Aayush Shah
19BCE245
23 August 2022

BlockChain Technology

Practical 1

To implement digital signature to sign and verify authenticated user. Also, show a message when tampering is detected

• **RSA algorithm steps :**

1. p,q - prime numbers
2. n = pxq
3. phi(n) = (p-1)(q-1)
4. e => GCD(e, phi(n)) = 1; {e,d < phi(n)}
5. (e.d)% phi(n) = 1; {d!=p,d!=q} | {d = e - 1 mod phi(n)}

C = (M^e)%n

M = (C^d)%n

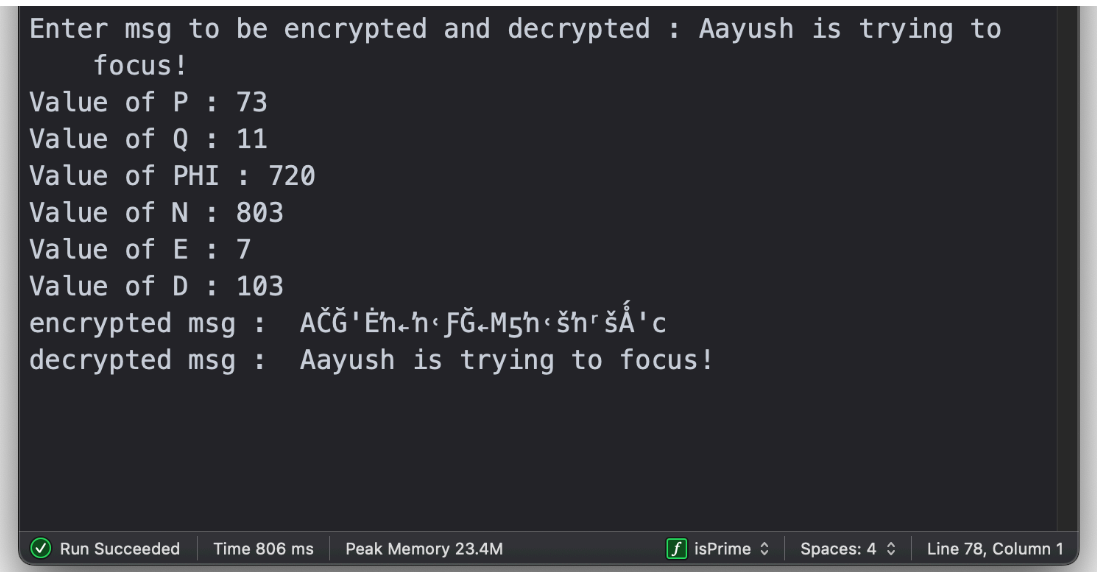
• **Code :**

practical1.1

```
# 19BCE245 Aayush Shah
# To implement digital signature to sign and verify
authenticated user. Also, show a message when tampering
is detected.

# Part 1 : RSA algorithm

import numpy as np
import math

def getRandomPrimeInteger(bounds):
    for i in range(bounds.__len__()-1):
        if bounds[i + 1] > bounds[i]:
            x = bounds[i] +
np.random.randint(bounds[i+1]-bounds[i])
            if isPrime(x):
                return x

        else:
            if isPrime(bounds[i]):
                return bounds[i]

        if isPrime(bounds[i + 1]):
            return bounds[i + 1]

    newBounds = [0 for i in range(2*bounds.__len__() -
1)]
    newBounds[0] = bounds[0]
    for i in range(1, bounds.__len__()):
        newBounds[2*i-1] = int((bounds[i-1] + bounds[i])/
2)
        newBounds[2*i] = bounds[i]

    return getRandomPrimeInteger(newBounds)

def isPrime(x):
    count = 0
    for i in range(int(x/2)):
        if x % (i+1) == 0:
            count = count+1
    return count == 1

bounds = [10, 100]
msg = input("Enter msg to be encrypted and decrypted : ")
p = getRandomPrimeInteger(bounds) # 1st prime number
q = getRandomPrimeInteger(bounds) # 2nd prime number

while p==q: # will select distinct prime numbers
    p = getRandomPrimeInteger(bounds) # 1st prime number
    q = getRandomPrimeInteger(bounds) # 2nd prime number

n = p*q
phi = (p-1)*(q-1)
#phi = int((p-1)*(q-1)/math.gcd(p-1,q-1))
```

```
print(f'Value of P : {p}')
print(f'Value of Q : {q}')
print(f'Value of PHI : {phi}')
print(f'Value of N : {n}')

e = 1
for i in range(2,phi):
    if math.gcd(i,phi)==1:
        e = i
        break
if e==1:
    print('> Cannot find value for e')
else:
    print(f'Value of E : {e}')

d = -1
for i in range(9):
    x = 1+(i*phi)
#   print(f'i={i};x={x}')
    if x%e==0:
        d = x//e
        break
if d==-1:
    print('> Cannot find value for d')
else:
    print(f'Value of D : {d}')

encrypted_chars = []
encrypted_msg = ""
for character in msg:
    encrypted_chars.append(chr(pow(ord(character), e)%n))
    encrypted_msg+=(chr(pow(ord(character), e)%n))
print('encrypted msg : ',encrypted_msg)

decrypted_chars = []
decrypted_msg = ""
for character in encrypted_chars:
    decrypted_chars.append(chr(pow(ord(character), d)%n))
    decrypted_msg+=decrypted_chars[-1]
print('decrypted msg : ',decrypted_msg)
```

• **Output :**



```
Enter msg to be encrypted and decrypted : Aayush is trying to
    focus!
Value of P : 73
Value of Q : 11
Value of PHI : 720
Value of N : 803
Value of E : 7
Value of D : 103
encrypted msg :  AČĞ'Ėŉ₊ŉ‹ƑĞ₊M5ŉ‹šŉʳšÅ'c
decrypted msg :  Aayush is trying to focus!
```

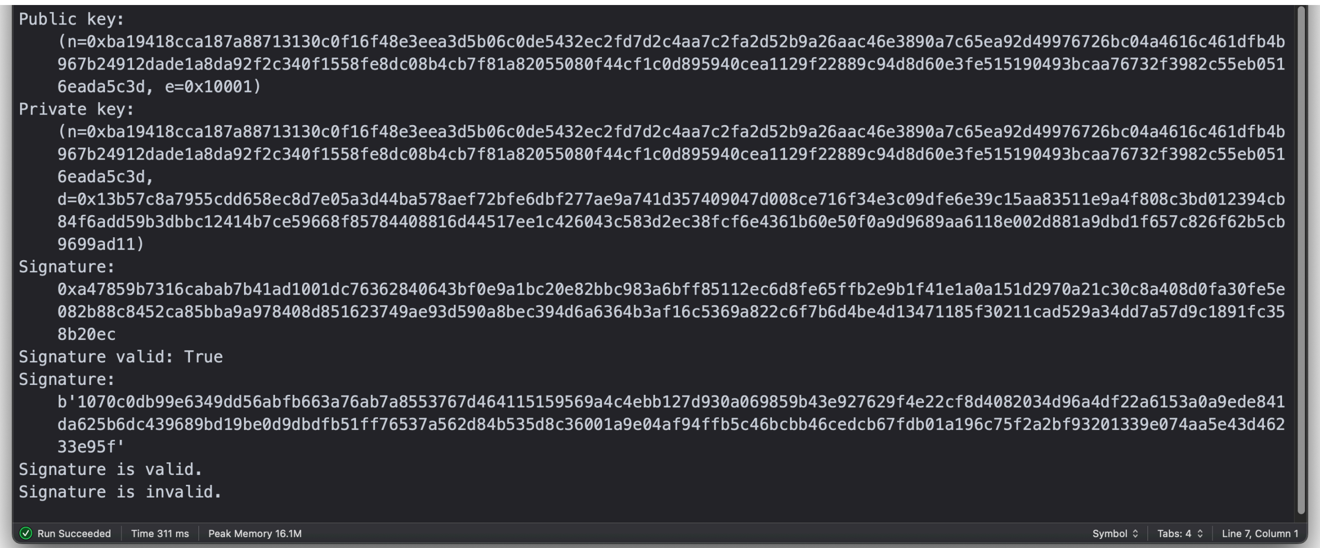
<u>**practical1.2.1**</u>

```python
# 19BCE245 Aayush Shah
# To implement digital signature to sign and verify
authenticated user. Also, show a message when tampering
is detected.

# Part 2 : Signing and Verifying with Detection of
Tempering.

# installed `pycryptodome` with `pip install
pycryptodome`

from Crypto.PublicKey import RSA
from Crypto.Signature.pkcs1_15 import PKCS115_SigScheme
from Crypto.Hash import SHA256
import binascii
from hashlib import sha512

# Generate 1024-bit RSA key-pair
keyPair = RSA.generate(bits=1024)
```

```
print(f"Public key:  (n={hex(keyPair.n)}, 
e={hex(keyPair.e)})")
print(f"Private key: (n={hex(keyPair.n)}, 
d={hex(keyPair.d)})")

#pubKey = hex(keyPair.e)
pubKey = keyPair.publickey()


# RSA sign the message
msg = b'Aayush, focus please!'
hash = int.from_bytes(sha512(msg).digest(), 
byteorder='big')
signature = pow(hash, keyPair.d, keyPair.n)
print("Signature:", hex(signature))

# RSA verify signature
hashFromSignature = pow(signature, keyPair.e, keyPair.n)
print("Signature valid:", hash == hashFromSignature)


# Sign the message using the PKCS#1 v1.5 signature scheme 
(RSASP1)
msg = b'Aayush, focus please!'

hash = SHA256.new(msg)
signer = PKCS115_SigScheme(keyPair)
signature = signer.sign(hash)
print("Signature:", binascii.hexlify(signature))

# Verify valid PKCS#1 v1.5 signature (RSAVP1)
msg = b'Aayush, focus please!'
hash = SHA256.new(msg)
verifier = PKCS115_SigScheme(pubKey)
try:
     verifier.verify(hash, signature)
     print("Signature is valid.")
except:
     print("Signature is invalid.")

# Verify invalid PKCS#1 v1.5 signature (RSAVP1)
msg = b'A tampered message'
hash = SHA256.new(msg)
```

```python
verifier = PKCS115_SigScheme(pubKey)
try:
    verifier.verify(hash, signature)
    print("Signature is valid.")
except:
    print("Signature is invalid.")
```

- **Output :**



```
Public key:
    (n=0xba19418cca187a88713130c0f16f48e3eea3d5b06c0de5432ec2fd7d2c4aa7c2fa2d52b9a26aac46e3890a7c65ea92d49976726bc04a4616c461dfb4b
    967b24912dade1a8da92f2c340f1558fe8dc08b4cb7f81a82055080f44cf1c0d895940cea1129f22889c94d8d60e3fe515190493bcaa76732f3982c55eb051
    6eada5c3d, e=0x10001)
Private key:
    (n=0xba19418cca187a88713130c0f16f48e3eea3d5b06c0de5432ec2fd7d2c4aa7c2fa2d52b9a26aac46e3890a7c65ea92d49976726bc04a4616c461dfb4b
    967b24912dade1a8da92f2c340f1558fe8dc08b4cb7f81a82055080f44cf1c0d895940cea1129f22889c94d8d60e3fe515190493bcaa76732f3982c55eb051
    6eada5c3d,
    d=0x13b57c8a7955cdd658ec8d7e05a3d44ba578aef72bfe6dbf277ae9a741d357409047d008ce716f34e3c09dfe6e39c15aa83511e9a4f808c3bd012394cb
    84f6add59b3dbbc12414b7ce59668f85784408816d44517ee1c426043c583d2ec38fcf6e4361b60e50f0a9d9689aa6118e002d881a9dbd1f657c826f62b5cb
    9699ad11)
Signature:
    0xa47859b7316cabab7b41ad1001dc76362840643bf0e9a1bc20e82bbc983a6bff85112ec6d8fe65ffb2e9b1f41e1a0a151d2970a21c30c8a408d0fa30fe5e
    082b88c8452ca85bba9a978408d851623749ae93d590a8bec394d6a6364b3af16c5369a822c6f7b6d4be4d13471185f30211cad529a34dd7a57d9c1891fc35
    8b20ec
Signature valid: True
Signature:
    b'1070c0db99e6349dd56abfb663a76ab7a8553767d464115159569a4c4ebb127d930a069859b43e927629f4e22cf8d4082034d96a4df22a6153a0a9ede841
    da625b6dc439689bd19be0d9dbdfb51ff76537a562d84b535d8c36001a9e04af94ffb5c46bcbb46cedcb67fdb01a196c75f2a2bf93201339e074aa5e43d462
    33e95f'
Signature is valid.
Signature is invalid.
```

- **Learning Outcomes :**

From this practical, I learned about RSA algorithm, digital signature signing and verifying along with detection of tampering.

In Blockchain, digital signing is a mechanism that verifies the user's impressions of the transaction. The private key is used to sign the digital transaction, while the associated public key is used to assist authorise the sender.

The miners who know her public key will next review the transaction conditions and validate the signature. Once the transaction's legitimacy is validated, the block containing it is available for finalisation by a validator/miner.

Data stored in various types of database management systems is always vulnerable to alteration from internal or external sources. We deploy a variety of Database Tampering Detection technologies to detect such unauthorised changes in the database.