Information Security & Cryptography
ASSIGNMENT- 8

U20CS005
BANSI MARAKANA

Implement the Signature scheme- Digital Signature Standard using RSA.

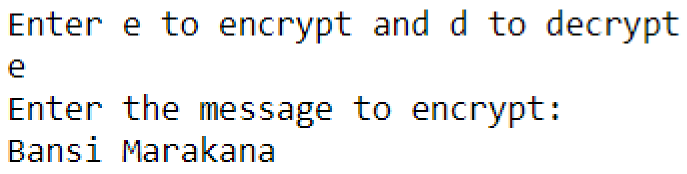
```
import Crypto.Util.number as CryNum
import random
import gmpy2
import sys
import hashlib

#genPrime returns a prime number for a fixed bit size
def genPrime(size):
    return (CryNum.getPrime(size))

def genN(p,q):
    return (p*q)

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

#getRandE return the value of e such that e is coprime of fi(n)
def genRandE(phin):
    e=65537
    g = gcd(e, phin)
    while g != 1:
        e = random.randrange(1, phin)
        g = gcd(e, phin)
    return e

#genPrivKey return the private key d
def genPrivKey(phin, e):
    k = genRand(512)
    return ((((k*phin) +1))/e)

#encrypts message digest(sha224) and then stores private key and public key pair into a file
and also the message and the encrypted digest
```

```
def encrypt():
    print("Enter the message to encrypt:")
    msg = input()
    # p and q large prime numbers between length 512 and 2048
    p, q = genPrime(512), genPrime(512)
    #if p = q then genrate another p and q
    while p == q:
        p, q = genPrime(512), genPrime(512)
    n = genN(p,q)
    phin = genN(p-1, q-1)
    e = genRandE(phin)
    d = gmpy2.invert(e, phin)
    #Public Key = (e,n) Private Key = d
    digest = hashlib.sha256(msg.encode()).hexdigest()
    #converting the digest to its ascii value
    m = ''
    for i in digest:
        m = m+str(ord(i))
    #encypting the digest using RSA algorithm
    encDigest = pow(int(m),d,n)
    '''
    print("p = "+str(p)+"\nq = "+str(q)+"\nN = "+str(n)+"\nO(n) = "+str(phin)+"\ne = "+str(e)+"\nd =
"+str(d)+"\nDigest = "+str(digest)+"\nEncrypted Digest = "+str(encDigest))
    '''
    #writing message+digest public key and private key to respective file file
    with open('transfermsg.txt', 'w') as file:
        file.write(msg+str(encDigest))
        file.close()
    with open('publicKey.txt', 'w') as file:
        file.write(str(e).strip()+"\n"+str(n).strip())
        file.close()
    with open('privateKey.txt', 'w') as file:
        file.write(str(d).strip())
        file.close()

#decrypt opens the file seperates message and digest, decrypt the digest using the public key
and compares it to caluclated hash of the message
def decrypt():
    msg = open('transfermsg.txt').read()
    #calulating the length of message + digest
    l = len(msg)
    #calculating the length of message
    start = 0
    for i in msg:
        start += 1
        if i.isdigit() == True:
            break
    #getting the lenght of encrypted digest
    start = (l - int(start)+1) * -1
    #sperating digest from message
    digest = msg[start:l]
    msg = msg.replace(digest, '')
    pk = open('publicKey.txt').read()
    pk = pk.split("\n")
    e = pk[0]
    n = pk[1]
    o = ''
    e = int(e)
    n = int(n)
    #decrypting the encrypted digest using RSA algorithm
    o = str(pow(int(digest), e, n))
    #calculating the hash of the message using sha224
    msg = hashlib.sha256(msg.encode()).hexdigest()
    #converting the message digest to its ascii value
    m = ''
    for i in msg:
        m = m+str(ord(i))
    '''
    print("Recieved Digest = "+str(digest)+"\nRecieved Message = "+str(msg)+"\ne =
"+str(e)+"\nn = "+str(n)+"\nCaculated hash = "+str(m))
    '''
    #if decypted digest == calculated hash then the sender is verified
    if m == o:
        print("Sender Verified.")
    else:
        print("Not able to verify the sender and message has been tampered.")

if __name__ == '__main__':
    sys.maxsize = sys.maxsize*sys.maxsize
    print("Enter e to encrypt and d to decrypt")
    mode = input()
    if mode == 'e':
        encrypt()
    elif mode == 'd':
        decrypt()
    else:
        print("Wrong choice")
```

OUTPUT:

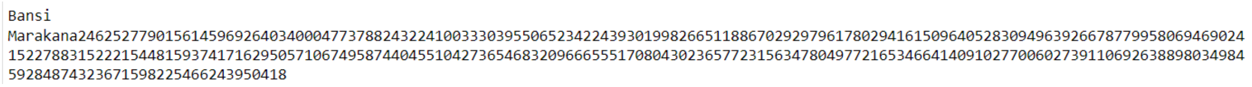
Message send by sender after digital signature:

Input:



```
Enter e to encrypt and d to decrypt
e
Enter the message to encrypt:
Bansi Marakana
```

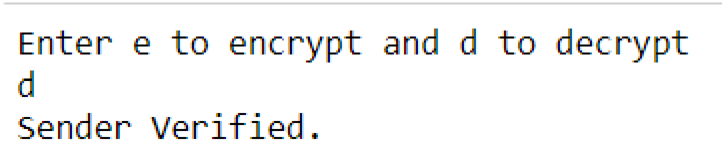
Message after appending hash value and encryption:



```
Bansi
Marakana2462527790156145969264034000477378824322410033303955065234224393019982665118867029297961780294161509640528309496392667877995806946902 4
1522788315222154481593741716295057106749587440455104273654683209666555170804302365772315634780497721653466414091027700602739110692638898034984
5928487432367159822546624395041 8
```

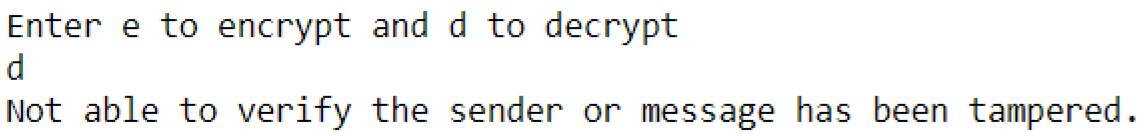
After receiver decrypts the message:

If hash value matches then:



```
Enter e to encrypt and d to decrypt
d
Sender Verified.
```

If hash value does not matches then:



```
Enter e to encrypt and d to decrypt
d
Not able to verify the sender or message has been tampered.
```