**Lab 5: Asymmetric (Public) Key**



**Objective:** The key objective of this lab is to provide a practical introduction to public keyencryption, and with a focus on RSA and Elliptic Curve methods. This includes the creation of key pairs and in the signing process. As a part of this objective first you perform section c which is given below.

* **Web link (Weekly activities):** https://asecuritysite.com/esecurity/unit04
* **Video demo:** https://youtu.be/6T9bFA2nl3c
* **RSA Encryption**



**A.1** The following defines a public key that is used with PGP email encryption:

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v2

mQENBFTzi1ABCADIEWchOyqRQmU4AyQAMj2Pn68Sqo9lTPdPcItwo9LbTdv1YCFz

w3qLlp2RORMP+Kpdi92CIhdUYHDmZfHZ3IWTBgo9+y/Np9UJ6tNGocrgsq4xWz15

4vX4jJRddC7QySSh9UxDpRWf9sgqEv1pah136r95ZuyjC1EXnoNxdLJtx8PliCXc

hV/v4+KfOyzYh+HDJ4xP2bt1S07dkasYZ6cA7BHYi9k4xgEwxVvYtNjSPjTsQY5R

cTayXveGafuxmhSauZKiB/2TFErjEt49Y+p07tPTLX7bhMBVbUvojtt/JeUKV6vK

R82dmOd8seUvhwOHYB0JL+3S7PgFFsLo1NV5ABEBAAG0LkJpbGwgQnVjaGFuYW4g

KE5vbmUpIDx3LmJ1Y2hhbmFuQG5hcGllci5hYy51az6JATkEEwECACMFAlTzi1AC

GwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIXgAAKCRDsAFZRGtdPQi13B/9KHeFb

l1AxqbafFGRDEvx8UfPnEww4FFqWhcr8RLWyE8/COlUpB/5AS2yvojmbNFMGzURb

LGf/u1LVH0a+NHQu57u8Sv+g3bBthEPh4bKaEzBYRS/dYHOx3APFyIayfm78JVRF

zdeTOOf6PaXUTRx7iscCTkN8DUD3lg/465ZX5aH3HWFFX500JSPSt0/udqjoQuAr

WA5JqB//g2GfzZe1UzH5Dz3PBbJky8GiIfLm0OXSEIgAmpvc/9NjzAgjOW56n3Mu

sjVkibc+lljw+rOo97CfJMppmtcOvehvQv+KG0LZnpibiWVmM3vT7E6kRy4gEbDu

enHPDqhsvcqTDqaduQENBFTzi1ABCACzpJgZLK/sge2rMLURUQQ6l02UrS/GilGC

ofq3WPnDt5hEjarwMMwN65Pb0Dj0i7vnorhL+fdb/J8b8QTiyp7i03dZVhDahcQ5

8afvCjQtQstY8+K6kZFzQOBgyOS5rHAKHNSPFq45MlnPo5aaDvP7s9mdMILITvlb

CFhcLoC6Oqy+JoaHupJqHBqGc48/5NU4qbt6fB1AQ/H4M+6og4OozohgkQb80Hox

YbJV4sv4vYMULd+FKOg2RdGeNMM/aWdqYo90qb/W2aHCCyXmhGHEEuok9jbc8cr/

xrWL0gDwlWpad8RfQwyVU/VZ3Eg3OseL4SedEmwOO

cr15XDIs6dpABEBAAGJAR8E

GAECAAkFAlTzi1ACGwwACgkQ7ABWURrXT0KZTgf9FUpkh3wv7aC5M2wwdEjt0rDx

nj9kxH99hhuTX2EHXuNLH+SwLGHBq5O2sq3jfP+owEhs8/Ez0j1/fSKIqAdlz3mB

dbqWPjzPTY/m0It+wv3epOM75uWjD35PF0rKxxZmEf6SrjZD1sk0B9bRy2v9iWN9

9ZkuvcfH4vT++PognQLTUqNx0FGpD1agrG0lXSCtJWQXCXPfWdtbIdThBgzH4flZ

ssAIbCaBlQkzfbPvrMzdTIP+AXg6++K9SnO9N/FRPYzjUSEmpRp+ox31WymvczcU

RmyUquF+/zNnSBVgtY1rzwaYi05XfuxG0WHVHPTtRyJ5pF4HSqiuvk6Z/4z3bw==

=ZrP+

-----END PGP PUBLIC KEY BLOCK-----



 Using the following Web page, determine the owner of the key, and the ID on the key:

 https://asecuritysite.com/encryption/pgp1

By searching on-line, can you find the public key of three famous people, and view their key details, and can you discover some of the details of their keys (eg User ID, key encryption  method, key size, etc)?

 By searching on-line, what is an ASCII Armored Message?

1

**A.2** Bob has a private RSA key of:

MIICXAIBAAKBgQCwgjkeoyCXm9v6VBnUi5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnPAaDX3f2r4STZYYiqXGsH

CUBZcI90dvZf6YiEM5OY2jgsmqBjf2Xkp/8HgN/XDw/wD2+zebYGLLYtd2u3GXx9edqJ8kQcU9LaMH+ficFQyfq9UwTjQ

IDAQABAoGAD7L1a6Ess+9b6G70gTANWkKJpshVZDGb63mxKRepaJEX8sRJEqLqOYDNsC+pkKO8IsfHreh4vrp9bsZuECr

B1OHSjwDB0S/fm3KEWbsaaXDUAu0dQg/JBMXAKzeATreoIYJItYgwzrJ++fuquKabAZumvOnWJyBIs2z103kDz2ECQQDn

n3JpHirmgVdf81yBbAJaXBXNIPzOcCth1zwFAs4EvrE35n2HvUQuRhy3ahUKXsKX/bGvWzmC2O6kbLTFEygVAkEAwxXZn

PkaAY2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC71FbNIgHBg5srsUyDj3OsloLmDVjmQJAIy7qLyOA+s

Cc6BtMavBgLx+bxCwFmsoZHOSX3l79smTRAJ/HY64RREIsLIQ1q/yW7IWBzxQ5WTHgliNZFjKBvQJBAL3t/vCJwRz0Ebs

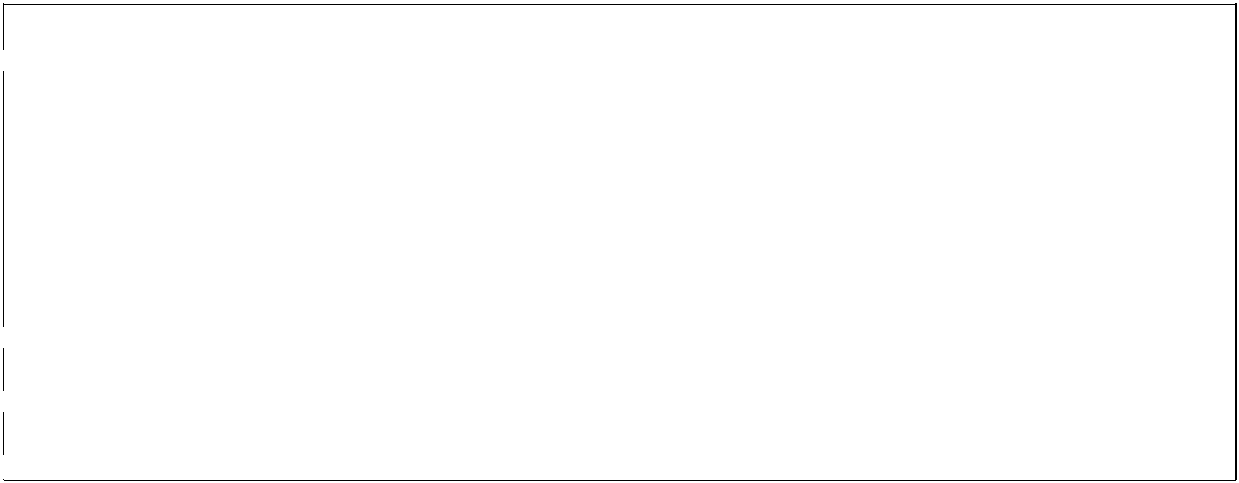
5FaB/8UwhhsrbtXlGdnkOjIGsmV0vHSf6poHqUiay/DV88pvhN11ZG8zHpeUhnaQccJ9ekzkCQDHHG9LYCOqTgsyYms//

cW4sv2nuOE1UezTjUFeqOlsgO+WN96b/M5gnv45/Z3xZxzJ4HOCJ/NRwxNOtEUkw+zY=

And receives a ciphertext message of:

Pob7AQZZSml618nMwTpx3V74N45x/rTimUQeTl0yHq8F0dsekZgOT385Jls1HUzWCx6ZRFPFMJ1RNYR2Yh7AkQtFLVx9l YDfb/Q+SkinBIBX59ER3/fDhrVKxIN4S6h2QmMSRblh4KdVhyY6cOxu+g48Jh7TkQ2Ig93/nCpAnYQ=

Using the following code:



from Crypto.PublicKey import RSA

from Crypto.Util import asn1

from base64 import b64decode

msg="Pob7AQZZSml618nMwTpx3V74N45x/rTimUQeTl0yHq8F0dsekZgOT385Jls1HUzWCx6ZRFPFMJ1RNYR2Yh7AkQtF

LVx9lYDfb/Q+SkinBIBX59ER3/fDhrVKxIN4S6h2QmMSRblh4KdVhyY6cOxu+g48Jh7TkQ2Ig93/nCpAnYQ="

privatekey =

'MIICXAIBAAKBgQCwgjkeoyCXm9v6VBnUi5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnPAaDX3f2r4STZYYiqXGs

HCUBZcI90dvZf6YiEM5OY2jgsmqBjf2Xkp/8HgN/XDw/wD2+zebYGLLYtd2u3GXx9edqJ8kQcU9LaMH+ficFQyfq9UwTj

QIDAQABAoGAD7L1a6Ess+9b6G70gTANWkKJpshVZDGb63mxKRepaJEX8sRJEqLqOYDNsC+pkKO8IsfHreh4vrp9bsZuEC

rB1OHSjwDB0S/fm3KEWbsaaXDUAu0dQg/JBMXAKzeATreoIYJItYgwzrJ++fuquKabAZumvOnWJyBIs2z103kDz2ECQQD

nn3JpHirmgVdf81yBbAJaXBXNIPzOcCth1zwFAs4EvrE35n2HvUQuRhy3ahUKXsKX/bGvWzmC2O6kbLTFEygVAkEAwxXZ

nPkaAY2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC71FbNIgHBg5srsUyDj3OsloLmDVjmQJAIy7qLyOA+

sCc6BtMavBgLx+bxCwFmsoZHOSX3l79smTRAJ/HY64RREIsLIQ1q/yW7IWBzxQ5WTHgliNZFjKBvQJBAL3t/vCJwRz0Eb

s5FaB/8UwhhsrbtXlGdnkOjIGsmV0vHSf6poHqUiay/DV88pvhN11ZG8zHpeUhnaQccJ9ekzkCQDHHG9LYCOqTgsyYms/

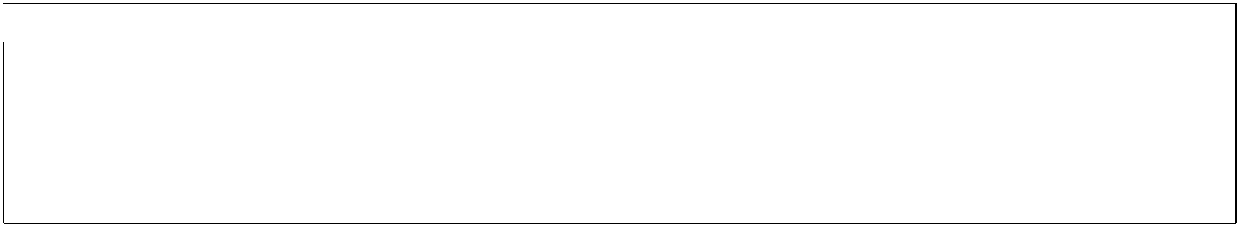
/cW4sv2nuOE1UezTjUFeqOlsgO+WN96b/M5gnv45/Z3xZxzJ4HOCJ/NRwxNOtEUkw+zY='

keyDER = b64decode(privatekey)

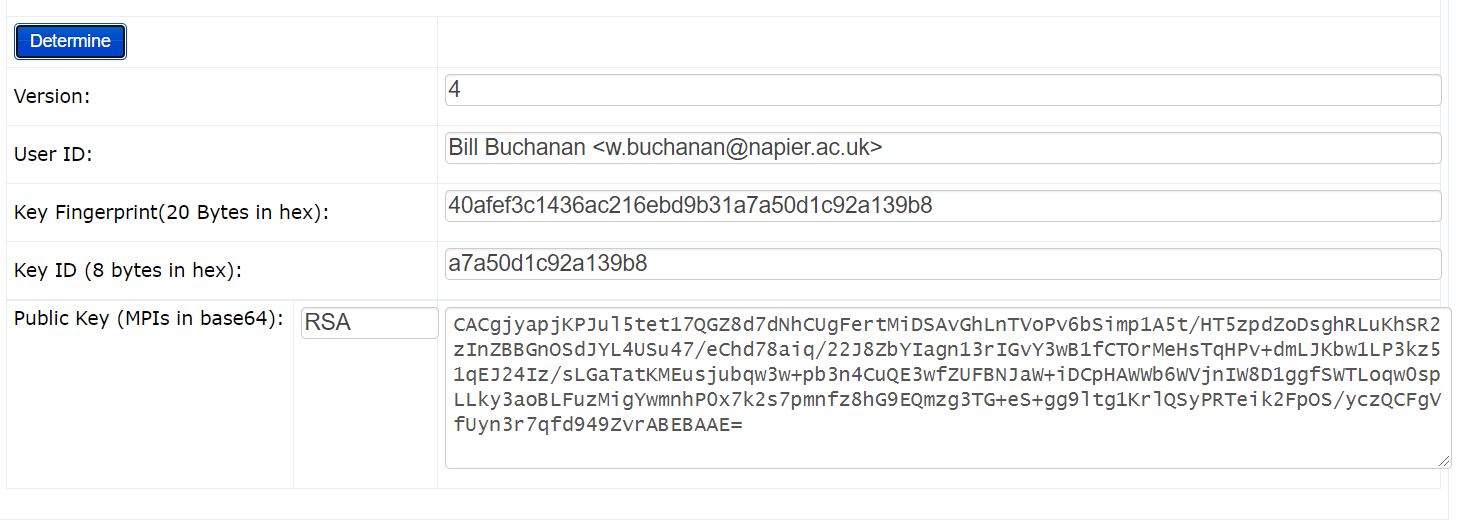
keys = RSA.importKey(keyDER)

dmsg = keys.decrypt(b64decode(msg))

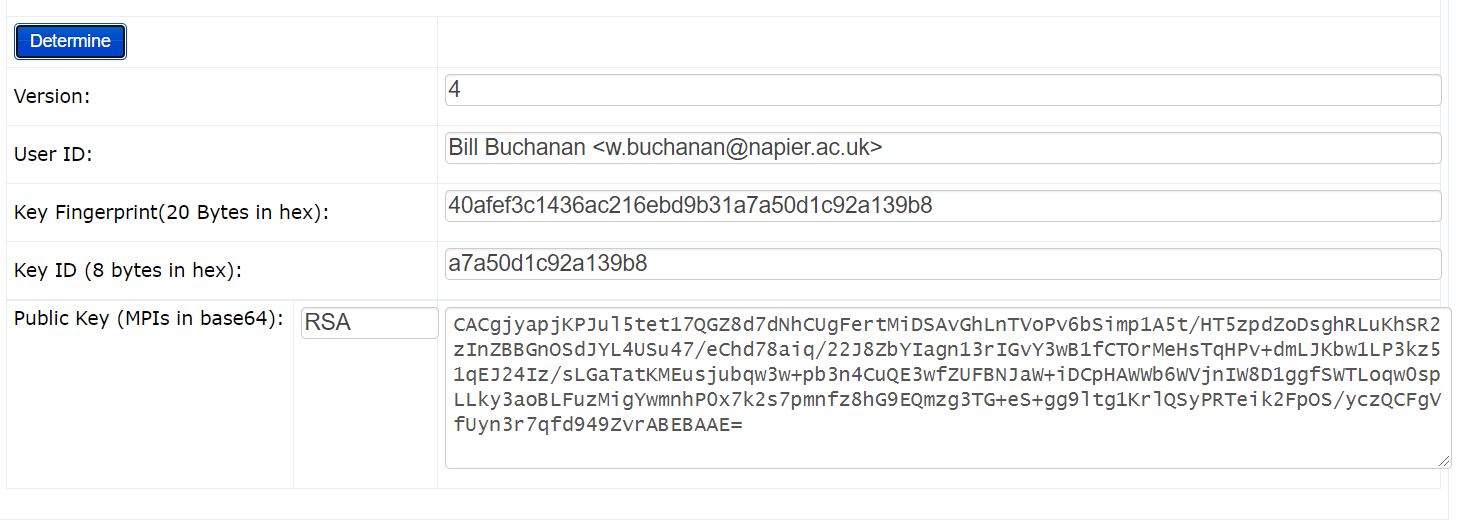
print dmsg



 What is the plaintext message that Bob has been sent?



A1:





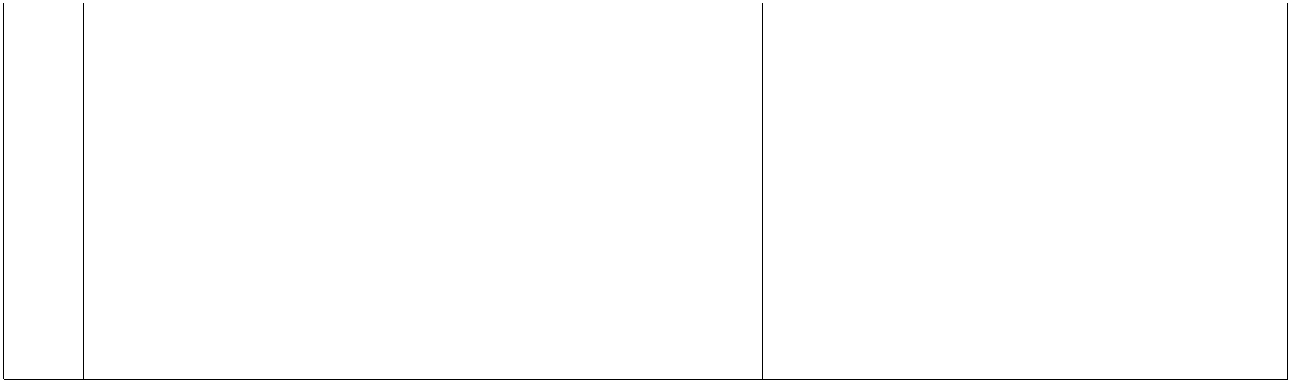
Key:



* **OpenSSL (RSA)**



We will use OpenSSL to perform the following:

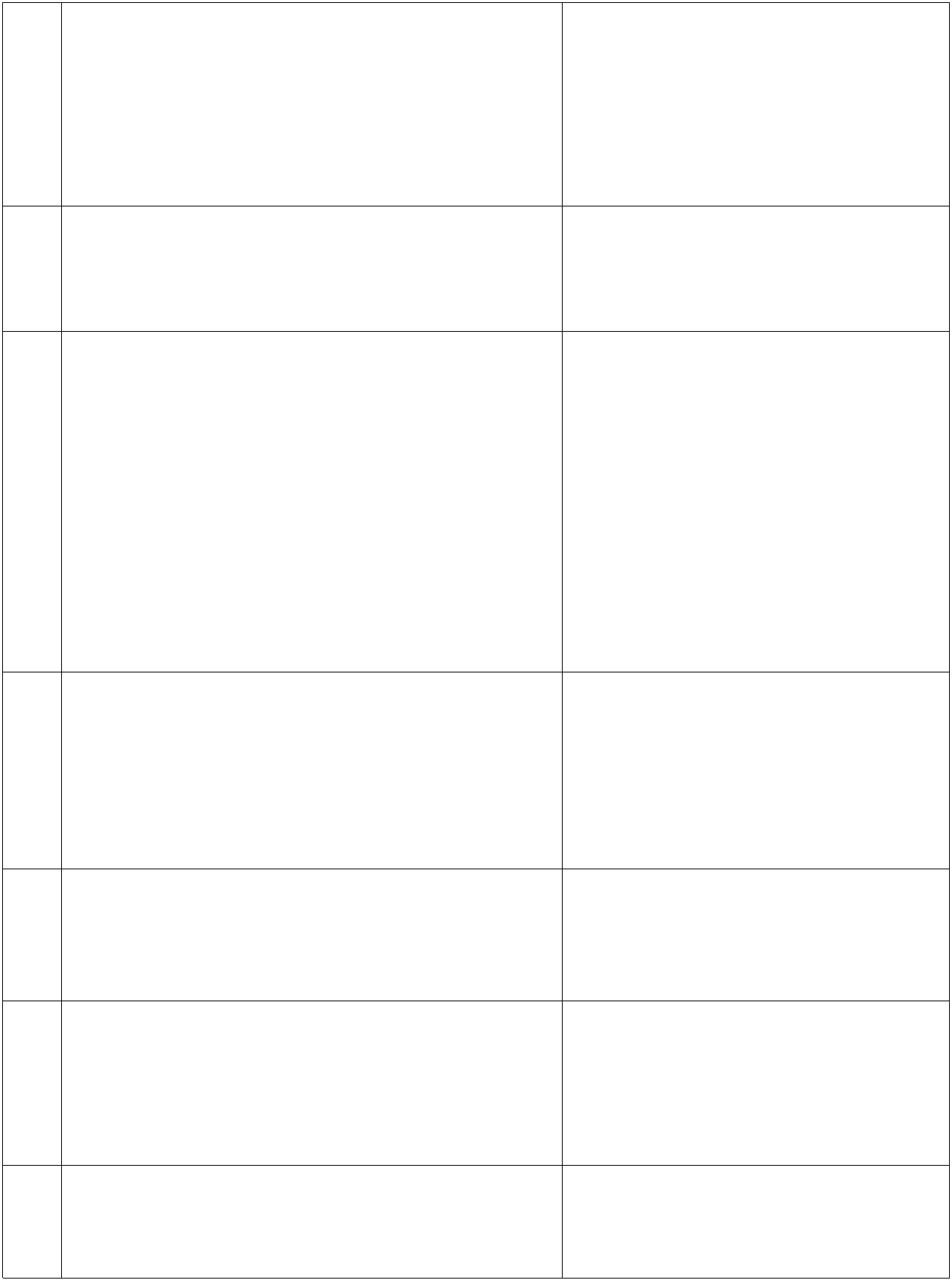


|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **B.1** | First we need to generate a key pair with: | What is the type of public key method |
|  | openssl genrsa -out private.pem 1024 | used: |
|  |  |
|  |  | How long is the default key: |

This file contains both the public and the private key.

How long did it take to generate a 1,024 bit key?

2



**B.2** Use following command to view the output file:

cat private.pem

**B.3** Next we view the RSA key pair:

openssl rsa -in private.pem -text

**B.4** Let’s now secure the encrypted key with 3-DES:

openssl rsa -in private.pem -des3 -out key3des.pem

**B.5** Next we will export the public key:

openssl rsa -in private.pem -out public.pem -outform PEM -pubout

**B.6** Now create a file named “myfile.txt” and put a message into it. Next encrypt it with your public key:

openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin

**B.7** And then decrypt with your private key:

openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt

Use the following command to view the keys:

cat private.pem

What can be observed at the start and end of the file:

Which are the attributes of the key shown:

Which number format is used to display the information on the attributes:

Why should you have a password on the usage of your private key?

View the output key. What does the header and footer of the file identify?

What are the contents of decrypted.txt

On your VM, go into the ~/.ssh folder. Now generate your SSH keys:

3

ssh-keygen -t rsa -C "your email address"

The public key should look like this:

ssh-rsa

AAAAB3NzaC1yc2EAAAADAQABAAABAQDLrriuNYTyWuC1IW7H6yea3hMV+rm029m2f6IddtlImHrOXjNwYyt4Elkkc7AzO

y899C3gpx0kJK45k/CLbPnrHvkLvtQ0AbzWEQpOKxI+tW06PcqJNmTB8ITRLqIFQ++ZanjHWMw2Odew/514y1dQ8dccCO

uzeGhL2Lq9dtfhSxx+1cBLcyoSh/lQcs1HpXtpwU8JMxWJl409RQOVn3gOusp/P/0R8mz/RWkmsFsyDRLgQK+xtQxbpbo

dpnz5lIOPWn5LnT0si7eHmL3WikTyg+QLZ3D3m44NCeNb+bOJbfaQ2ZB+lv8C3OxylxSp2sxzPZMbrZWqGSLPjgDiFIBL

w.buchanan@napier.ac.uk



 View the private key. Outline its format?

On your Ubuntu instance setup your new keys for ssh:

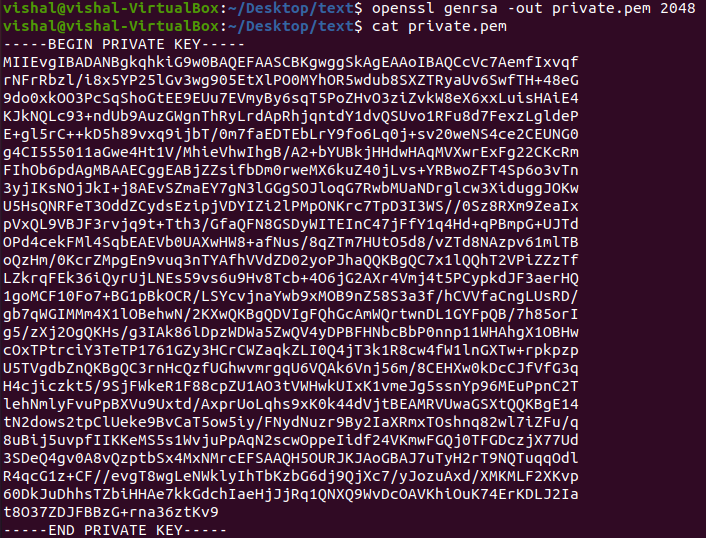
ssh-add ~/.ssh/id\_git

Now create a Github account and upload your public key to Github (select Settings-> **New** **SSH key** or **Add SSH key**). Create a new repository on your GitHub site, and add a new fileto it. Next go to your Ubuntu instance and see if you can clone of a new directory:

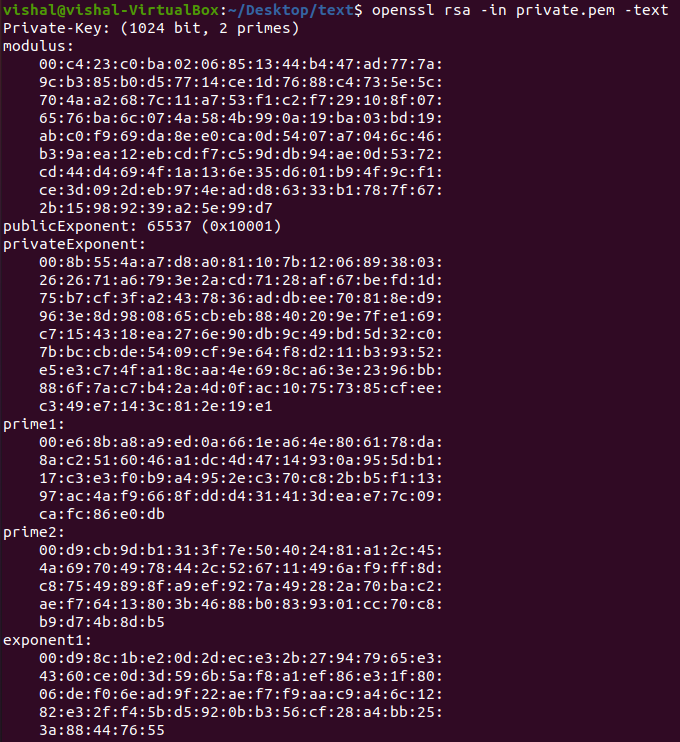
git clone ssh://git@github.com/<user>/<repository name>.git

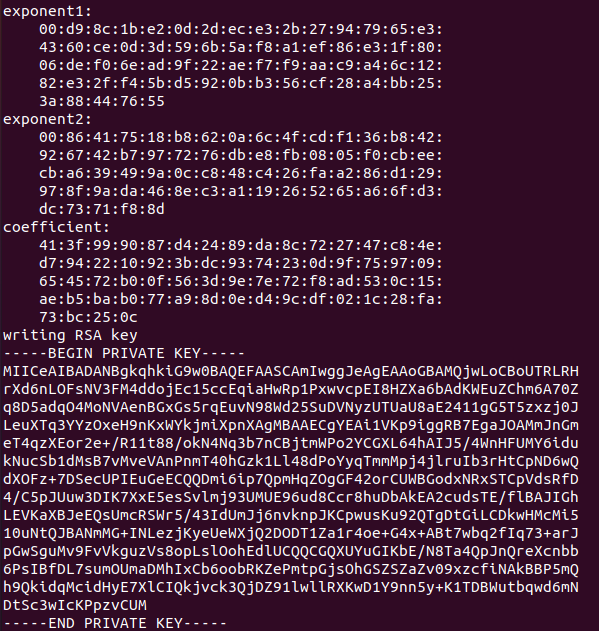
If this doesn’t work, try the https connection that is defined on GitHub.

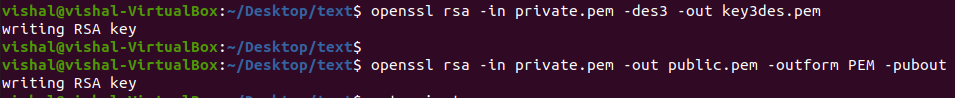






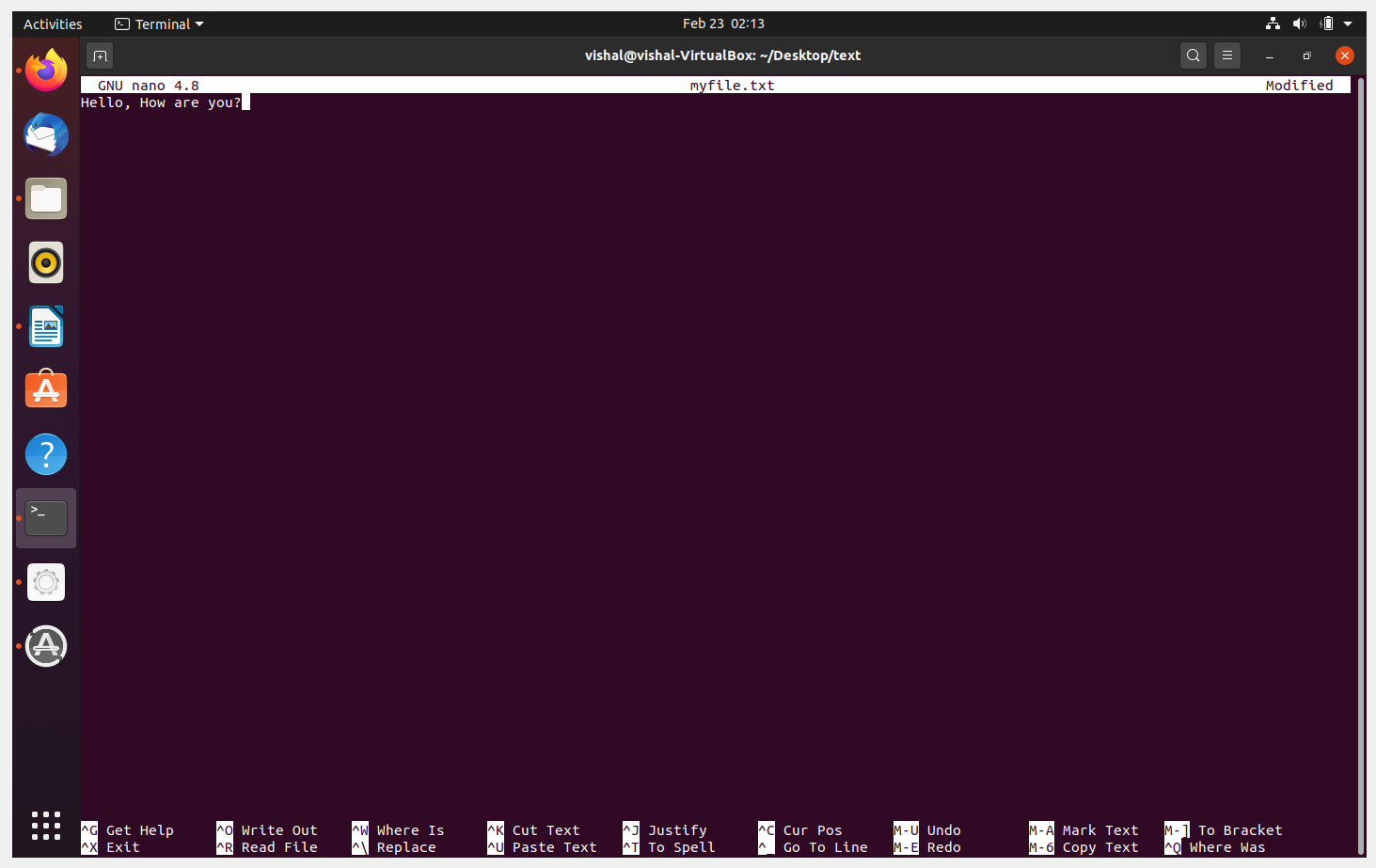


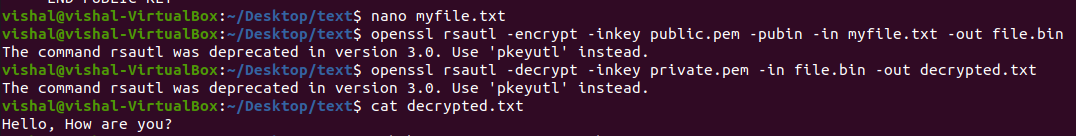


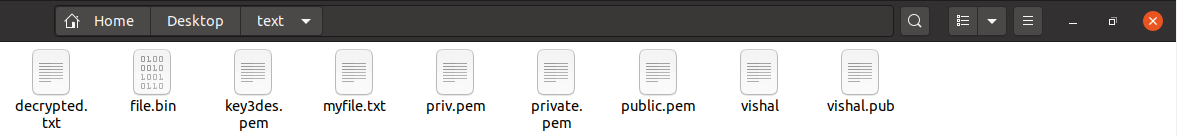


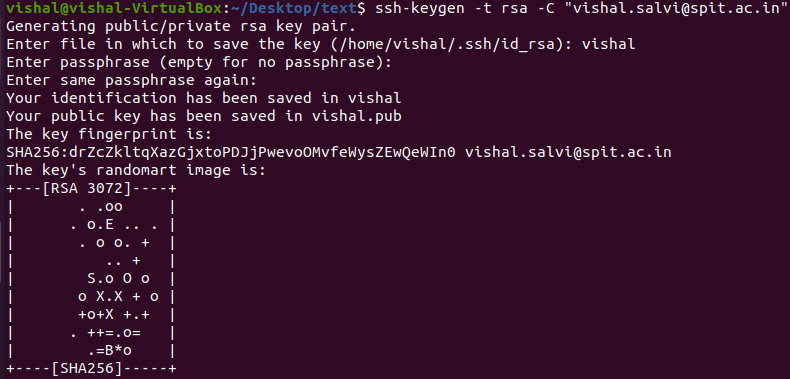








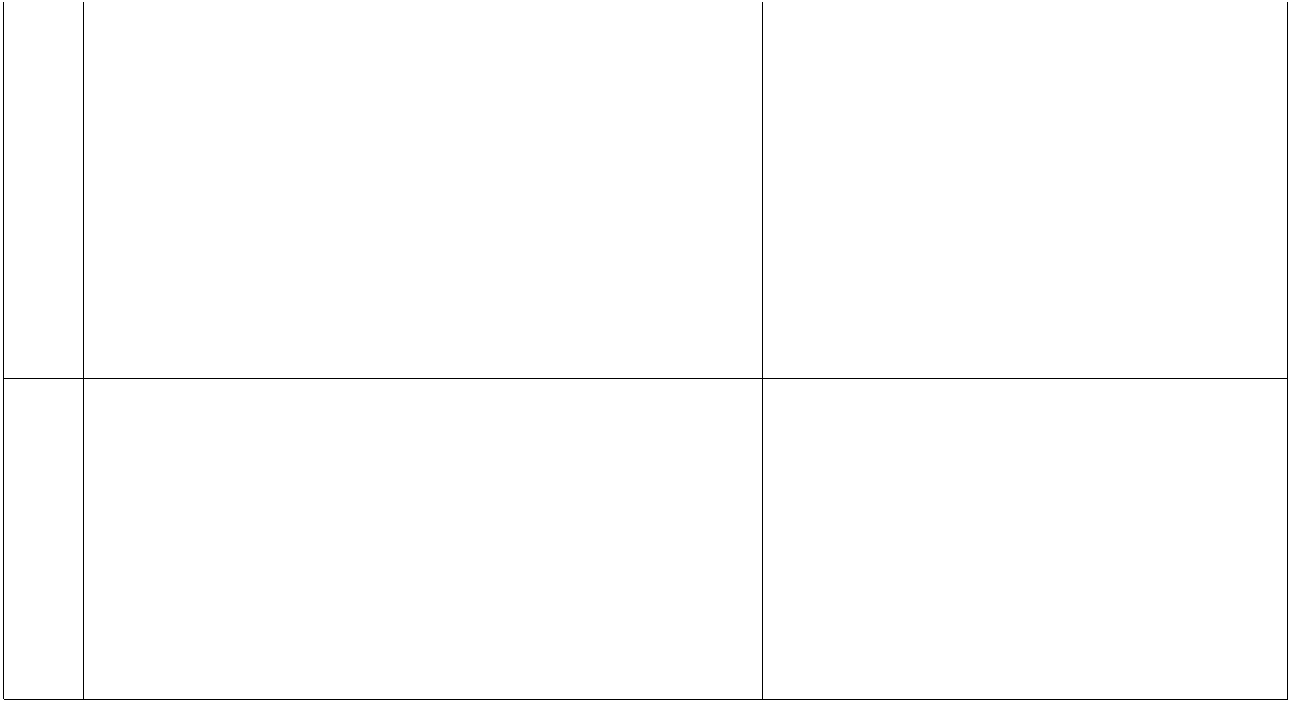




* **OpenSSL (ECC)**



Elliptic Curve Cryptography (ECC) is now used extensively within public key encryption, including with Bitcoin, Ethereum, Tor, and many IoT applications. In this part of the lab we will use OpenSSL to create a key pair. For this we generate a random 256-bit private key (*priv*), and then generate a public key point (*priv* multiplied by G), using a generator (*G*), and which is a generator point on the selected elliptic curve.



|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **C.1** | First we need to generate a private key with: | Can you view your key? **YES** |

openssl ecparam -name secp256k1 -genkey -out priv.pem

The file will only contain the private key (and should have 256 bits).

Now use “cat priv.pem” to view your key.

**C.2** We can view the details of the ECC parameters used with:

openssl ecparam -in priv.pem -text - param\_enc explicit -noout

Outline these values:

Prime (last two bytes): **fc:2f**

A: **0**

B: **7**

Generator (last two bytes):  **d4:b8**

Order last two bytes: **41:41**



**C.3** Now generate your public key based on your private key with:

How many bits and bytes does your private key have: **256 bits**

openssl ec -in priv.pem -text -noout

How many bit and bytes does your

public key have (Note the 04 is not

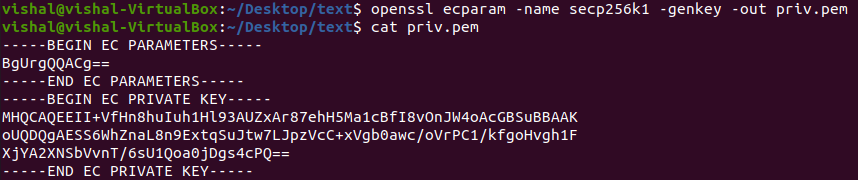
part of the elliptic curve point):

What is the ECC method that you

have used?

If you want to see an example of ECC, try here: <https://asecuritysite.com/encryption/ecc>

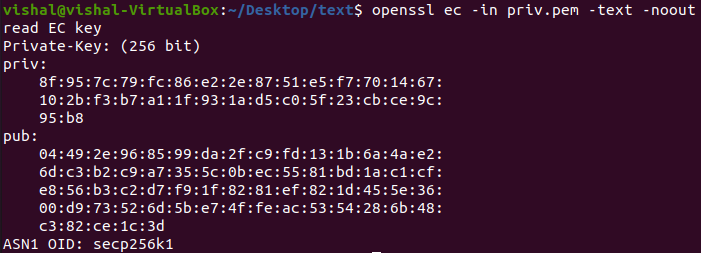
C1.



C2.



C3.



* **Elliptic Curve Encryption**



**D.1** In the following Bob and Alice create elliptic curve key pairs. Bob can encrypt a message for Alice with her public key, and she can decrypt with her private key. Copy and paste the program from here:

https://asecuritysite.com/encryption/elc

Code used:



import OpenSSL

import pyelliptic

secretkey="password"

test="Test123"

alice = pyelliptic.ECC()

bob = pyelliptic.ECC()

print "++++Keys++++"

print "Bob's private key: "+bob.get\_privkey().encode('hex')

 print "Bob's public key: "+bob.get\_pubkey().encode('hex')

print

print "Alice's private key: "+alice.get\_privkey().encode('hex')

 print "Alice's public key: "+alice.get\_pubkey().encode('hex')

 ciphertext = alice.encrypt(test, bob.get\_pubkey())

 print "\n++++Encryption++++"

 print "Cipher: "+ciphertext.encode('hex')

 print "Decrypt: "+bob.decrypt(ciphertext)

signature = bob.sign("Alice")

print

print "Bob verified: "+ str(pyelliptic.ECC(pubkey=bob.get\_pubkey()).verify  (signature, "Alice"))



For a message of “Hello. Alice”, what is the ciphertext sent (just include the first four  characters):

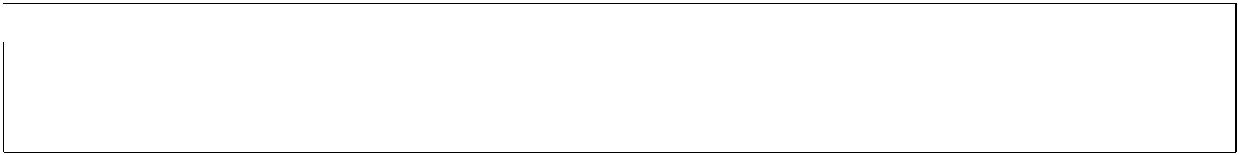
5



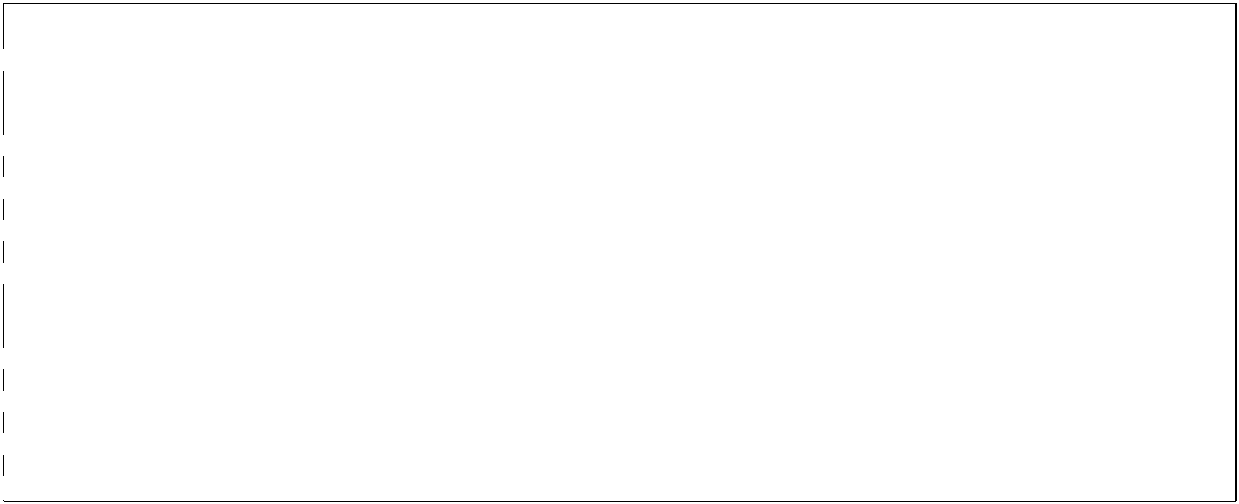
 How is the signature used in this example?

**D.2** Let’s say we create an elliptic curve with *y*2 = *x*3 + 7, and with a prime number of 89, generate the first five (*x*,*y*) points for the finite field elliptic curve. You can use the Python code at the following to generate them:

https://asecuritysite.com/encryption/ecc\_points  First five points: (14, 9) (15, 0) (16, 3)



**D.3** Elliptic curve methods are often used to sign messages, and where Bob will sign a message with his private key, and where Alice can prove that he has signed it by using his public key. With ECC, we can use ECDSA, and which was used in the first version of Bitcoin. Enter the following code:



from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST521p,SECP256k1 import base64

 import sys

msg="Hello"

type = 1

 cur=NIST192p

 sk = SigningKey.generate(curve=cur)

 vk = sk.get\_verifying\_key()

signature = sk.sign(msg)

print "Message:\t",msg

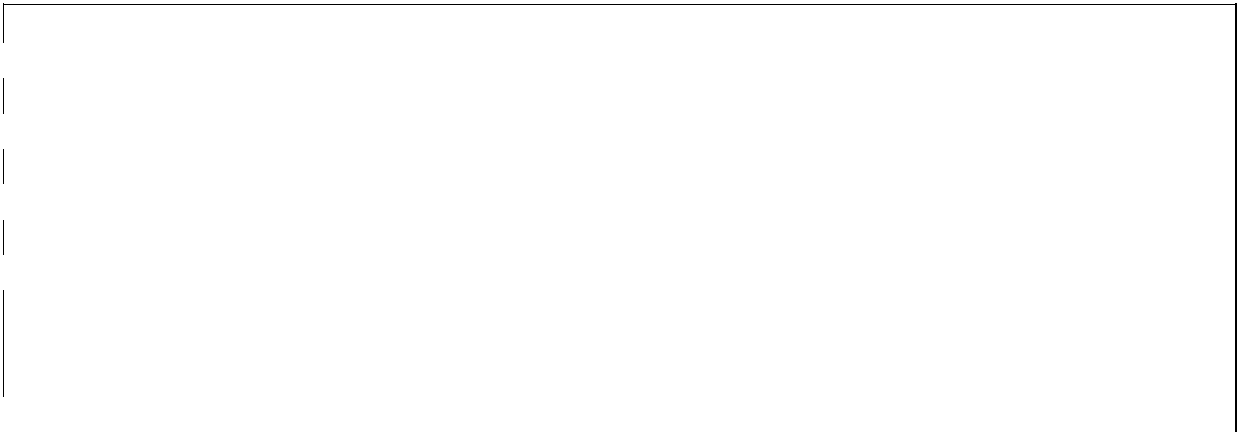
print "Type:\t\t",cur.name

 print "========================="

 print "Signature:\t",base64.b64encode(signature)

 print "========================="

 print "Signatures match:\t",vk.verify(signature, msg)



What are the signatures (you only need to note the first four characters) for a message of  “Bob”, for the curves of NIST192p, NIST521p and SECP256k1:

 NIST192p: **ANXA**

 NIST521p: **ANHb**

 SECP256k1: **LWT7**

By searching on the Internet, can you find in which application areas that SECP256k1 is  used?

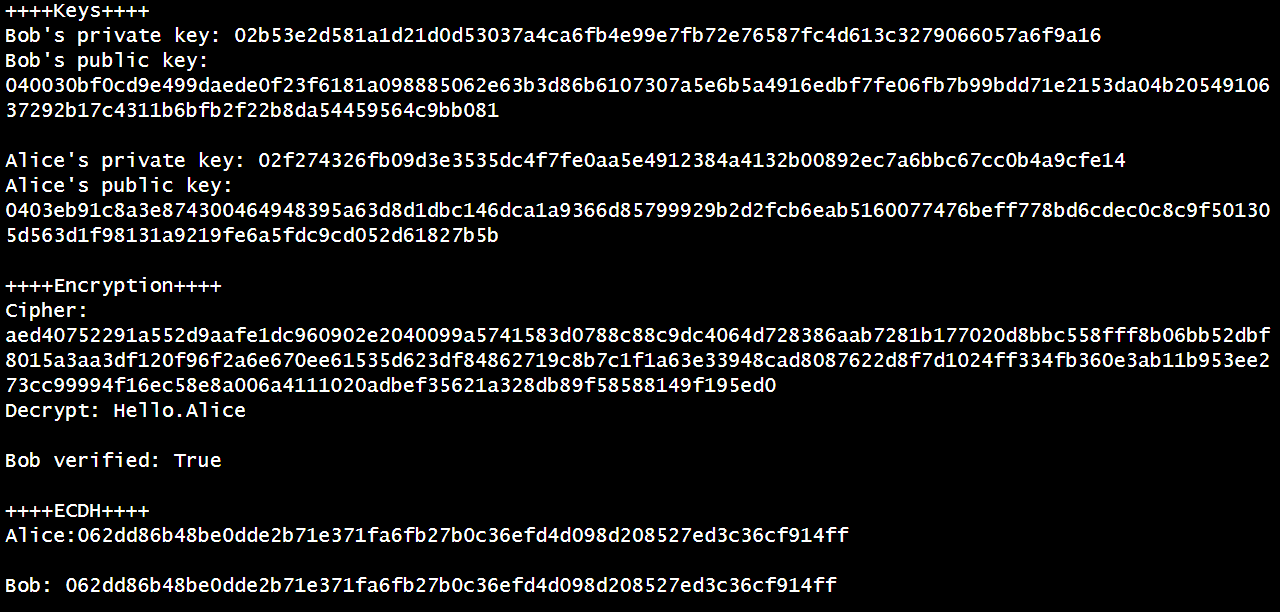
**SECP256K1 is used Bitcoin**

6

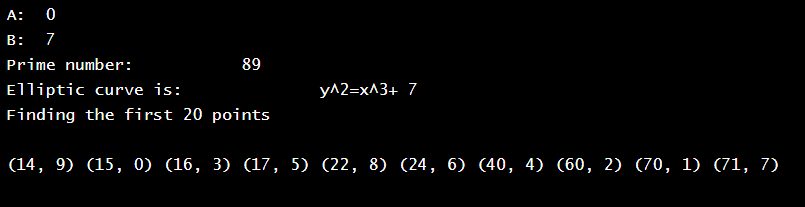


 What do you observe from the different hash signatures from the elliptic curve methods?

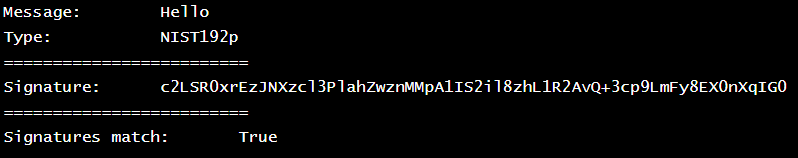
D1

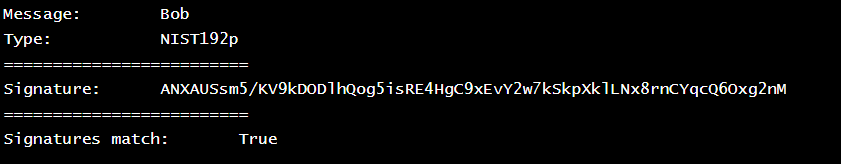


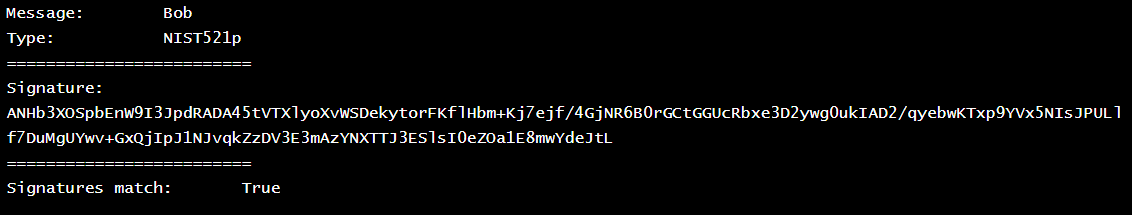
D2

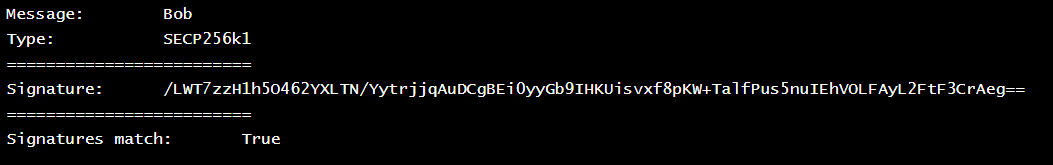


D3









**Conclusion:**

1. ECC serves as a feasible alternative to the existing and traditional algorithms and provides various advantages in terms of security, speed, performance, and speed.
2. The ability of ECC to use complex mathematical algorithms for data protection makes many researchers in the field of encryption anticipate the future of ECC to be bright.

**Curves over prime fields**

The general equation of an elliptic curve is simplified over prime field GF(p) (characteristic p>3):

y 2 = x3 + ax + b

* **RSA**



**E.1** We will follow a basic RSA process. If you are struggling here, have a look at the following page:

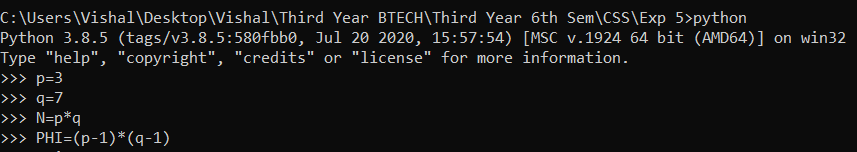
https://asecuritysite.com/encryption/rsa

First, pick two prime numbers:



 p=

q=



Now calculate N (p.q) and PHI [(p-1).(q-1)]:



 N=

PHI =

Now pick a value of *e* which does not share a factor with PHI [gcd(PHI,e)=1]:



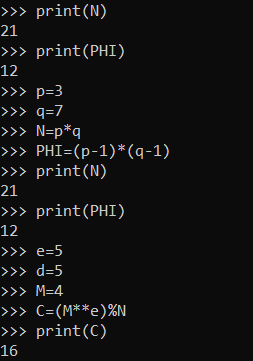
 *e*=

Now select a value of d, so that (e.d) (mod PHI) = 1:

[Note: You can use this page to find *d*: https://asecuritysite.com/encryption/inversemod]



 *d*=



Now for a message of M=5, calculate the cipher as:

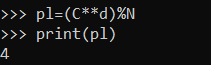


 C = M*e* (mod N) =

Now decrypt your ciphertext with:



 M = Cd (mod N) =



Did you get the value of your message back (M=5)? If not, you have made a mistake, so go back and check.

Now run the following code and prove that the decrypted cipher is the same as the message:



p=11

q=3

N=p\*q

PHI=(p-1)\*(q-1)

e=3

 for d in range(1,100):

7

if ((e\*d % PHI)==1): break

print e,N

print d,N

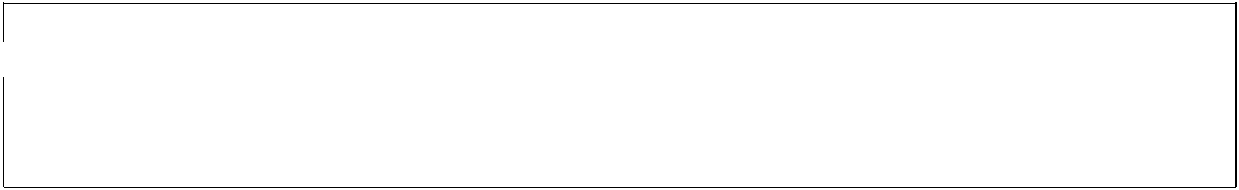
M=4

cipher = M\*\*e % N

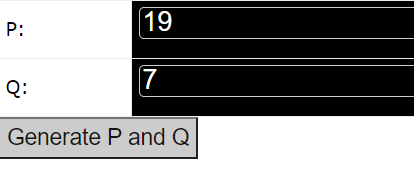
print cipher

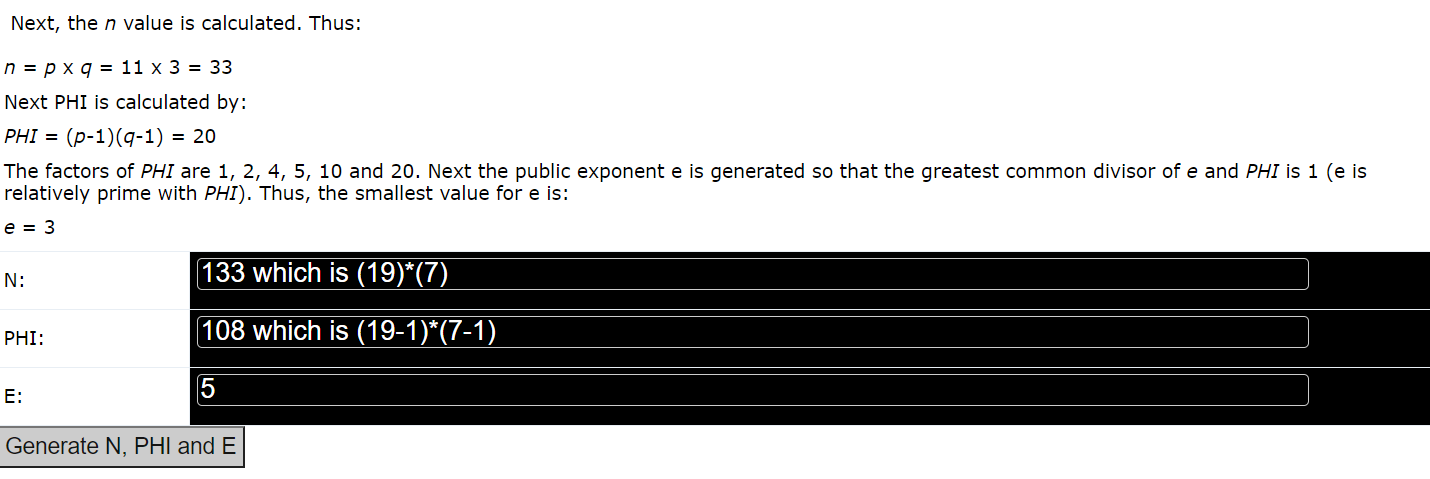
message = cipher\*\*d % N

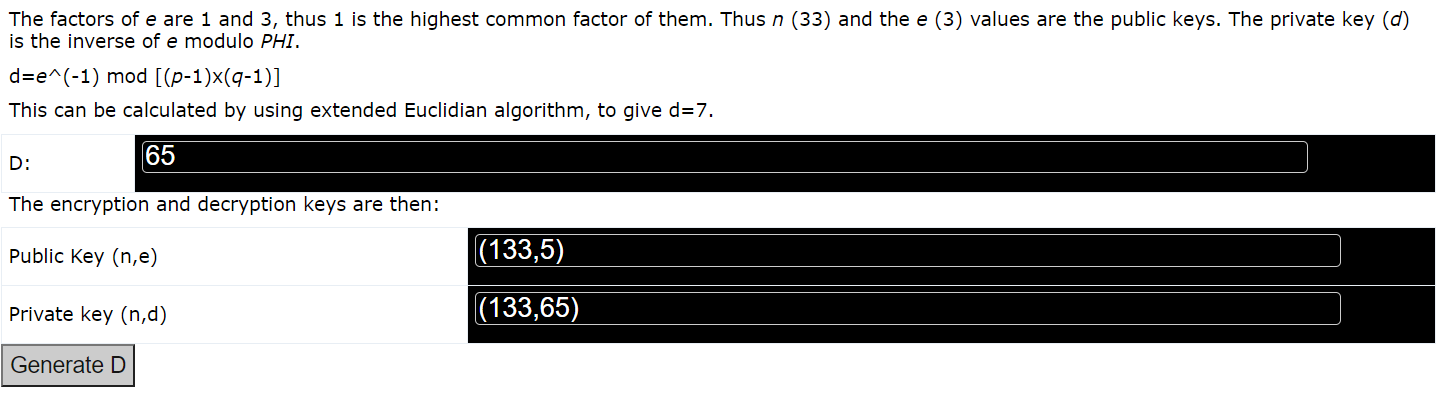
print message

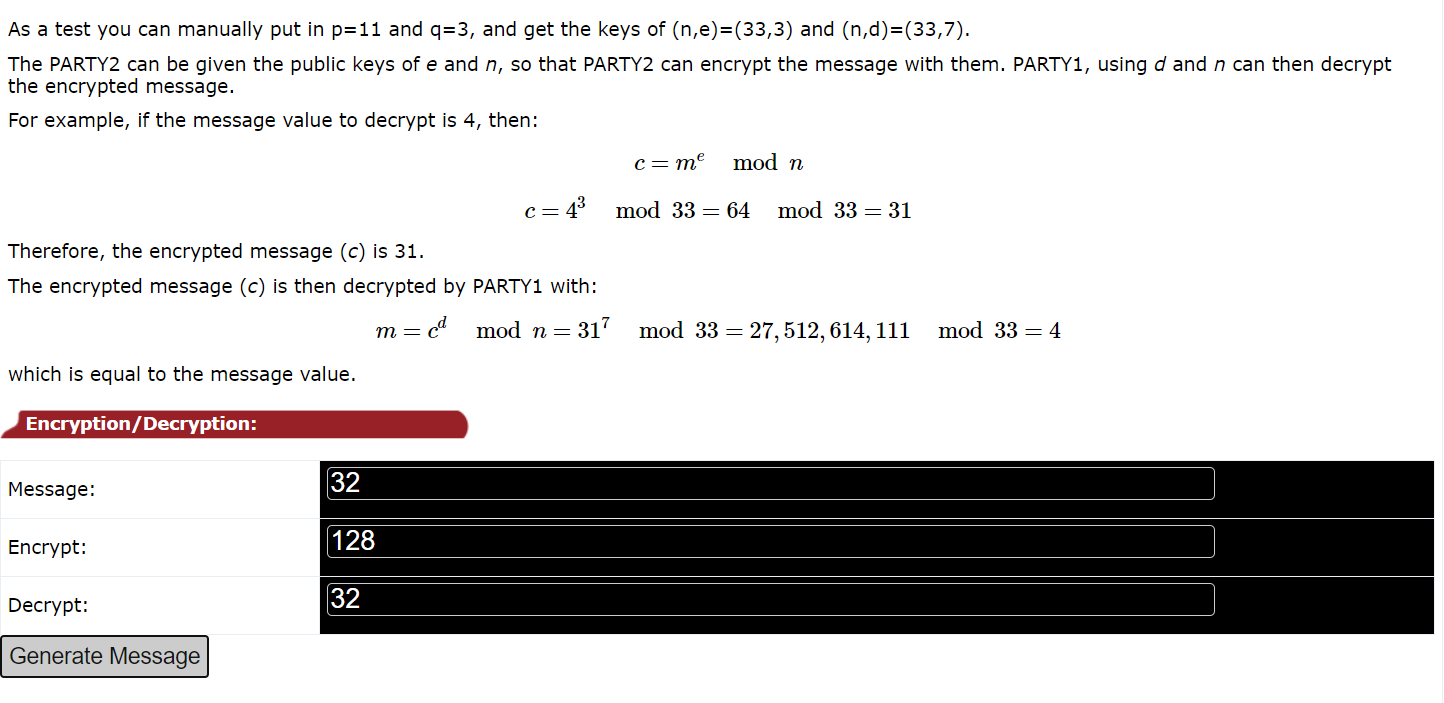


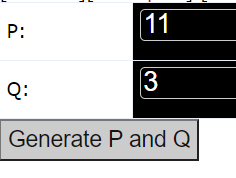
Select three more examples with different values of p and q, and then select e in order to  make sure that the cipher will work:

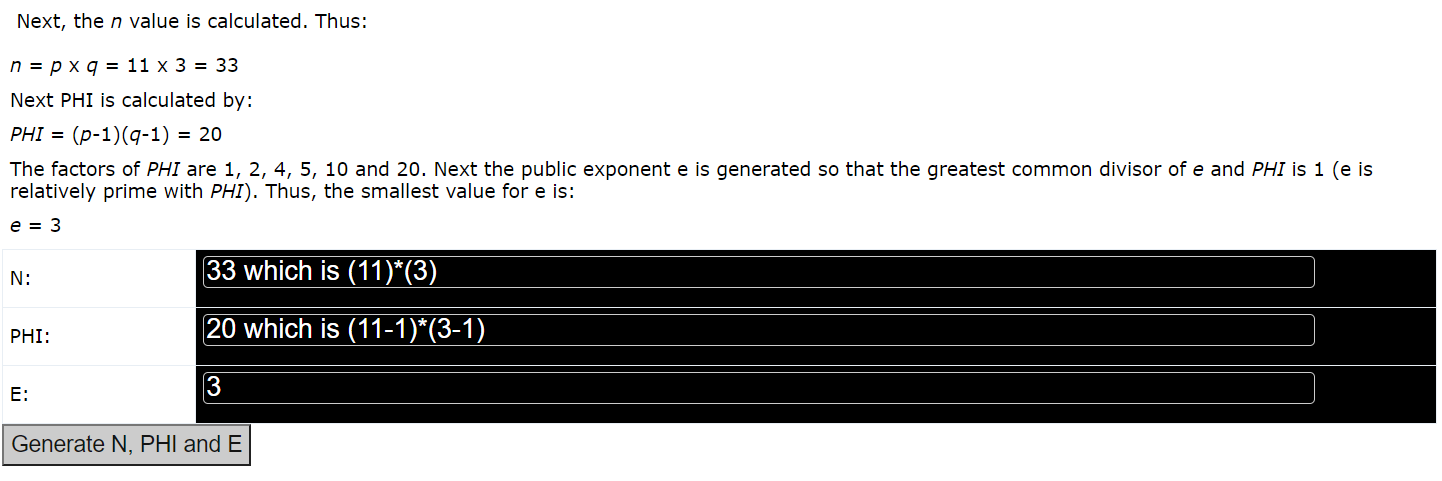


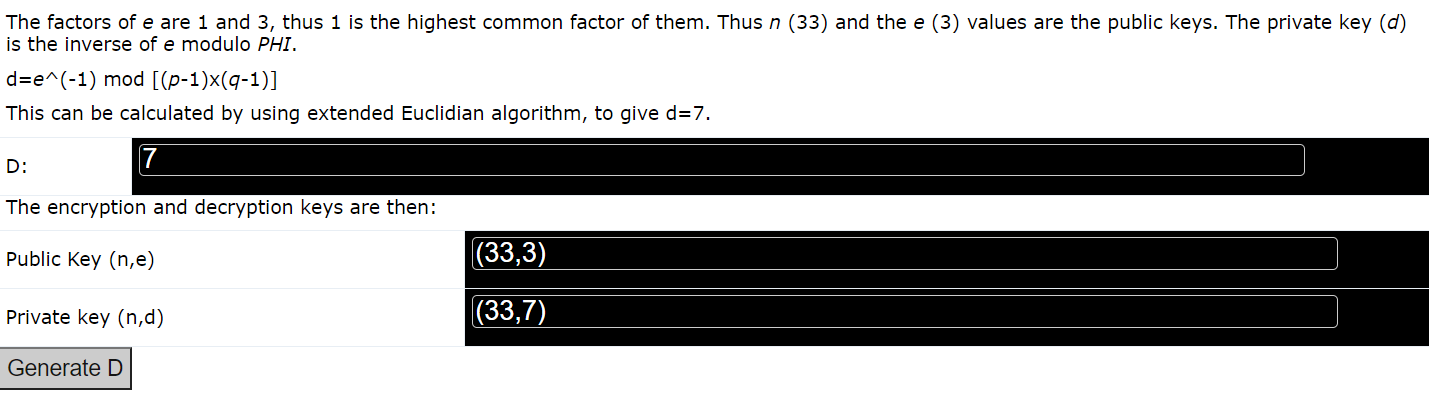


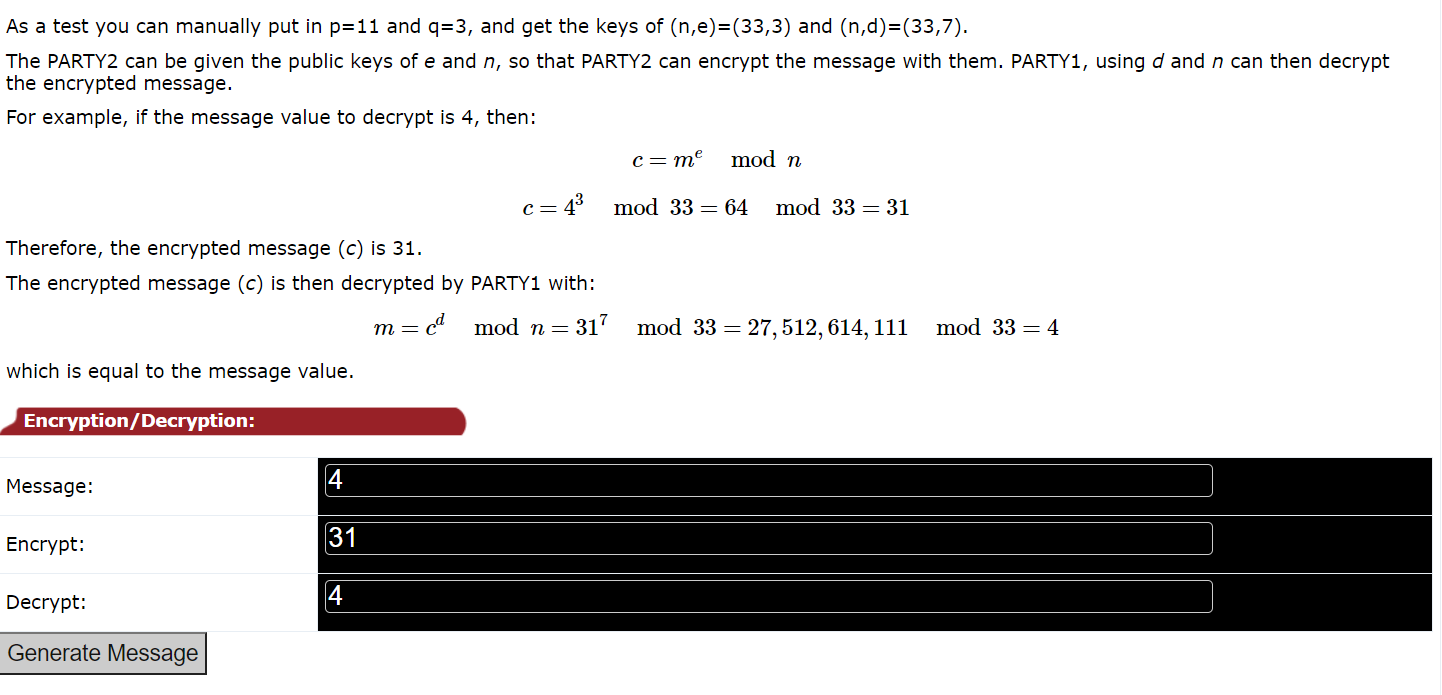












**E.2** In the RSA method, we have a value of e, and then determine d from (d.e) (mod PHI)=1. But how do we use code to determine d? Well we can use the Euclidean algorithm. The code for this is given at:

https://asecuritysite.com/encryption/inversemod

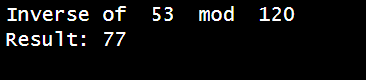
Using the code, can you determine the following:

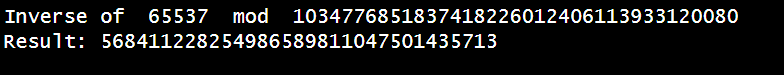


**Inverse of 53 (mod 120)** =

 **Inverse of 65537 (mod 1034776851837418226012406113933120080)** =

Using this code, can you now create an RSA program where the user enters the values of p, q, and e, and the program determines (e,N) and (d,N)?





**Conclusion:**

1. ECC serves as a feasible alternative to the existing and traditional algorithms and provides various advantages in terms of security, speed, performance, and speed.
2. The ability of ECC to use complex mathematical algorithms for data protection makes many researchers in the field of encryption anticipate the future of ECC to be bright.