key = b'bsyu%bsh'

cipher = DES.new(key,DES.MODE\_EAX)

data = "Welcome to cryptoassignment.com!".encode()

nonce = cipher.nonce

ciphertext = cipher.encrypt(data)

print("cipher text:",ciphertext)

cipher = DES.new(key,DES.MODE\_EAX,nonce=nonce)

plaintext = cipher.decrypt(ciphertext)

print("Plain text :",plaintext)

import random

import math

public\_key = None

private\_key = None

n = None

def setkeys():

global public\_key, private\_key, n

prime1 = int(input("Enter prime1 :"))

prime2 = int(input("Enter prime2 :"))

n = prime1 \* prime2

fi = (prime1 - 1) \* (prime2 - 1)

print("n : ", n)

e = 2

while True:

if math.gcd(e, fi) == 1:

break

e += 1

public\_key = e

d = 3

while True:

if (d \* e) % fi == 1:

break

d += 1

private\_key = d

def encrypt(message):

global public\_key, n

e = public\_key

encrypted\_text = 1

while e > 0:

encrypted\_text \*= message

encrypted\_text %= n

e -= 1

return encrypted\_text

def decrypt(encrypted\_text):

global private\_key, n

d = private\_key

decrypted = 1

while d > 0:

decrypted \*= encrypted\_text

decrypted %= n

d -= 1

return decrypted

def encoder(message):

encoded = 0

encoded = encrypt(message)

return encoded

def decoder(encoded):

s = 0

s += decrypt(encoded)

return s

if \_\_name\_\_ == '\_\_main\_\_':

setkeys()

message = 20

coded = encoder(message)

print("Initial message:")

print(message)

print("\n\nThe encoded message(encrypted by public key)\n")

print(coded)

print("\n\nThe decoded message(decrypted by public key)\n")

print(decoder(coded))

# Diffie-Hellman Code

def prime\_checker(p):

  # Checks If the number entered is a Prime Number or not

  if p < 1:

    return -1

  elif p > 1:

    if p == 2:

      return 1

    for i in range(2, p):

      if p % i == 0:

        return -1

      return 1

def primitive\_check(g, p, L):

  # Checks If The Entered Number Is A Primitive Root Or Not

  for i in range(1, p):

    L.append(pow(g, i) % p)

  for i in range(1, p):

    if L.count(i) > 1:

      L.clear()

      return -1

    return 1

l = []

while 1:

  P = int(input("Enter P : "))

  if prime\_checker(P) == -1:

    print("Number Is Not Prime, Please Enter Again!")

    continue

  break

while 1:

  G = int(input(f"Enter The Primitive Root Of {P} : "))

  if primitive\_check(G, P, l) == -1:

    print(f"Number Is Not A Primitive Root Of {P}, Please Try Again!")

    continue

  break

# Private Keys

x1, x2 = int(input("Enter The Private Key Of User 1 : ")), int(

  input("Enter The Private Key Of User 2 : "))

while 1:

  if x1 >= P or x2 >= P:

    print(f"Private Key Of Both The Users Should Be Less Than {P}!")

    continue

  break

# Calculate Public Keys

y1, y2 = pow(G, x1) % P, pow(G, x2) % P

# Generate Secret Keys

k1, k2 = pow(y2, x1) % P, pow(y1, x2) % P

print(f"\nSecret Key For User 1 Is {k1}\nSecret Key For User 2 Is {k2}\n")

if k1 == k2:

  print("Keys Have Been Exchanged Successfully")

else:

  print("Keys Have Not Been Exchanged Successfully")

**import** math

# step 1

p **=** 3

q **=** 7

# step 2

n **=** p**\***q

print("n =", n)

# step 3

phi **=** (p**-**1)**\***(q**-**1)

# step 4

e **=** 2

**while**(e<phi):

**if** (math.gcd(e, phi) **==** 1):

**break**

**else**:

        e **+=** 1

print("e =", e)

# step 5

k **=** 2

d **=** ((k**\***phi)**+**1)**/**e

print("d =", d)

print(f'Public key: {e, n}')

print(f'Private key: {d, n}')

# plain text

msg **=** 11

print(f'Original message:{msg}')

# encryption

C **=** pow(msg, e)

C **=** math.fmod(C, n)

print(f'Encrypted message: {C}')

# decryption

M **=** pow(C, d)

M **=** math.fmod(M, n)

print(f'Decrypted message: {M}')

import hashlib

inputstring ="this is a message sent by a computer user"

output=hashlib.md5(inputstring.encode())

print("Hash of the input string: ")

print(output.hexdigest())