Program 3:

Aim: Implement RSA algorithm for encryption and decryption of given data

Theory:

The RSA algorithm is a widely used method for encrypting and decrypting messages. It is named after its creators, **Ron Rivest**, **Adi Shamir**, and **Leonard Adleman**, who developed it in **1977**. The RSA algorithm is based on the difficulty of factoring large numbers, and it is widely considered to be a secure method for encrypting data. The security of the RSA algorithm is based on the fact that it is computationally infeasible to factorize a large composite number into its prime factors. In other words, given a number that is the product of two large prime numbers, it is very difficult to figure out what those prime numbers are. This is the basis of the 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.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

mechanism behind the RSA algorithm:

Select two prime no's. Suppose **P = 53 and Q = 59.**
Now First part of the Public key : n = P*Q = 3127.

We also need a small exponent say **e :**
But e Must be
An integer.
Not be a factor of Φ(n).
1 < e < Φ(n) [Φ(n) is discussed below],
Let us now consider it to be equal to 3.
Our Public Key is made of n and e

Generating Private Key:
We need to calculate Φ(n) :
Such that **Φ(n) = (P-1)(Q-1)**
so, Φ(n) = 3016
Now calculate Private Key, **d :**
d = (k*Φ(n) + 1) / e for some integer k
For k = 2, value of d is 2011.

Public Key (n = 3127 and e = 3) and Private Key(d = 2011) Now we will encrypt **"HI"**:

Algorithm:

1. Select **p**, **q** (**p** and **q** both prime and **p** not equal to **q**)
2. Calculate **n = p * q**
3. Calculate totient, **t = (p -1) * (q — 1)**
4. Select **e** using **gcd(t, e) = 1** where **1 < e < t**
5. Calculate **d** using **(d * e % t = 1)**
6. Consider **e** as **Public Key** and **d** as a **Private Key.**
7. For encryption, **Cipher Text = (Message ^ e) % n** (where, Message < n)
8. For decryption, **Message = (Cipher Text ^ d) % n**

Conclusion: the RSA algorithm is a widely used method for encrypting and decrypting messages. It is based on the difficulty of factoring large numbers, and it is considered to be a secure method for encrypting data.

It has several advantages, including its simplicity and its use of public and private keys.

```
# Python for RSA asymmetric cryptographic algorithm.

# For demonstration, values are

# relatively small compared to practical application

import math

def gcd(a, h):

    temp = 0

    while(1):

        temp = a % h

        if (temp == 0):

            return h

        a = h

        h = temp

p = 3

q = 7

n = p*q

e = 2

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

```
while (e < phi):
    # e must be co-prime to phi and
    # smaller than phi.
    if(gcd(e, phi) == 1):
        break
    else:
        e = e+1
# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient
k = 2
d = (1 + (k*phi))/e
# Message to be encrypted
msg = 12.0
print("Message data = ", msg)
# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)
```

```
# Decryption m = (c ^ d) % n

m = pow(c, d)

m = math.fmod(m, n)

print("Original Message Sent = ", m)
```

Output

```
Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000
```