VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Aditi Patra

21BIT0125

L55 + 56

LAB2

Blind Sign Authentication

```
#Blind Sign Authentication


import random


print("Aditi Patra 21BIT0125")
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
```

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

m = int(input("Enter your message: "))

p = int(input("Enter first prime number: "))
while not is_prime(p):
    print("The number entered is not prime. Please enter a prime number.")
    p = int(input("Enter first prime number: "))

q = int(input("Enter second prime number: "))
while not is_prime(q):
    print("The number entered is not prime. Please enter a prime number.")
    q = int(input("Enter second prime number: "))

#-------------------Key Generation----------------------------------
phi = (p-1) * (q-1)
print("Φ = ", phi)
N = p * q
print("N = ", N)

# Selection of e
e = random.randint(2, phi - 1)
while gcd(e, phi) != 1: #Check if e and phi are coprime
    e = random.randint(2, phi - 1)

print("Selected e = ", e)
```

```
# Private Key
def modular_inverse(a, m):
    g, x, _ = extended_gcd(a, m)
    if g != 1:
        raise ValueError("Modular inverse does not exist")
    return x % m

def extended_gcd(a, b):
    if b == 0:
        return a, 1, 0
    g, x1, y1 = extended_gcd(b, a % b)
    x = y1
    y = x1 - (a // b) * y1
    return g, x, y

d = modular_inverse(e, phi)  # Calculate the modular multiplicative inverse of e modulo phi
print("Process d = ", d)
#---------------------------------------------------------------------

#---------------------Encryption-----------------------------------------
r = random.randint(2, N-1)  # Generate a random value for r between 2 and N-1
while gcd(r, N) != 1:  # Check if r and N are coprime
    r = random.randint(2, N-1)
print("r = ", r)

#Blind Factor
bf = pow(r, e, N)
print("Blind Factor = ", bf)
```

```
#Blind Message
bm = (pow(r, e, N) * m) % N
print("Blind Message = ", bm)
#-------------------------------------------------------------------


#--------------------Sign Generation---------------------------------
sg = pow(bm, d, N) #Sign Generated
print("Sign Generated = ", sg)
#-------------------------------------------------------------------


#--------------------Sign Verification-------------------------------
sv = pow(r, -1, N)
print("Sign Verified = ", sv)
bdm = (sg * sv) % N #Blinded Message
print("Blinded Message = ", bdm)
fm = pow(bdm, e, N)
print("The Computed Message = ", fm)
#-------------------------------------------------------------------


#-------------------------Verification of the Code--------------------
if fm == m:
    print("The process is correct as the computed message is the same as the original Message")
else:
    print("The process is incorrect as the computed message is not the same as the original Message")


#-------------------------------------------------------------------
```

CODE SNIPPET:

BlindSign.py - C:/Users/patra/Desktop/books/5th sem/infosec/LAB 2/BlindSign.py (3.9.6)

File Edit Format Run Options Window Help

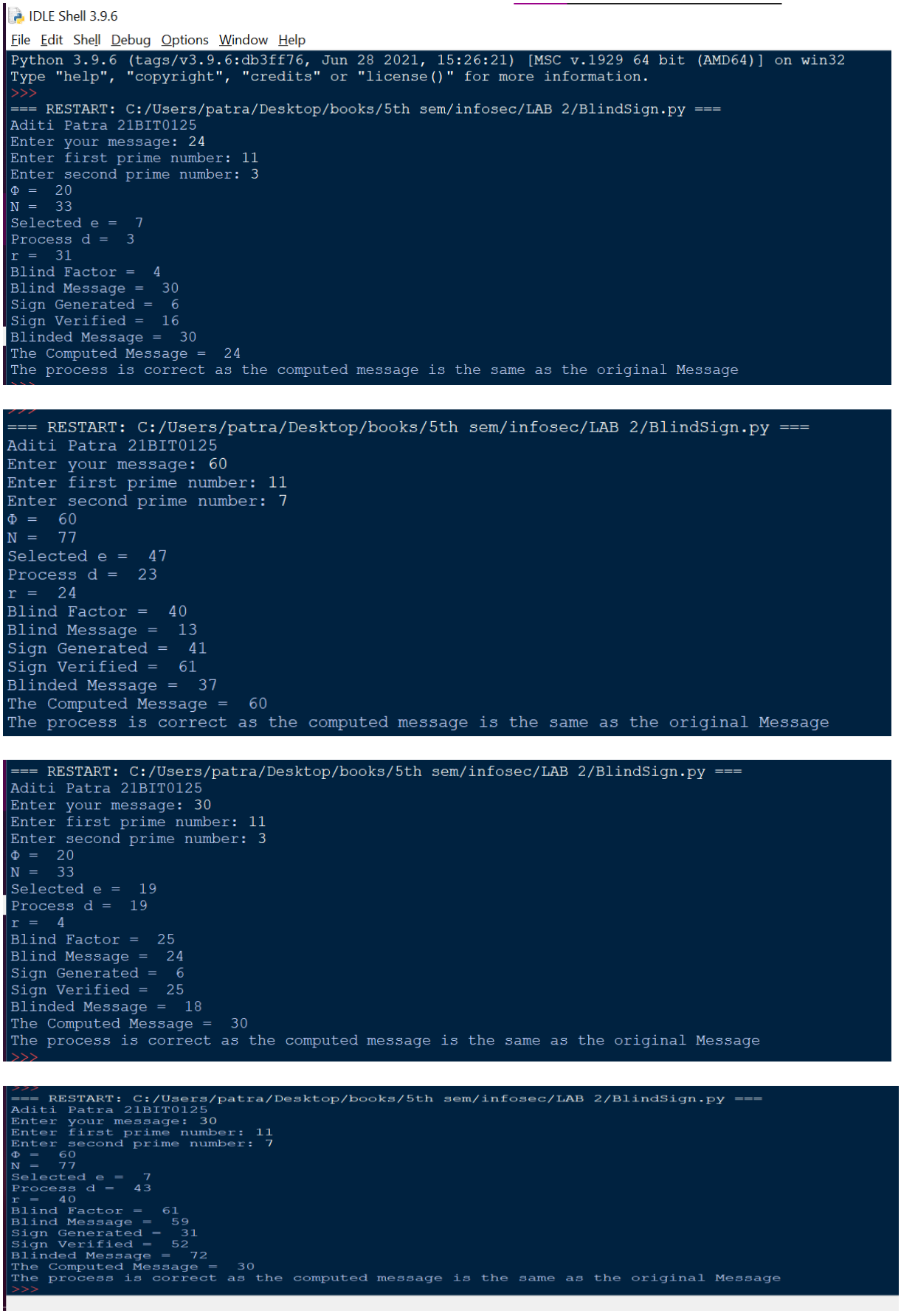
```
#Blind Sign Authentication

import random

print("Aditi Patra 21BIT0125")
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

m = int(input("Enter your message: "))

p = int(input("Enter first prime number: "))
while not is_prime(p):
    print("The number entered is not prime. Please enter a prime number.")
    p = int(input("Enter first prime number: "))

q = int(input("Enter second prime number: "))
while not is_prime(q):
    print("The number entered is not prime. Please enter a prime number.")
    q = int(input("Enter second prime number: "))

#-----------------Key Generation-----------------------------------
phi = (p-1) * (q-1)
print("Φ = ", phi)
N = p * q
print("N = ", N)

# Selection of e
e = random.randint(2, phi - 1)
while gcd(e, phi) != 1: #Check if e and phi are coprime
    e = random.randint(2, phi - 1)

print("Selected e = ", e)
```

```
# Private Key
def modular_inverse(a, m):
    g, x, _ = extended_gcd(a, m)
    if g != 1:
        raise ValueError("Modular inverse does not exist")
    return x % m

def extended_gcd(a, b):
    if b == 0:
        return a, 1, 0
    g, x1, y1 = extended_gcd(b, a % b)
    x = y1
    y = x1 - (a // b) * y1
    return g, x, y

d = modular_inverse(e, phi)  # Calculate the modular multiplicative inverse of e modulo phi
print("Process d = ", d)
#-----------------------------------------------------------------

#-----------------Encryption-----------------------------------
r = random.randint(2, N-1)  # Generate a random value for r between 2 and N-1
while gcd(r, N) != 1:  # Check if r and N are coprime
    r = random.randint(2, N-1)
print("r = ", r)

#Blind Factor
bf = pow(r, e, N)
print("Blind Factor = ", bf)

#Blind Message
bm = (pow(r, e, N) * m) % N
print("Blind Message = ", bm)
#-----------------------------------------------------------------

#-----------------Sign Generation-----------------------------------
sg = pow(bm, d, N) #Sign Generated
print("Sign Generated = ", sg)
#-----------------------------------------------------------------
```

```
#-----------------Sign Generation-----------------------------------
sg = pow(bm, d, N) #Sign Generated
print("Sign Generated = ", sg)
#-----------------------------------------------------------------

#-----------------Sign Verification-----------------------------------
sv = pow(r, -1, N)
print("Sign Verified = ", sv)
bdm = (sg * sv) % N #Blinded Message
print("Blinded Message = ", bdm)
fm = pow(bdm, e, N)
print("The Computed Message = ", fm)
#-----------------------------------------------------------------

#-----------------Verification of the Code-----------------------------------
if fm == m:
    print("The process is correct as the computed message is the same as the original Message")
else:
    print("The process is incorrect as the computed message is not the same as the original Message")

#-----------------------------------------------------------------
```

Ln: 18 Col: 0

OUTPUT:



```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 2/BlindSign.py ===
Aditi Patra 21BIT0125
Enter your message: 24
Enter first prime number: 11
Enter second prime number: 3
Φ =  20
N =  33
Selected e =  7
Process d =  3
r =  31
Blind Factor =  4
Blind Message =  30
Sign Generated =  6
Sign Verified =  16
Blinded Message =  30
The Computed Message =  24
The process is correct as the computed message is the same as the original Message
>>>
>>>
=== RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 2/BlindSign.py ===
Aditi Patra 21BIT0125
Enter your message: 60
Enter first prime number: 11
Enter second prime number: 7
Φ =  60
N =  77
Selected e =  47
Process d =  23
r =  24
Blind Factor =  40
Blind Message =  13
Sign Generated =  41
Sign Verified =  61
Blinded Message =  37
The Computed Message =  60
The process is correct as the computed message is the same as the original Message
=== RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 2/BlindSign.py ===
Aditi Patra 21BIT0125
Enter your message: 30
Enter first prime number: 11
Enter second prime number: 3
Φ =  20
N =  33
Selected e =  19
Process d =  19
r =  4
Blind Factor =  25
Blind Message =  24
Sign Generated =  6
Sign Verified =  25
Blinded Message =  18
The Computed Message =  30
The process is correct as the computed message is the same as the original Message
>>>
>>>
=== RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 2/BlindSign.py ===
Aditi Patra 21BIT0125
Enter your message: 30
Enter first prime number: 11
Enter second prime number: 7
Φ =  60
N =  77
Selected e =  7
Process d =  43
r =  40
Blind Factor =  61
Blind Message =  59
Sign Generated =  31
Sign Verified =  52
Blinded Message =  72
The Computed Message =  30
The process is correct as the computed message is the same as the original Message
>>>
```