# SUB: Information Security

## AY 2023-24 (Semester-V)

## Experiment No: 7

**Name:**Bhuvi Ghosh
**SAPID**:60009210191

**Aim:** To study and implement RSA algorithm.

**Theory:**

RSA was invented by Ron Rivest, Adi Shamir, and Len Adleman and hence, it is termed as RSA cryptosystem.We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of
keys is described below –

● Generate the RSA modulus (n)
○ Select two large primes, p and q.
○ Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

● Find Derived Number (e)
○ Number e must be greater than 1 and less than (p − 1)(q − 1).
○ There must be no common factor for e and (p − 1)(q − 1) except for 1. In other words two numbers e and (p – 1)(q – 1) are coprime.

● Form the public key
○ The pair of numbers (n, e) form the RSA public key and is made public. Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.

● Generate the private key
○ Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.

## SUB: Information Security

○ Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).
○ This relationship is written mathematically as follows –
$ed = 1 \bmod (p - 1)(q - 1)$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.
Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

● Let two primes be $p = 7$ and $q = 13$. Thus, modulus $n = pq = 7 \times 13 = 91$.
● Select $e = 5$, which is a valid choice since there is no number that is common factor of 5 and
$(p - 1)(q - 1) = 6 \times 12 = 72$, except for 1.
● The pair of numbers $(n, e) = (91, 5)$ forms the public key and can be made available to anyone
whom we wish to be able to send us encrypted messages.
● Input $p = 7$, $q = 13$, and $e = 5$ to the Extended Euclidean Algorithm. The output will be $d =$
29.
● Check that the d calculated is correct by computing –
$de = 29 \times 5 = 145 = 1 \bmod 72$
● Hence, public key is (91, 5) and private keys is (91, 29).

## SUB: Information Security

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.
Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and
(p − 1)(q − 1) = 6 × 12 = 72, except for 1.
- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone
whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d =
29.
- Check that the d calculated is correct by computing −
de = 29 × 5 = 145 = 1 mod 72
- Hence, public key is (91, 5) and private keys is (91, 29).

## SUB: Information Security

Encryption and Decryption
Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.

RSA Encryption
● Suppose the sender wish to send some text message to someone whose public key is (n, e).
● The sender then represents the plaintext as a series of numbers less than n
● To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple
mathematical step as −

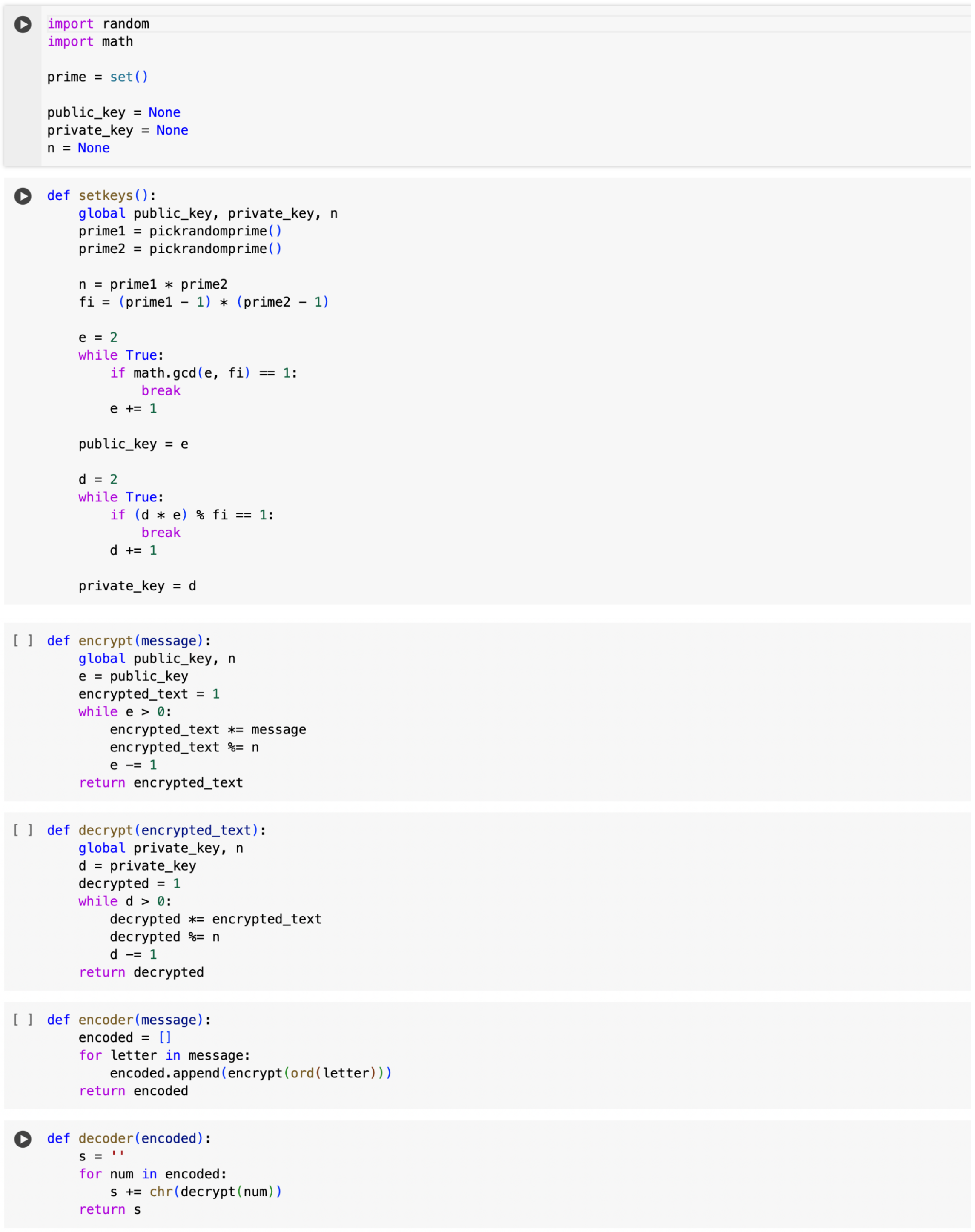
C = Pe mod n
● In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then
reduced modulo n. This means that C is also a number less than n.
● Returning to our Key Generation example with plaintext P = 10, we get ciphertext C −
C = 105 mod 91

RSA Decryption
● The decryption process for RSA is also very straightforward. Suppose that the receiver of
public-key pair (n, e) has received a ciphertext C.
● Receiver raises C to the power of his private key d. The result modulo n will be the plaintext
P.
Plaintext = Cd mod n
● Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −
Plaintext = 8229 mod 91 = 10

**Conclusion:**

RSA Algorithm for symmetric key cryptography has been successfully implemented.

**SUB: Information Security**



```
import random
import math

prime = set()

public_key = None
private_key = None
n = None
```

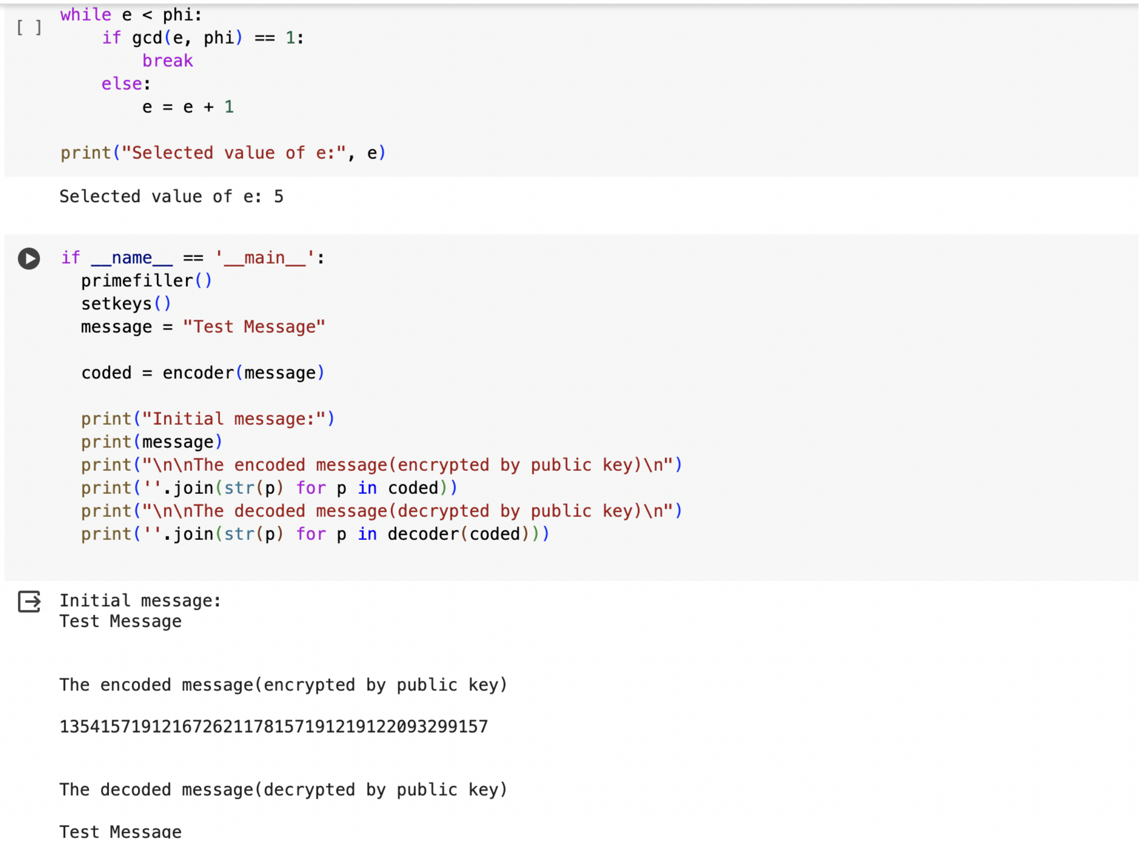
```
def setkeys():
    global public_key, private_key, n
    prime1 = pickrandomprime()
    prime2 = pickrandomprime()

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

    e = 2
    while True:
        if math.gcd(e, fi) == 1:
            break
        e += 1

    public_key = e

    d = 2
    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 = []
        for letter in message:
            encoded.append(encrypt(ord(letter)))
        return encoded
```

```
def decoder(encoded):
    s = ''
    for num in encoded:
        s += chr(decrypt(num))
    return s
```



```
while e < phi:
    if gcd(e, phi) == 1:
        break
    else:
        e = e + 1

print("Selected value of e:", e)
```

```
Selected value of e: 5
```

```
if __name__ == '__main__':
  primefiller()
  setkeys()
  message = "Test Message"

  coded = encoder(message)

  print("Initial message:")
  print(message)
  print("\n\nThe encoded message(encrypted by public key)\n")
  print(''.join(str(p) for p in coded))
  print("\n\nThe decoded message(decrypted by public key)\n")
  print(''.join(str(p) for p in decoder(coded)))
```

```
Initial message:
Test Message


The encoded message(encrypted by public key)

135415719121672621178157191219122093299157


The decoded message(decrypted by public key)

Test Message
```