**Name: Gaurav Kumar Singh**

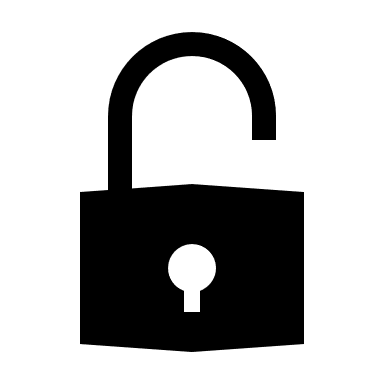
**Registration Number: 19BCE2119**

**Course: Information Security Analysis and Audit (ISAA)**

**RSA Encryption**

**INTRODUCTION ABOUT ASYMMETRIC ENCRYPTION ALGORITHM**

In Asymmetric cryptography algorithms the encryption key and decryption key are different, unlike Symmetric cryptography algorithms, where a single key is used to both encrypt and decrypt the messages. The two keys involved in asymmetric encryption algorithms are usually referred to as, public-key and private-key. Suppose a user ‘A’ wants to send message to another user ‘B’; both the users are assigned their own different sets of public and private keys. Public keys, as the name suggests are made public and everyone can see the public key. Private key however is tied to the user concerned and should not be made public. Since ‘A’ wants to send the message to ‘B’, ‘A’ encrypts the message using the public key of ‘B’ and sends the encrypted message to ‘B’. When ‘B’ receives the encrypted message, ‘B’ can use its private key which only ‘B’ knows about and decrypts the message using its private key. Since, no-one other than ‘B’ knows the private key of ‘B’ ideally, unless, the ‘B’s system is compromised, no one can read or decrypt the message except ‘B’.

Sender🡪Message🡪 🡪Encrypted Message🡪 🡪Decrypted Message🡪Recipient



Public Key Private Key

Asymmetric cryptographic algorithms are usually also slower than symmetric ones and require more computational power but are more secure.

Examples of asymmetric encryption algorithms are RSA, Diffie-Hellman, ECC, DSA, SHA-256 etc.

**PSEUDOCODE:**

int p = RandomPrime(a), q=RandomPrime(a) //a is the bit size of the prime number to be generated.

int n = p\*q;

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

int e = FindRandomCoprime(phi);

int d = ModuloInverse(e,phi);

public\_key = {e,n};

private\_key = {d,n};

message = input();

message\_arr = [];

for chars in message:

message\_arr.append(ascii(chars));

encrypted\_arr=[]

//ENCRYPTION

for m in message\_array:

encrypted\_arr.append((m^e)%n);

//DECRYPTION

decrypted\_arr=[]

for m in encrypted\_arr:

decrypted\_arr.append((m^d)%n);

decrypted\_message=””;

for m in decrypted\_arr:

decrypted\_message += AsciiToChar(m);

print (decrypted\_message);

**CODE (PYTHON):**

import numpy as np

import math

from Crypto.Util import number as crypt

import time

from tabulate import tabulate

# print(crypt.getPrime(3));

def egcd(a, b):

if a == 0:

return (b, 0, 1)

g, y, x = egcd(b%a,a)

return (g, x - (b//a) \* y, y)

def modinv(a, m):

g, x, y = egcd(a, m)

if g != 1:

raise Exception('No modular inverse')

return x%m

x=crypt.getPrime(9);

y=crypt.getPrime(9);

print("Prime 1 (x): ",x,"\tPrime 2 (y): ",y,"\n")

n=x\*y

print("x\*y= ",n)

totient =(x-1)\*(y-1)

i=3

while(math.gcd(i,totient)!=1):

i+=1

e=i

print("e=",e," Phi=",totient,"\n")

d=modinv(e,totient)

#print("d =",d)

print("Public Key: ",n,",",e)

print("Private Key: ",n,",",d)

public\_key=[n,e]

# print((e\*d)%totient)

for q in range(3):

P=str(input("Enter the word/sentence/paragraph to be encrypted:"))

start\_time = time.time()

Original=[]

print("Original")

for character in P:

character=ord(character)

Original.append(character)

print(character,end=" ")

print("\n")

#ENCRYPTION

print(n,e,d)

Encrypted=[]

print("Encrypted")

for a in Original:

temp=(a\*\*e)%n

Encrypted.append(temp)

print(temp,end="")

print("\n")

encrypt\_time=time.time()

print(encrypt\_time-start\_time)

#DECRYPTION

Decrypted=[]

i=0

print("Decrypted")

for a in Encrypted:

temp=(a\*\*d)%n

Decrypted.append(temp)

# print(temp,end=" ")

i+=1

for a in Decrypted:

print(chr(a),end="")

decrypt\_time=time.time()

print(decrypt\_time-encrypt\_time)

if(q==0):

words=[1,P,encrypt\_time-start\_time,decrypt\_time-encrypt\_time]

if(q==1):

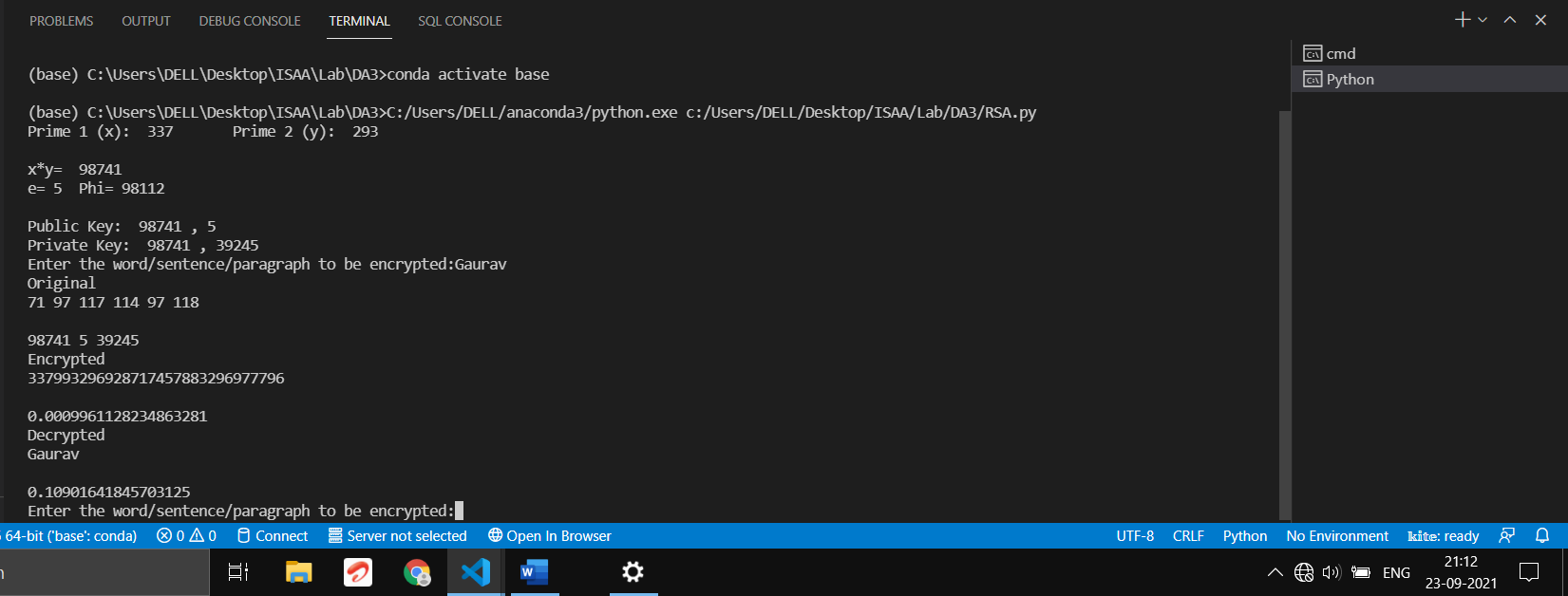
sentence=[2,P,encrypt\_time-start\_time,decrypt\_time-encrypt\_time]

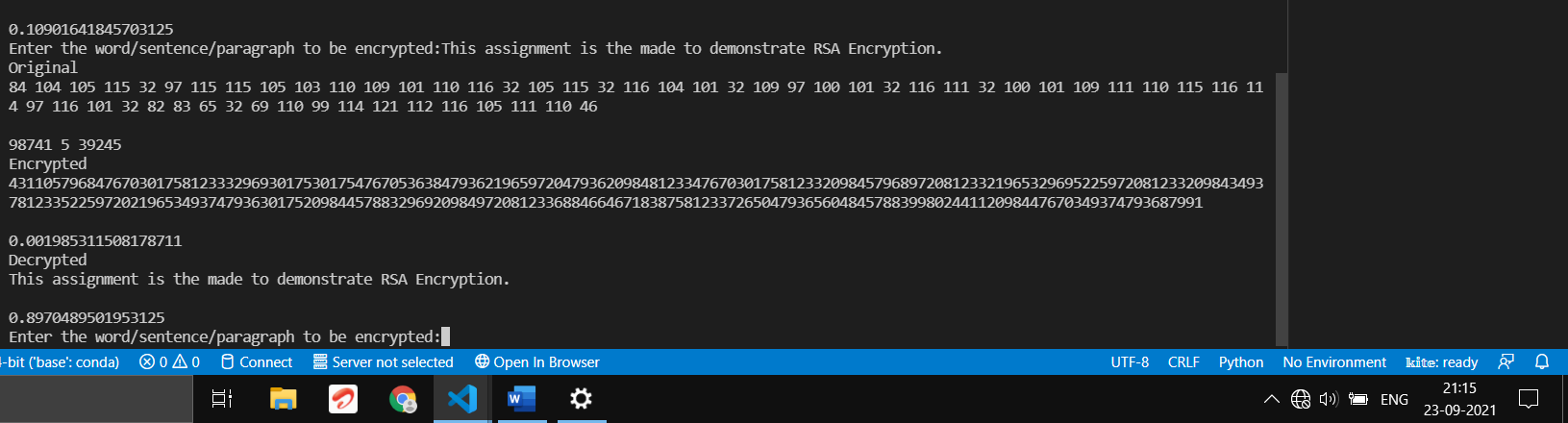
if(q==2):

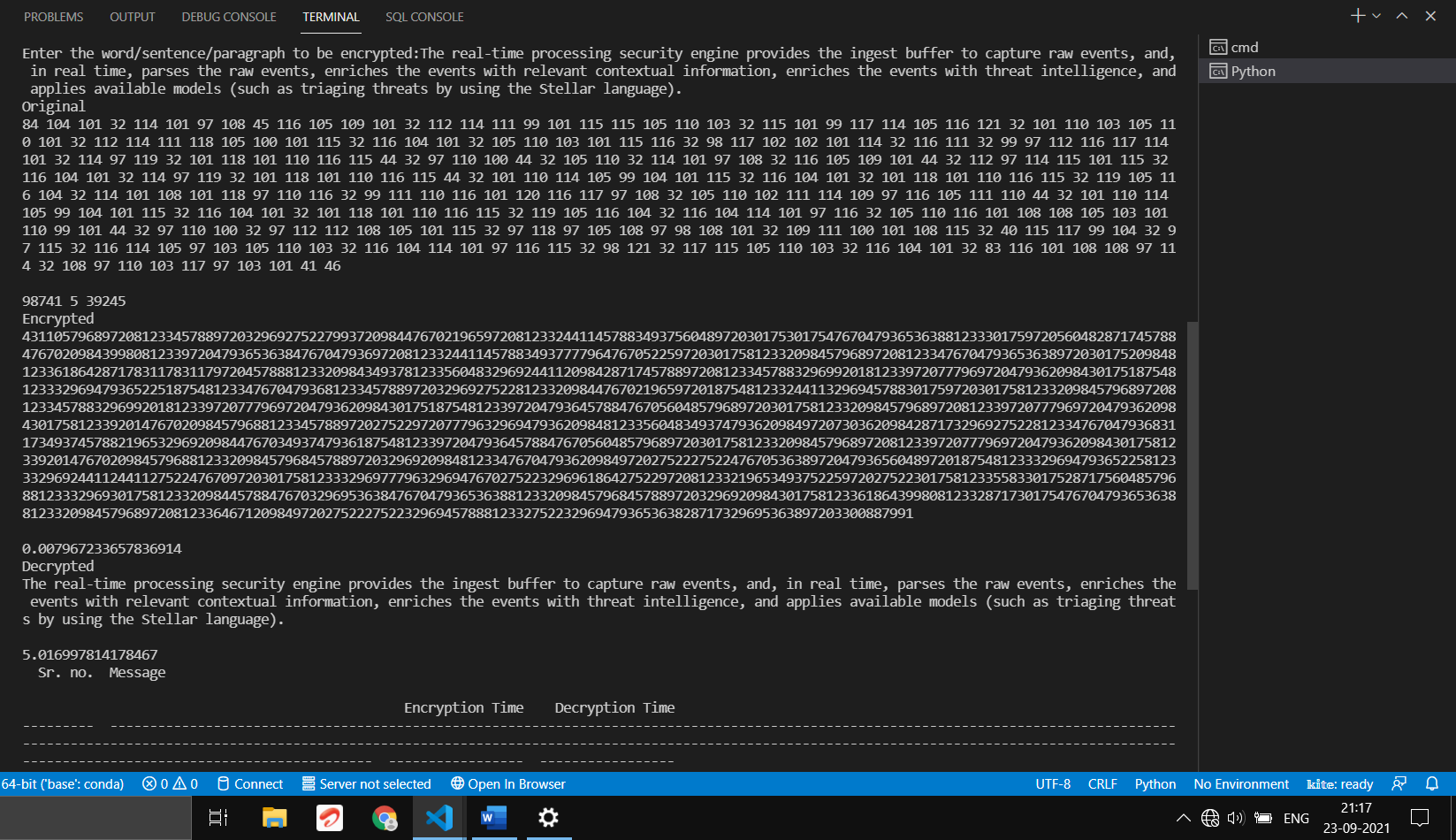
paragraph=[3,P,encrypt\_time-start\_time,decrypt\_time-encrypt\_time]

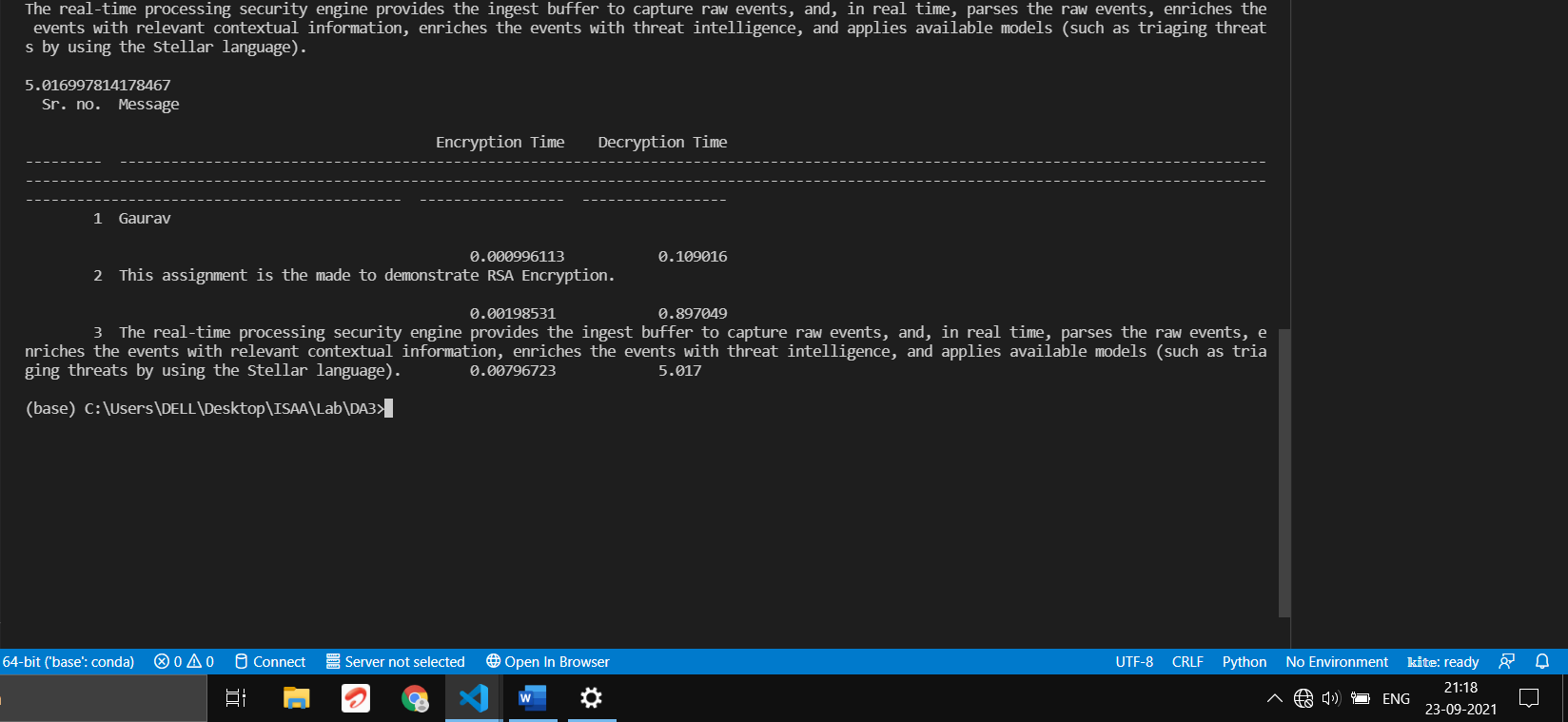
print(tabulate([words,sentence,paragraph], headers=["Sr. no.","Message", "Encryption Time","Decryption Time"]))

**RESULTS:**









**COMPARITIVE CHART:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Message** | **Encryption Time**  **(In seconds)** | **Decryption Time**  **(In second)** |
| **1** | Gaurav | 0.000996113 | 0.109016 |
| **2** | This assignment is the made to demonstrate RSA Encryption. | 0.00198531 | 0.897049 |
| **3** | The real-time processing security engine provides the ingest buffer to capture raw events, and, in real time, parses the raw events, enriches the events with relevant contextual information, enriches the events with threat intelligence, and applies available models (such as triaging threats by using the Stellar language). | 0.00796723 | 5.017 |