**Introduction to Asymmetric Encryption Algorithm:**

How we can encrypt and be safe from intruders.

Asymmetric-key algorithms work in a similar manner to symmetric-key algorithms, where plaintext is combined with a key, input to an algorithm, and outputs ciphertext. The major difference is the keys used for the encryption and decryption portions are different, thus the asymmetry of the algorithm. The key pair is comprised of a private key and a public key. As the names imply, the public key is made available to everyone, whereas the private key is kept secret.

The two main uses of asymmetric-key algorithms are public-key encryption and digital signatures. Public-key encryption is a method where anyone can send an encrypted message within a trusted network of users. The sender encrypts the message using the receiver’s public key, allowing only the receiver to decrypt the message using his or her own private key. Anyone could intercept the encrypted message, but only the receiver can decrypt it.

Public key private key

| |

Plain text -> encryption algorithm -> cipher text -> decryption algorithm -> plain text

**Algorithm:**

**Alice:**

Select large prime numbers PA and QA

Corrupt nA = pA \* qA

Compute euler function for nA

Pi(nA) = (pA-1)\*(qA-1)

Select public component eA such that 1< eA< pi(nA) and gcd(eA, pi(nA)) =1

Compute private key dA

eA\*dA = 1mod pi(nA)

**Bob:**

Select large prime numbers pB and qB

Compute nB = pB \* qB

Compute euler function for nB

Pi(nB)= (pB-1)\*(qB-1)

Select public component eB such that 1< eB< pi(nB) and gcd (eB, pi(nB)) =1

Compute private key dB

eB\*dB = 1mod pi(nB)

Store public key values in CA (certification authority)

Encryption c= (meB) mod nB

Decryption m= (cdB) mod nB

**Pseudo code of RSA:**

define encryption function:

n = int(p\*q)

f = int((p-1)\*(q-1))

cipher\_text =[]

for ch in msg:

w = ord(ch) - 97

if w is less than 0:

w = 0-pow(w,e)%n

else:

w = pow(w,e,n)

append(w) to cipher\_text

return cipher\_text

define gcd function:

small,large = (x,y) if x<y else (y,x)

while small != 0:

temp = large % small

large = small

small = temp

return large

define calculation of d function:

d is equal to 2

while d is less than f:

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

return d

d += 1

define decryption function:

n = p \* q

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

d = cal\_d(e, f)

plain\_text =[]

for ch in msg:

if ch is less than 0:

m=0-pow(ch,d)%n

else:

m=pow(ch,d,n)

append(m) to plain\_text

message=''

for i in plain\_text:

message+=chr(i+97)

return message

start main function:

Take input of p

Take input of q

Take input of e

Take input of message to be processed

Start encryption timer

cipher\_text = encryption(p, q, e,msg);

print Encrypted text

end & print Encryption Response time

Start decryption timer

plain\_text= decryption(p, q, e,cipher\_text)

print Decrypted text

end & print Decryption Response time

**Coding:**

import time

def encryption(p: int,q: int,e: int, msg: str):

n = int(p\*q) #Calculate nB=pB\*qB

f = int((p-1)\*(q-1)) #Calculate pi(nB)=(pB-1)(qB-1)

cipher\_text =[]

for ch in msg:

w = ord(ch) - 97 #-97 is used as the algorithm doesnt follow ascii and decrypts starting from a(a=1)

if w<0:

w = 0-pow(w,e)%n #for caps and special characters shouldnt decrypt negative values

else:

w = pow(w,e,n)

cipher\_text.append(w)

return cipher\_text

def gcd(x: int, y: int): #defining gcd function to be used

small,large = (x,y) if x<y else (y,x)

while small != 0:

temp = large % small

large = small

small = temp

return large

def cal\_d(e: int, f: int): #Calculate dB -> eB\*dB = 1(mod pi) where 1 < d < pi

d = 2

while d < f:

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

return d

d += 1

def decryption(p: int,q: int,e: int, msg=[]):

n = p \* q #Calculate nB=pB\*qB

f = (p-1)\*(q-1) #Calculate pi(nB)=(pB-1)(qB-1)

d = cal\_d(e, f) #Calculate dB using function defined

plain\_text =[]

for ch in msg:

if ch < 0:

m=0-pow(ch,d)%n

else:

m=pow(ch,d,n)

plain\_text.append(m) #Writing Decrypted text

message=''

for i in plain\_text:

message+=chr(i+97)

return message

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

p = int(input("Enter p : ")) #Get input of p,q,e

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

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

msg = input("Enter Message : ")

start1 = time.time()

cipher\_text = encryption(p, q, e,msg);

print("Encrypted text :",cipher\_text)

end1= time.time()

print(f"Encryption Response time : {end1-start1}sec") #to get response times,subtract start and end times

start2 = time.time()

plain\_text= decryption(p, q, e,cipher\_text)

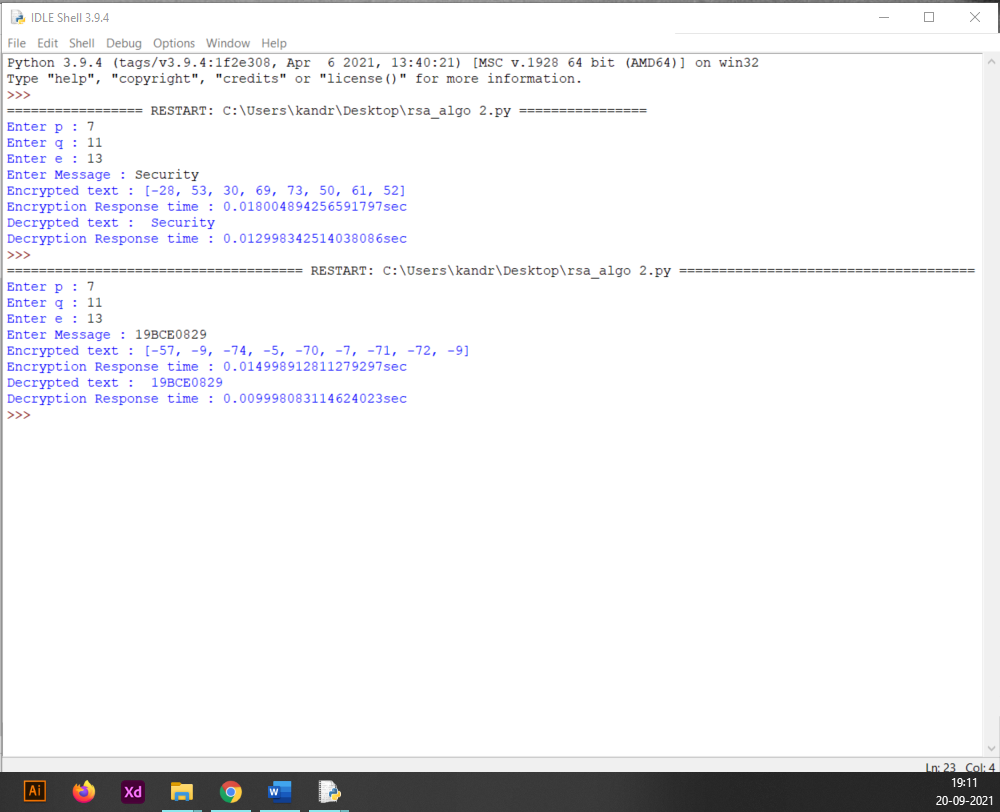
print("Decrypted text : ", plain\_text)

end2= time.time()

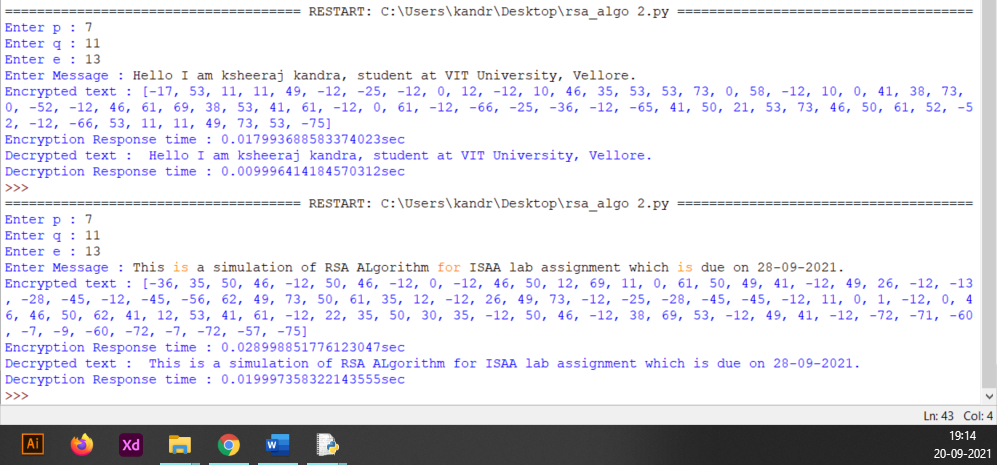
print(f"Decryption Response time : {end2-start2}sec")

**Results:**

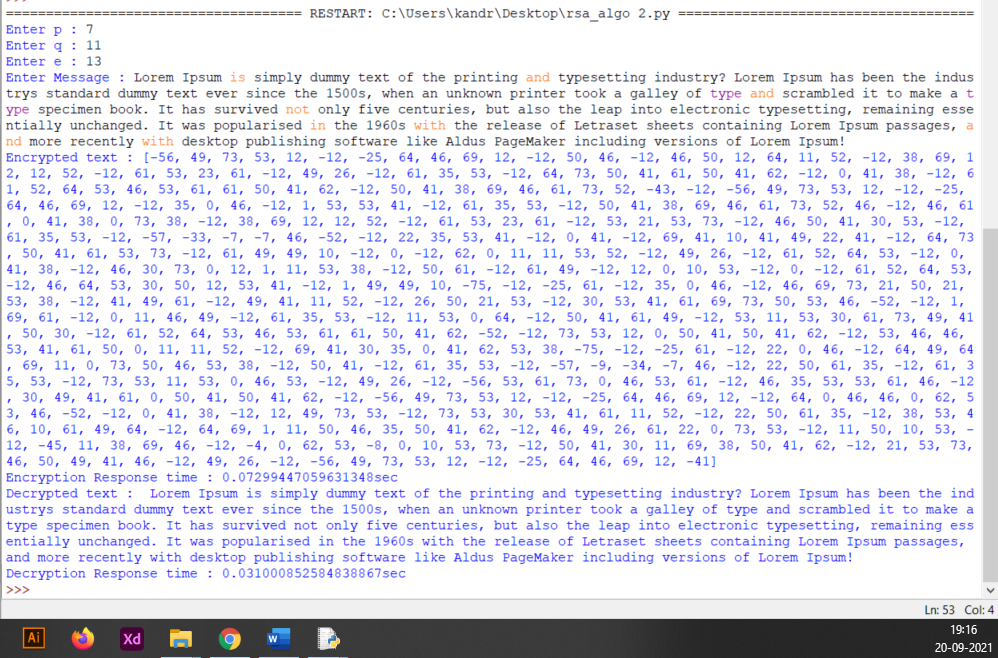
**Word:**

****

**Sentence:**

****

**Paragraph:**

****

**Comparative Chart:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Input** | **Encryption Response time** | **Decryption Response time** |
| 1. | Word | 0.018004894256591797 secs | 0.012998342514038086 secs |
| 2. | Sentence | 0.028998851776123047 secs | 0.019997358322143555 secs |
| 3. | Paragraph | 0.07299447059631348 secs | 0.031000852584838867 secs |