Practical 1 - Implement to perform digital signature to sign and verify authenticated user. Also, show a message when tampering is detected.

→ Below code is a simple implementation of RSA in python.
→ RSA algorithm is asymmetric cryptography algorithm.
→ Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key.
→ The public key consists of two numbers where one number is multiplication of two large prime numbers.
→ Private key is also derived from the same two prime numbers.
→ Therefore, encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially.

+==+

```
import random
from hashlib import sha256


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


def extended_gcd(aa, bb):
    lastremainder, remainder = abs(aa), abs(bb)
    x, lastx, y, lasty = 0, 1, 1, 0
    while remainder:
```

```
        lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)
        x, lastx = lastx - quotient*x, x
        y, lasty = lasty - quotient*y, y
    return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb < 0 else 1)

#Euclid's extended algorithm for finding the multiplicative inverse of two numbers
def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    return x % m

def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True

def generate_keypair(p, q):
    if not (is_prime(p) and is_prime(q)):
```

```
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')

    n = p * q

    #Phi is the totient of n
    phi = (p-1) * (q-1)

    #Choose an integer e such that e and phi(n) are coprime
    e = random.randrange(1, phi)

    #Use Euclid's Algorithm to verify that e and phi(n) are comprime
    g = coprime(e, phi)

    while g != 1:
        e = random.randrange(1, phi)
        g = coprime(e, phi)

    #Use Extended Euclid's Algorithm to generate the private key
    d = modinv(e, phi)

    #Return public and private keypair
    #Public key is (e, n) and private key is (d, n)
    return ((e, n), (d, n))
```

```
def encrypt(privatek, plaintext):
    #Unpack the key into it's components
    key, n = privatek

    #Convert each letter in the plaintext to numbers based on the character using
a^b mod m

    numberRepr = [ord(char) for char in plaintext]
    print("Number representation before encryption: ", numberRepr)
    cipher = [pow(ord(char),key,n) for char in plaintext]

    #Return the array of bytes
    return cipher

def decrypt(publick, ciphertext):
    #Unpack the key into its components
    key, n = publick

    #Generate the plaintext based on the ciphertext and key using a^b mod m
    numberRepr = [pow(char, key, n) for char in ciphertext]
    plain = [chr(pow(char, key, n)) for char in ciphertext]

    print("Decrypted number representation is: ", numberRepr)

    #Return the array of bytes as a string
    return ''.join(plain)
```

```
def hashFunction(message):
    hashed = sha256(message.encode("UTF-8")).hexdigest()
    return hashed

def verify(receivedHashed, message):
    ourHashed = hashFunction(message)
    if receivedHashed == ourHashed:
        print("Verification successful: ", )
        print(receivedHashed, " = ", ourHashed)
    else:
        print("Verification failed")
        print(receivedHashed, " != ", ourHashed)

def main():
    p = int(input("Enter a prime number (17, 19, 23, etc): "))
    q = int(input("Enter another prime number (Not one you entered above): "))
    #p = 17
    #q=23

    print("Generating your public/private keypairs now . . .")
    public, private = generate_keypair(p, q)

    print("Your public key is ", public ," and your private key is ", private)
    message = input("Enter a message to encrypt with your private key: ")
```

```
print("")

hashed = hashFunction(message)

print("Encrypting message with private key ", private ," . . .")
encrypted_msg = encrypt(private, hashed)
print("Your encrypted hashed message is: ")
print(''.join(map(lambda x: str(x), encrypted_msg)))
#print(encrypted_msg)

print("")
print("Decrypting message with public key ", public ," . . .")

decrypted_msg = decrypt(public, encrypted_msg)
print("Your decrypted message is:")
print(decrypted_msg)

print("")
print("Verification process . . .")
verify(decrypted_msg, message)

 main()
 +==============================================================+
```

Output:

```
C:\Users\kshitij\AppData\Local\Programs\Python\Python37-32\python.exe C:/Users/kshitij/Downloads/dig_sign.py
Enter a prime number (17, 19, 23, etc): 31
Enter another prime number (Not one you entered above): 17
Generating your public/private keypairs now . . .
Your public key is  (127, 527)  and your private key is  (223, 527)
Enter a message to encrypt with your private key: Hello there!

Encrypting message with private key  (223, 527)  . . .
Number representation before encryption:  [56, 57, 98, 56, 98, 56, 101, 52, 56, 54, 52, 50, 49, 52, 54, 51, 100, 55, 101, 48, 102, 53, 99, 97, 102, 54, 48, 102, 98, 57, 99, 98, 51,
Your encrypted hashed message is:
2118819121119121116239211108239169144239108289484353169620423013095204108962041918813019128923013016144108881913531081610823035395191169144204130239484144484108191968828910896289

Decrypting message with public key  (127, 527)  . . .
Decrypted number representation is:  [56, 57, 98, 56, 98, 56, 101, 52, 56, 54, 52, 50, 49, 52, 54, 51, 100, 55, 101, 48, 102, 53, 99, 97, 102, 54, 48, 102, 98, 57, 99, 98, 51, 53, 9
Your decrypted message is:
89b8b8e486421463d7e0f5caf60fb9cb35ce169b76e657ab21fc4d1d6b093603

Verification process . . .
Verification successful:
89b8b8e486421463d7e0f5caf60fb9cb35ce169b76e657ab21fc4d1d6b093603  =  89b8b8e486421463d7e0f5caf60fb9cb35ce169b76e657ab21fc4d1d6b093603

Process finished with exit code 0
```

References

[1]https://gist.github.com/JonCooperWorks/5314103

[2] RSA Algorithm in Cryptography ,https://www.geeksforgeeks.org/rsa-algorithm-cryptography/