Q1

md5um

Graphical user interface, text

Description automatically generated

Hash\_comp.py

A screenshot of a computer

Description automatically generated with medium confidence

2.

Generate private RSA key

Text

Description automatically generated

Extract public key

Graphical user interface, text, application, chat or text message

Description automatically generated

Display private key

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Display public key

Text

Description automatically generated

3. encrypt & decrypt with RSA

Encrypt

Text

Description automatically generated with medium confidence

Decrypt

Graphical user interface, text

Description automatically generated

4.

RSA signature sign - $2000

Graphical user interface, text

Description automatically generated

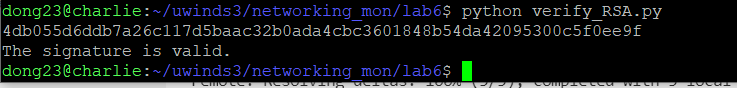
RSA signature sign - $3000

A picture containing table

Description automatically generated

The experiment shows that signatures are different for messages of slightly different.

Verify the signature



5.

Text

Description automatically generated

Entire source code

#!/usr/bin/python3

from Crypto.Cipher import AES

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Util import Padding

from Crypto.Random import get\_random\_bytes

import binascii

def encryptRSA(message, keyfile):

    key=RSA.importKey(open(keyfile).read())

    cipher=PKCS1\_OAEP.new(key)

    ciphertext=cipher.encrypt(message)

    return ciphertext

def decryptRSA(ciphertext, privatekeyfile):

    key\_str = open(privatekeyfile).read()

    prikey = RSA.importKey(key\_str, passphrase='111111')

    cipher = PKCS1\_OAEP.new(prikey)

    message = cipher.decrypt(ciphertext)

    return message

key\_hex\_string = '00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF'

key = bytearray.fromhex(key\_hex\_string)

# a.

iv = get\_random\_bytes(16)

print("sk - {}".format(binascii.hexlify(bytearray(iv))))

# b.

c1 = encryptRSA(iv, "public.pem")

print("c1 - {}".format(binascii.hexlify(bytearray(c1))))

data = b'I find the solution for P not equal NP'

# c.

cipher = AES.new(key, AES.MODE\_CBC, iv)

c2 = cipher.encrypt(Padding.pad(data, 16))

print("c2: {0}\n".format(binascii.hexlify(bytearray(c2))))

# d.

decrpted\_message = decryptRSA(c1, "private.pem")

print("decrypted c1 (sk) - {}".format(binascii.hexlify(bytearray(decrpted\_message))))

# Decrypt the ciphertext

cipher = AES.new(key, AES.MODE\_CBC, iv)

plaintext = cipher.decrypt(c2)

print("decrypted msg - {0}".format(Padding.unpad(plaintext, 16)))