : ").upper()

padd\_key = key

if len(plain\_txt) > len(key):

    for i in range(len(plain\_txt)-len(key)):

        padd\_key += key[i%len(key)]

encrpyted,decrpyted = "",""

for i in range(len(plain\_txt)):

    encrpyted += chr(((ord(plain\_txt[i]) + ord(padd\_key[i])) % 26) + 65)

for i in range(len(plain\_txt)):

    decrpyted += chr(((ord(encrpyted[i]) - ord(padd\_key[i])) % 26) + 65)

print(f"\nPlain Text : {plain\_txt}\nKey : {key}\nPadded Key : {padd\_key}")

print(f"\nafter encrption, cipher text : {encrpyted}\nafter decyprtion, decrypted text : {decrpyted}")

Vernam cipher:

plain = input("Enter text: ").upper()

key = input("Enter key: ").upper()

padded\_key = key

if len(key)<len(plain):

    for i in range(len(plain)-len(key)):

        padded\_key+=key[i%len(key)]

key = padded\_key

encrypt, decrypt = "",""

for p,q in zip(plain,key):

    encrypt += chr(ord(p)^ord(q))

for p,q in zip(encrypt,key):

    decrypt += chr(ord(p)^ord(q))

print("Encrypted: ",encrypt)

print("Decrypted: ",decrypt)

Columnar Cipher:

import math

def find\_rank(key):

    rank = 0

    for i in sorted(key):

        key = key.replace(i, str(rank), 1)

        rank += 1

    key = [int(i) for i in key]

    return key

def encrypt(pt, key):

    pt = pt.upper().replace(" ", "")  # Capitalize and remove spaces

    key = key.upper().replace(" ", "")  # Capitalize and remove spaces

    cols = len(key)

    rows = math.ceil(len(pt) / cols)

    key\_rank = find\_rank(key)

    print(key\_rank)

    pt += "".join(["X"] \* (rows \* cols - len(pt)))  # Pad the plaintext with 'X'

    matrix = [list(pt[i: i+cols]) for i in range(0, len(pt), cols)]

    for i in range(rows):

        print(matrix[i])

    ciphertext = ""

    for ind in range(len(key\_rank)):

        col = key\_rank.index(ind)

        for i in range(rows):

            ciphertext += matrix[i][col]

    return ciphertext

def decrypt(cip, key):

    cip = cip.upper().replace(" ", "")  # Capitalize and remove spaces

    key = key.upper().replace(" ", "")  # Capitalize and remove spaces

    cols = len(key)

    rows = math.ceil(len(cip) / cols)

    key\_rank = find\_rank(key)

    cip += "".join(["X"] \* (rows \* cols - len(cip)))  # Pad the ciphertext with 'X' if necessary

    cip\_mat = [[0 for col in range(cols)] for row in range(rows)]

    count = 0

    for ind in range(len(key\_rank)):

        col = key\_rank.index(ind)

        for row in range(rows):

            cip\_mat[row][col] = cip[count]

            count += 1

    result = ""

    for row in cip\_mat:

        result += ''.join(row)

    return result.rstrip('X')  # Remove padding characters

pt, key = 'were89ik', 'io'

ciphertext = encrypt(pt, key)

decrypted = decrypt(ciphertext, key)

print(f"After encryption, Cipher Text: {ciphertext}\nAfter decryption, Plain Text: {decrypted}")

doubleCipher = encrypt(ciphertext, key)

doubleDecrypted = decrypt(decrypt(doubleCipher, key), key)

print(f"After double encryption, Cipher Text: {doubleCipher}\nAfter double decryption, Plain Text: {doubleDecrypted}")

RSA and Digital Signature:

#RSA encryption/Decryption and Digital Signature

import random, hashlib

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def generate\_keys():

    p = int(input("Enter p (greater than 100): ")) #113

    q = int(input("Enter q (greater than 100): ")) #101

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randint(1, phi)

    while gcd(e, phi) != 1:

        e = random.randint(1, phi)

    d = pow(e, -1, phi)

    return (e, n), (d, n)

def encrypt(message, public\_key):

    e, n = public\_key

    encrypted\_message = [pow(ord(char), e, n) for char in message]

    return encrypted\_message

def decrypt(encrypted\_message, private\_key):

    d, n = private\_key

    decrypted\_message = [chr(pow(char, d, n)) for char in encrypted\_message]

    return ''.join(decrypted\_message)

message = "Hello, World!"

public\_key, private\_key = generate\_keys()

encrypted\_message = encrypt(message, public\_key)

decrypted\_message = decrypt(encrypted\_message, private\_key)

print("Public key: ",public\_key)

print("Private key: ",private\_key)

print(f"Original message:{message}\nEncryptedMeessage:{encrypted\_message}\nDecryptedMsg:{decrypted\_message}")

#Digital Signature part

hash\_value = hashlib.md5(message.encode()).hexdigest()

print("Hash value at sender end: ", hash\_value)

signature = encrypt(hash\_value, public\_key)

print("Digital signature: ", signature)

# receiver side

hash\_value\_check = decrypt(signature, private\_key)

print("Hash value checked at receiver end: ", hash\_value\_check)

if hash\_value\_check == hash\_value:

    print("Verified")

else:

    print("Not verified")

Diffie-Hellman:

p = 23

g = 7

a = 4

b = 3

if a>=p or b>=p:

    print("a or b is less than p")

else:

    XA = (g\*\*a) % p

    XB = (g\*\*b) % p

    AK = (XB\*\*a) % p

    BK = (XA\*\*b) % p

    if AK==BK:

        print(AK,BK,"User verified")

    else:

        print("nahhh")