RSA ALGORITHM

Y. Sree Santhosh

RA2011030010214

CSE- N1

RSA Algorithm:-

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private**

Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

RSA encryption algorithm is a type of public-key encryption algorithm. To better understand RSA, lets first understand what is public-key encryption algorithm.

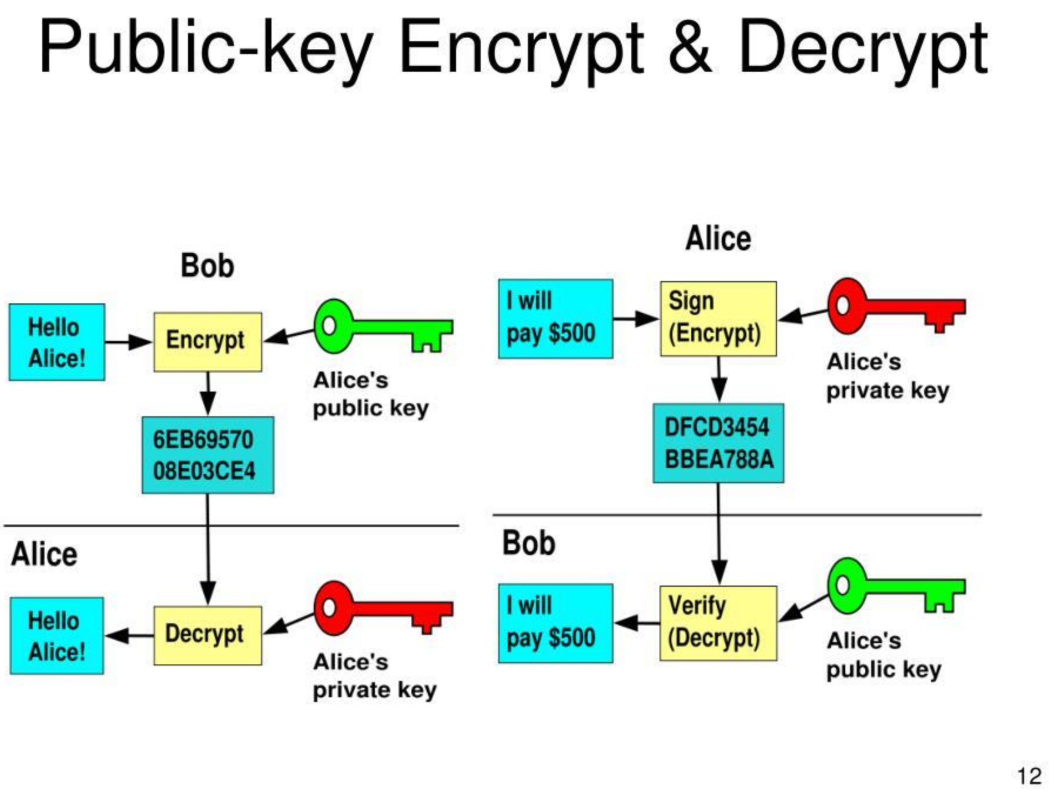
Public key encryption algorithm:

Public Key encryption algorithm is also called the Asymmetric algorithm. Asymmetric algorithms are those algorithms in which sender and receiver use different keys for encryption and decryption. Each sender is assigned a pair of keys:

- **Public key**
- **Private key**

The **Public key** is used for encryption, and the **Private Key** is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and it is known only to the user who owns the key. It means that everybody can send a message to the user using user's public key. But only the user can decrypt the message using his private key.

The Public key algorithm operates in the following manner:



- The data to be sent is encrypted by sender **A** using the public key of the intended receiver
- B decrypts the received ciphertext using its private key, which is known only to B. B replies to A encrypting its message using A's public key.
- A decrypts the received ciphertext using its private key, which is known only to him.

RSA encryption:

RSA is the most common public-key algorithm, named after its inventors **Rivest, Shamir, and Adelman (RSA).**

RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and **q**.
- Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
- Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q - 1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), **gcd (e,d(n)) =1**
- If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C=m^e \bmod n$
  Here**, m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  $D_e \bmod\{(p-1) \times (q-1)\} = 1$
  Or
  $D_e \bmod \varphi(n) = 1$
- The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

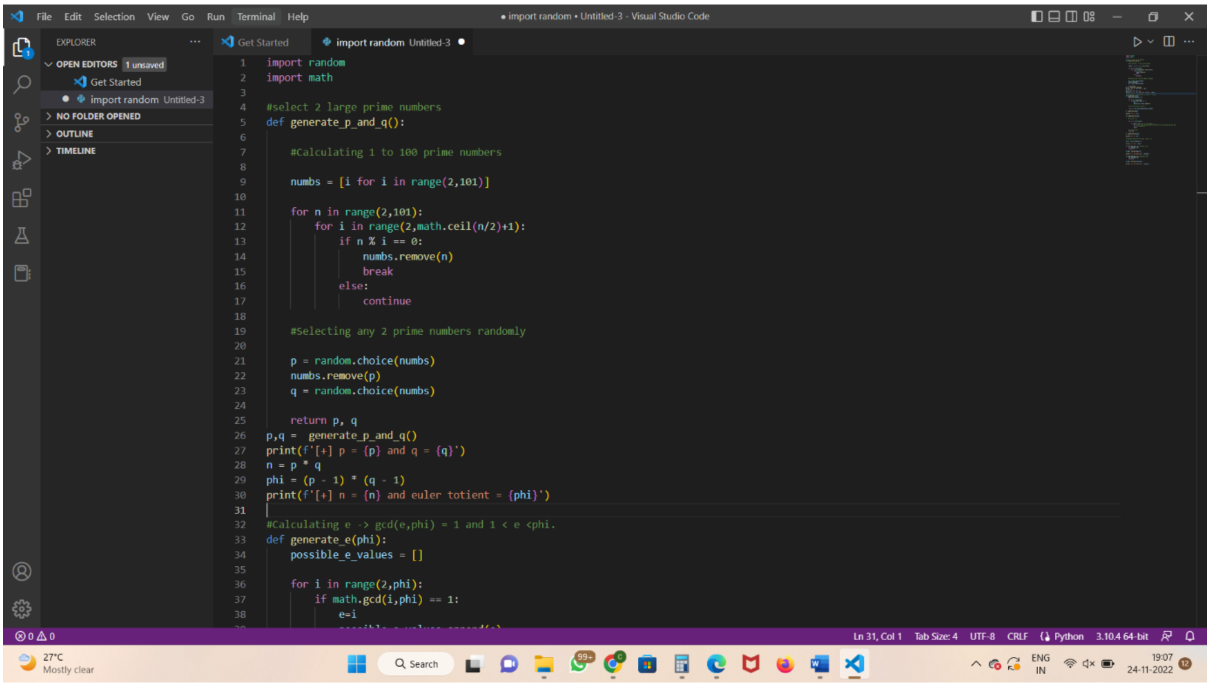
Implementation of RSA algorithm in python:

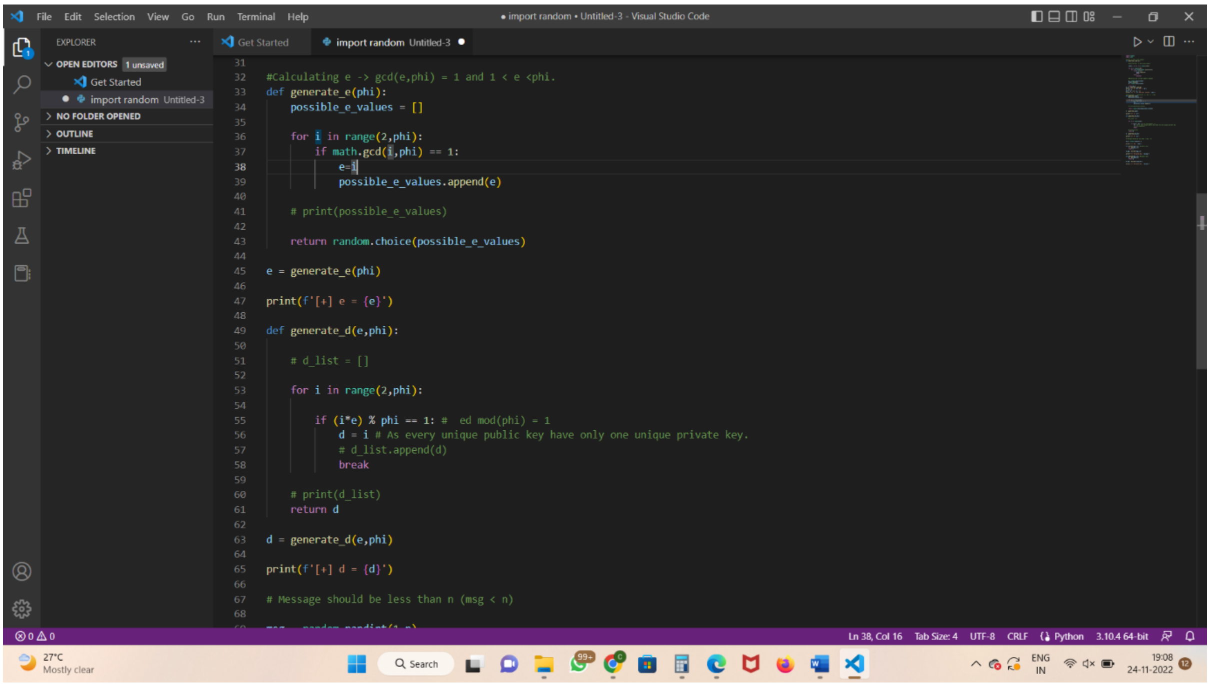
Code:

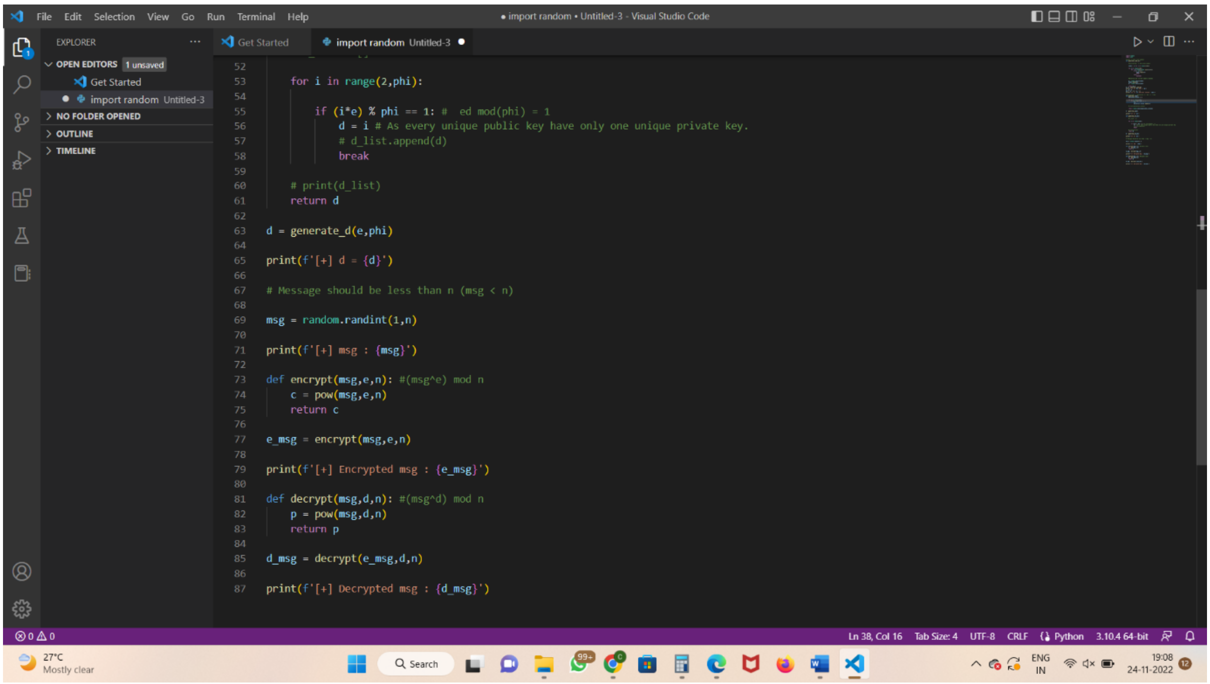
```
import random
import math

#select 2 large prime numbers
def generate_p_and_q():

        #Calculating 1 to 100 prime numbers

        numbs = [i for i in range(2,101)]

        for n in range(2,101):
                for i in range(2,math.ceil(n/2)+1):
                        if n % i == 0:
                                numbs.remove(n)
                                break
                        else:
                                continue

        #Selecting any 2 prime numbers randomly

        p = random.choice(numbs)
        numbs.remove(p)
        q = random.choice(numbs)

        return p, q

p,q =  generate_p_and_q()

print(f'[+] p = {p} and q = {q}')

n = p * q

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


print(f'[+] n = {n} and euler totient = {phi}')


#Calculating e -> gcd(e,phi) = 1 and 1 < e <phi.
def generate_e(phi):
        possible_e_values = []

        for i in range(2,phi):
                if math.gcd(i,phi) == 1:
                        e=i
                        possible_e_values.append(e)

        # print(possible_e_values)

        return random.choice(possible_e_values)
```

```
e = generate_e(phi)

print(f'[+] e = {e}')

def generate_d(e,phi):

        # d_list = []

        for i in range(2,phi):

                if (i*e) % phi == 1: #  ed mod(phi) = 1
                        d = i # As every unique public key have only one
unique private key.
                        # d_list.append(d)
                        break

        # print(d_list)
        return d

d = generate_d(e,phi)

print(f'[+] d = {d}')

# Message should be less than n (msg < n)

msg = random.randint(1,n)

print(f'[+] msg : {msg}')

def encrypt(msg,e,n): #(msg^e) mod n
        c = pow(msg,e,n)
        return c

e_msg = encrypt(msg,e,n)

print(f'[+] Encrypted msg : {e_msg}')

def decrypt(msg,d,n): #(msg^d) mod n
        p = pow(msg,d,n)
        return p

d_msg = decrypt(e_msg,d,n)

print(f'[+] Decrypted msg : {d_msg}')
```

Implementation screenshots:







Output screenshots: