


# Lab 4: Asymmetric (Public) Key

**Objective:** The key objective of this lab is to provide a practical introduction to public key encryption, and with a focus on RSA and Elliptic Curve methods. This includes the creation of key pairs and in the signing process.

 **Web link (Weekly activities):** <https://asecuritysite.com/eseconomy/unit04>

 **Video demo:** <https://youtu.be/6T9bFA2nl3c>

## A 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

mQENBFTzi1ABCADIEWch0yqRQmU4AyQAMj2Pn68Sqo91TPdPcItwo9LbTdv1YCFz
w3qLlp2RORMP+kpdi92CIhduYHDMZFHZ3IWTBgo9+y/Np9UJ6tNGocrgsq4xwz15
4vx4jJRddC7QySSh9UXDpRWF9sgqEv1pah136r95ZuyjC1EXnoNxdLJtx8PlICXC
hV/v4+KfOyzYh+HDJ4xP2bt1S07dkasYZ6cA7BHYi9k4xgEwxvVytNjSPjTsQY5R
CTayXveGafuxmhSauZKiB/2TFerjEt49Y+p07tPTLX7bhMBVbUvojtT/JeUKV6vK
R82dmOd8seUvhwOHYB0JL+3S7PgFFsLo1NV5ABEBAAG0LJpbGwgQnVjaGFuYW4g
KE5vbmUpIDx3LmJ1Y2hhbmFuQG5hcGllci5hYy51az6JATKEEwECACMFA1Tzi1AC
GwMHcWkiBwMCAQYVCAIJCgSEFGIDAQIEAQIXgAAKCRDsAFZRGtdPQi13B/9KHeFb
l1AxqbafFGRDEVx8UfPnEww4FFqWhcr8RLWyE8/COlUpB/5AS2yvojmbNFMGzURb
LGF/u1LVH0a+NHQu57u8Sv+g3bbthEPH4bkaEzBYRS/dYHOx3APFyIayfm78JVRf
zdeTOOf6PaxUTRx7isCCTkN8DUD3lg/465ZX5aH3HwFFX500JSPSt0/udqjoQuAr
WA5JqB//g2GfzZe1UzH5Dz3PBbJky8GiIfLm0OXSEIqAmpvc/9NjzAgjOW56n3Mu
sjvkiBc+lljw+roo97CfJmPmtcOvehvQv+KG0LZnpibiWvmM3vT7E6kRy4gEbdu
enHPDqhsvcqTQaduQENBFTzi1ABCACzpJgZLK/sge2rMLURUQQ6l02Urs/GilGC
ofq3WPndt5hEjarwMMWn65Pb0Dj0i7vnorhL+fdb/J8b8Qtiyp7i03dzVhdahcQ5
8afvCjQtQstY8+K6kZfZQOBgyOS5rHAKHNSPFq45M1nPo5aaDvP7s9mdMILITv1b
CFhCLoC60qy+JoahupJqHBqGc48/5NU4qbt6fB1AQ/H4M+6og40ozohgkQb80Hox
YbJV4sv4vYMULD+FKOG2RdGenMM/awdqYo90qb/W2aHCCyXmhGHEEuok9jbc8cr/
xrWL0gdWlwpad8RfQwyVU/VZ3Eg3OseL4SedEmw00
cr15XDI56dpABEBAAGJAR8E
GAECAAKFA1Tzi1ACGwwACqKQ7ABWURrXT0KZTgf9FUPkh3wv7ac5M2wwdEjtOrDx
nj9KxH99hhuTX2EHXUNLH+SwLGHBq502sq3jfp+owEhs8/Ez0j1/fSKIqAdl3mB
dbqWPjzPTY/m0It+vv3epOM75uWjD35PF0rKxxZmEf6SrjZD1sk0B9bry2v9iWN9
9ZkuvCFH4vt++PognQLTUqNxFGpD1agrG0lXSctJWQXCXpfdwtbIdThBgZ4f1Z
ssAIBCaB1QkzfbPvrMzdTIP+AXg6++K9Sn09N/FRPYzjUSEmpRp+ox31WymvcZCU
RmyUquF+/zNnSBVgtY1rzwai05XfuxG0WHVHPTtRYJ5pF4HSqiuvk6Z/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?

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

```
MIICXAIBAAKBgQCWgjkEoyCxm9v6VBnUi5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnPAaDX3f2r4STZYYiqXGSH
CUBZcI90dvZf6YiEM5OY2jgsmqBjF2Xkp/8HgN/XDw/wD2+zebYGLLYtd2u3Gxx9edqJ8kQcU9LaMH+fiCFQyfq9UwTjQ
IDAQABAoGAD7L1a6Ess+9b6G70gTANWkKJpshVZDGB63mxKRepaJEX8sRJEQLqOYDnSc+pkK08IsfHreh4vrp9bsZuECr
B1OHSjwDB0S/fm3KEWbsaaXDUAU0dQg/JBMXAKzeATreoIYJItYgwzrJ++fuquKabAZumvOnWjyBIs2z103kDz2ECQQDn
n3JpHirmgVdf81yBbAJaXBNIPzOCcTh1zWfAs4EvrE35n2HvUQuRhy3ahUKXsKX/bGvwzmC206kbLTfEygVAKAEawXXZn
PkaAY2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC71FbNIgHBg5srsUyDj30s1oLmDVjmQJAIy7qLyOA+s
Cc6BtMavBgLx+bxCWfmsOZH0SX3179smTRAJ/HY64RREISLIQ1q/yw7IWBzxQ5WTHg1iNZFjKBvQJBAL3t/vCJwRz0EbS
5FaB/8UwhhsrbtXlGdnkojIGsmV0vHSf6poHQuiay/DV88pvhN11ZG8zHpeUhnAQccJ9ekzkCQDHHG9LYCoqTgsyYms//
cw4sv2nuOE1UezTjUFeq01sg0+WN96b/M5gnv45/Z3xZxzJ4HOCJ/NRwxNOTeUkw+ZY=
```

And receives a ciphertext message of:

```
Pob7AQZZSml618nMwTpx3V74N45x/rTimUqETl0yHq8F0dsekZgOT385J1s1HUZWCx6ZRFPFMJ1RNYR2Yh7AkQtFLVx91
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/rTimUqETl0yHq8F0dsekZgOT385J1s1HUZWCx6ZRFPFMJ1RNYR2Yh7AkQtFLVx91YDfb/Q+SkinBIBX59ER3/fDhrVKxIN4S6h2QmMSRblh4kdVhyY6cOxu+g48Jh7TkQ2Ig93/nCpAnyQ="
privatekey =
'MIICXAIBAAKBgQCWgjkEoyCxm9v6VBnUi5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnPAaDX3f2r4STZYYiqXGSH
CUBZcI90dvZf6YiEM5OY2jgsmqBjF2Xkp/8HgN/XDw/wD2+zebYGLLYtd2u3Gxx9edqJ8kQcU9LaMH+fiCFQyfq9UwTjQ
IDAQABAoGAD7L1a6Ess+9b6G70gTANWkKJpshVZDGB63mxKRepaJEX8sRJEQLqOYDnSc+pkK08IsfHreh4vrp9bsZuECr
B1OHSjwDB0S/fm3KEWbsaaXDUAU0dQg/JBMXAKzeATreoIYJItYgwzrJ++fuquKabAZumvOnWjyBIs2z103kDz2ECQQDn
n3JpHirmgVdf81yBbAJaXBNIPzOCcTh1zWfAs4EvrE35n2HvUQuRhy3ahUKXsKX/bGvwzmC206kbLTfEygVAKAEawXXZn
PkaAY2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC71FbNIgHBg5srsUyDj30s1oLmDVjmQJAIy7qLyOA+s
Cc6BtMavBgLx+bxCWfmsOZH0SX3179smTRAJ/HY64RREISLIQ1q/yw7IWBzxQ5WTHg1iNZFjKBvQJBAL3t/vCJwRz0EbS
5FaB/8UwhhsrbtXlGdnkojIGsmV0vHSf6poHQuiay/DV88pvhN11ZG8zHpeUhnAQccJ9ekzkCQDHHG9LYCoqTgsyYms//
cw4sv2nuOE1UezTjUFeq01sg0+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?

## B OpenSSL (RSA)

We will use OpenSSL to perform the following:

No	Description	Result
B.1	First we need to generate a key pair with:  openssl genrsa -out private.pem 1024  This file contains both the public and the private key.	What is the type of public key method used:  How long is the default key:  How long did it take to generate a 1,024 bit key?

		Use the following command to view the keys:  <code>cat private.pem</code>
<b>B.2</b>	Use following command to view the output file:  <code>cat private.pem</code>	What can be observed at the start and end of the file:
<b>B.3</b>	Next we view the RSA key pair:  <code>openssl rsa -in private.pem -text</code>	Which are the attributes of the key shown:  Which number format is used to display the information on the attributes:
<b>B.4</b>	Let's now secure the encrypted key with 3-DES:  <code>openssl rsa -in private.pem -des3 -out key3des.pem</code>	
<b>B.5</b>	Next we will export the public key:  <code>openssl rsa -in private.pem -out public.pem -outform PEM -pubout</code>	View the output key. What does the header and footer of the file identify?
<b>B.6</b>	Now we will encrypt with our public key:  <code>openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin</code>	
<b>B.7</b>	And then decrypt with our private key:  <code>openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt</code>	What are the contents of decrypted.txt

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

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

The public key should look like this:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDLrriUNYTyWuClIW7H6yea3hMV+rm029m2f6Iddt1ImHroXjNwYyt4E1kkc7AzO
y899C3gpx0kJK45k/CLbPnrHvKlvtQ0AbzWEQpOKXI+tw06PcqJNmTB8ITRLqIFQ++ZanjHWMw2Odew/514y1dQ8dcccO
uzeGhL2Lq9dtfhSxx+1cBLcyoSh/lQcs1HpXtpwU8JmXWJl409RQOVn3gousp/P/0R8mz/RwkmsFsyDRLgQK+xtQxbpbo
dpnz5lIOPwn5LnT0si7eHmL3WikTyg+QLZ3D3m44NCeNb+b0JbfaQ2ZB+lv8C30xy1xSp2sxzPZMbrZWqGSLPjgdIFIBL
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 file to 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.

## C 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:  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.	Can you 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):  A:  B:  Generator (last two bytes):  Order (last two bytes):

<b>C.3</b>	<p>Now generate your public key based on your private key with:</p> <pre>openssl ec -in priv.pem -text -noout</pre>	<p>How many bits and bytes does your private key have:</p> <p>How many bit and bytes does your public key have (Note the 04 is not part of the elliptic curve point):</p> <p>What is the ECC method that you have used?</p>
------------	---	---

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

## D 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):

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](https://asecuritysite.com/encryption/ecc_points)

First five points:

**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:

NIST521p:

SECP256k1:

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

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

## E 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 [ $\text{gcd}(\text{PHI}, e) = 1$ ]:

$e$ =

Now select a value of  $d$ , so that  $(e.d) \pmod{\text{PHI}} = 1$ :

[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:

$C = M^e \pmod{N} =$

Now decrypt your ciphertext with:

$M = C^d \pmod{N} =$

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):
    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  $(d \cdot e) \pmod{\phi(N)} = 1$ . 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  $(d, N)$ ?

## F PGP

**F.1** The following is a PGP key pair. Using <https://asecuritysite.com/encryption/pgp>, can you determine the owner of the keys:

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: OpenPGP.js v4.4.5
Comment: https://openpgpjs.org

xk0EXEOYvQECAIpLP8wFLxzgco1mpwgzcUzTlH0icggOIyuQKsHM4XNPugzu
X0NeaawrJhfi+f8hDrojj5Fv8jBI0m/KwFMNTT8AEQEAAcOUYm1sbCA8Ym1s
bEBob21lLmNvbnR7CdqQAQgAHwUCXEOYvQYLCQcIAwIEFQgKAgMWAgECGQEC
GwMCHgEACgkQOonsXEDYt2ZjkTAH/b6+pDfQLi6zg/Y0tHS5PPrv1323cwoay
VMCPjnwq+VfiNyXzY+UJKR1PXskzDvHML0yVpUcj1e5ChyT5LOW/ZM5NBFxD
mL0BAGDY1Tst06vVQu3jmfLzKMar4kLqqIuFFRCapRuHYLOjwlgJZS9p0bF
S0qs8zMEGpn9QZxkG8Yech3ghx1rvALTABEBAAHcXwQYAQgACQUCXEOYvQIb
DAKCRG2xcQN13zmMAGaf9w/XazfELDG1w3512zw12rkWm7rk97aFrtrx5W
xwA/5gqoVP0iqk1b9qpx7Rvd6rLKu7zoX7F+sQod1sCWrmW
=CXT5
-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: OpenPGP.js v4.4.5
Comment: https://openpgpjs.org

xcBmBFxDmL0BAGCKSz/MHy8c4HKJTKcIM3FM05R9InIIDiMrkCrBzOFzT7oM
1F9DXmmskyYX4vn/IQ0aIyerb/IwSNjvysBTDU0/ABEBAAH+CQMIBNTT/OPv
TJzgVF+fL0SLSNYP64QfNHav50744y0MLV/EZT3gsBw09v4XF2Sszj6+EHbk

```



```

O9gwi31BAIDgSaDsJYf7xPOhp8iEwwrUkC+jlGpdTsGDJpeYMiSVVv8Ycam
0g7MSRSL+dYQauIgtVb3dloLMPtuL59nVAYuIgD8HXyAH2vsEgSZSQn0kfVf
+dweqJxwFM/ux5PVKcuYsroJFBE01zas4ERfxbbwnsQgNHPjdIpueHx6/4EO
b1kmhOd6UT7BamubY7bcma1PBSv8PH31Jt8SzRRiawxsIDxiawxsQGHvbwUu
Y29tPsJ1BBABCAAFBQJcQ5i9BgsJBwgDAGQVCAoCAXYCAQIZAQIbAwIeAQAK
CRcg2xcQNj3ZmORMAF9vr6kN9AuLrOD9jS0dLk89G/XfbdzChrK8xw+odar5
V+I3JfNj5Qkphu9eyTMO8cws7JWlRyOV7kKHJPKs7D9kx8BmBFxDmLOBAGDY
lTst06vVQxu3jmfLzKMar4kLqqIuFFRCapRuHYLOjw1gJZS9p0bFS0qs8zME
GpN9QZxkg8YECH3gHxlrvaLTABEBAAH+CQMI2Gyk+BqVOgzgZX3C80JRLBRM
T4SLCHOUglwaspe+qatOVjeEuxA5DuSs0bVMrw7mJYQZLTjNkFAT92lSwfxY
gavS/biLlW3QGA0CT5mqijKr0nurkkekKBDsgjkjvbiOPLMYHfepOju1322
Nw4V3JQ04LBh/sdgGbrnWw3LhHEK4Qe70cuiert8C+S5xfg+T5RWAD15HR8u
UTyH8x1h0ZrOF7K0Wq4UcNvrUm6c35H6lC1C4Zaar4JSN8fZPqVKLlHTVCL9
lpDzXxqxkjS05KXXZBh5w18EGAEIAAKFA1xDmL0CGwwACgkQoNsXEDYt2Zja
BgH/cP12S3xCwxtvt+Zds8NdqysD06yve2ha7cc+Vl8AP+YKqFT9IkmZJW/a
qV+0VXeQyru86F+xfREKHdbAlqzMA==
=5NaF
-----END PGP PRIVATE KEY BLOCK-----

```

**F.2** Using the code at the following link, generate a key:

<https://asecuritysite.com/encryption/openpgp>

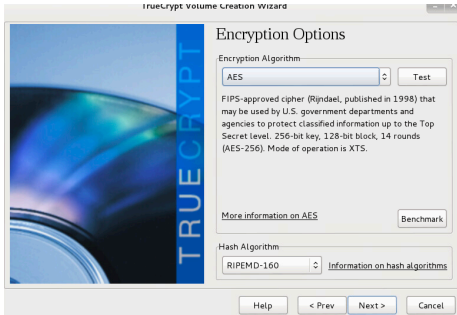
**F.3** An important element in data loss prevention is encrypted emails. In this part of the lab we will use an open source standard: PGP.

No	Description	Result
1	<p>Create a key pair with (RSA and 2,048 bit keys):</p> <p><b>gpg --gen-key</b></p> <p>Now export your public key using the form of:</p> <p><b>gpg --export -a "Your name" &gt; mypub.key</b></p> <p>Now export your private key using the form of:</p> <p><b>gpg --export-secret-key -a "Your name" &gt; mypriv.key</b></p>	<p>How is the randomness generated?</p> <p>Outline the contents of your key file:</p>
2	<p>Now send your lab partner your public key in the contents of an email, and ask them to import it onto their key ring (if you are doing this on your own, create another set of keys to simulate another user, or use Bill's public key – which is defined at <a href="http://asecuritysite.com/public.txt">http://asecuritysite.com/public.txt</a> and send the email to him):</p> <p><b>gpg --import theirpublickey.key</b></p> <p>Now list your keys with:</p> <p><b>gpg --list-keys</b></p>	<p>Which keys are stored on your key ring and what details do they have:</p>
3	<p>Create a text file, and save it. Next encrypt the file with their public key:</p> <p><b>gpg -e -a -u "Your Name" -r "Your Lab Partner Name" hello.txt</b></p>	<p>What does the <b>-a</b> option do:</p> <p>What does the <b>-r</b> option do:</p>

		<p>What does the –u option do:</p> <p>Which file does it produce and outline the format of its contents:</p>
4	<p>Send your encrypted file in an email to your lab partner, and get one back from them.</p> <p>Now create a file (such as myfile.asc) and decrypt the email using the public key received from them with:</p> <p><b>gpg -d myfile.asc &gt; myfile.txt</b></p>	Can you decrypt the message:
5	<p>Next using this public key file, send Bill (w.buchanan@napier.ac.uk) a question (<a href="http://asecuritysite.com/public.txt">http://asecuritysite.com/public.txt</a>):</p> <p>-----BEGIN PGP PUBLIC KEY BLOCK-----</p> <pre> mQENBFxEQeMBCACTgu58j4RuE340W3Xoy4PIX1Lv/8P+FUUFs8Dk4wO5ZUJN2NfN 45fIASdKCH8CV2wbCVwjKEP0h4p5IE+lrwQK7bwYx7Qt+qmrM5eLMUM8IvXA18wf AOPS7XektZxa4/jwagJupmmYL+Muv9o5haqYp1OYCCVR135KAZfx743YuWcNqvcR 3Em0+gh4F2TXsefjniwuJRGY3kbb/MAM2zc2f7FfCJvB1C300LB+KwCddZP/2311 nOqmzaVF0qQRHQ5EZGK3j3S4fzHNq14TMS3c21YkPOO/DV6BkgIHTG5NIIdVEDqh wV8c1pj0ZP7ShIE8cDhTy8k+xrIBYPUVfPmpABEBAAG0J0JpbGwgQnVjaGFuYXV4 PHcuYnVjaGFuYXV5AbmFwawVyLmFjLnVrPokBVAQTAQgAPHyHbK9cqx/wECcpQ6+5 TFPDJcQRPXoQBQJCREHjAhsDBQkDwmCABQsJCACCBHUKCQGLagQWAgMBAh4BAheA AAoJEFDPDjCqRPXoQ2KIh/2sRASqbrqCMNMRsiBo9xtCFzQ052odbzubIScnwzrDF Y9z+qPSAwawGO+1R3LPDH5SMLQ2YOSnqg8VvTJBt0jR9YGNX9/bqqVFRKKSQ0HiD Sb2M7phBdk4WLkqLZ/AfgHaLKpfNX0bq7WhqZ+Pez0nqjN08JkIog7LhaQzh/Chf Op1+whV0rEFuadQn83yF5DWB1Dt4fbzfvurEJb92tSrReHALQQA3h5wKTA0qxhdD 9XyEwknDrYCWiw0j0XwjivUre2fw3SKn8KHvJDeDYVKzYy18oA+da+xgs9b+n+Tq mM1fs1whw9Wryp0jBVLEs3yxLgE4elbCCmgiTnNpmMW5AQ0EXERB4WEIAKCPJqmM o8m6Xm163XtAZnx3t02EJSAV6u0yINIC8aEudNwg+/ptKKanUDm38dPn01lmgOyC FEu4qFJHbMidkEEac5J01gvHRK7jv94KF3vxqkr/bYnxltghqCfXesga9jFAHV8J M6sx4exOoc+/52YskpvdUS/eTPnwoQnbgjP+wsZpNq0owS6yO5urdF61vefgK5A TFB9lQUE01pb6IMKkCBZZvpZW0chbWPwCB9JZMuirdSyksuTldqgEsw7MyKBjCae E/ThuTazumad/PyEb0RCBODdMb55L6CD2W2DUquvBLI9FN6KTYwk5L/JZNAIWBV9 TKfevup933j1m+SAEQEAAYkBPAQYAQgAJhYhBK9cqx/wECcpQ6+5TFPDJcQRPXoQ BQJCREHjAhsMBQkDwmCAAoJEFDPDjCqRPXoQGRGH/3592g1F4+WRaPbuCgFEMihd ma5gp1U2J7NjNbV9Icy8VZSGw7UAT7FfmTPq1vwFM3w3gQCDXCKGztieUkZMTPqb LujBR4y55d5XDY6mP40zWRgdR1en2XsgHLPajRQpAhZq8ZvOdGe/ANCyXvdfHbGy aFAMUFAhxbkBITQKXh+EIkCHXDtDUHUXmAQvsZ8Z+Jm+ZwdhwkMsk43tw8UXLIynp AeOoATdohke3EVK5+0DC/jezcUwz2IKfw7LB3sQ4c6H8Ey8Pth1NAIgwMCDp5WTB DmForWtU6CpKtwIg/1b1ncbs1H2xAfEUX6ASHXR8vBOnIXWss21FuAaNMwe41myZ AQ0EXF1iYQEIALCmZgCvOira+YmtgQZuoos6veQ+uxysi9+waBtpEY5Bahe2BqtY /xrVE1bhekVftpuVeKtTYQxe7WiyjJ5xBnwNLzp/XedgIywgTWYnIHe+61DoBqtX US7Wfmc8CBcJahp9outNP+/yI8TzJMODtDDGAgF4n4Tb6nXRaWLEsn934ZfB88uG uvs6aofDWD1csdG0CnIGdOL+q+071j11/S13Pz+7E7ymPHJ1mFP6UxvFZFSHUUA6 Uk64uip1e61Lxbnfjdw3cZAFfxj7K0B+Hdb9kikZ1H5MYxomAmYbLZH9Zii1h 9ARR9K/+nes/7//83YzbyrvN1HxwKIDJ1sAEQEAABQnQm1sbCBcdwnoYw5hb1A8 dy5idwNoYw5hbkbUyXBPZXIuYWMudws+iQFUBBMBCAA+FiEEN/8zkuNo3g8ti6cX d5kNec0XwJMFAlxdYmECGWMFCQPCZwAFcwkIBWIGFQoJCASCBBYCAwECHGECF4AA CgkQd5kNec0XwJMKtggaI3FA+td7f0sdo+KfntWH4QnQVEarjJIXboFSx602wqME NZVPobw9ka4sYr9mejqm1vNzeAxJldAHVlk5BPMUWA/NdHozPvmvmbku7vjJxz/f MqpP2Pa10/zBdkw80pbJel2SbqBtFon4wQY3hSEBDYHCBWGI/ZbLSLXLJH2e+frL Z3wi6uzrGPERLNIhg1NADMDFU6mLTCSK8RaCJHjULOGy4zstiZGGBQIyr8209J0g tahUv/180s4DcvS3kyuJqQFv7sBYfDRCMQfWSXDWwJk1AmUbpQptZJAlYleB5tNE LizcJwHPou1oiY8/1tpfVHKv6EnzAqy12igj7F1S0rkBDQRcXWJhaQgAXuxras81 Css2KF0yKeXN/nufGL32bEPPoquMA7949eNatbF/6g8Gw5+sVa93q5ueBnVeQvn6 mywCF/62z8EL/vmmp47iagJuLdotSmayHr1mrJDogOg7GUG8mfFmZKwmp/Jzt2i +R0UDRkqp73RRncczKqSeGLRxjLnyY5+o17F4Nphen4XE0J10FgzAghAcSsZSEYQ9 xviFRHiCs4a72mFStuqIyQ6x3AS8otZN0GXezmIEoxxBz72jHurdJ15JS/Tt8qqq R69GvXgZx9+g7Vt0sWCoujljNskr5KPS4N0gFLKTFU17jlyfJpVN4yrs61mWTZHE BDWofdrQ/DTEuWARAQAB1QE8BBgBCAAmFiEEN/8zkuNo3g8ti6cXd5kNec0XwJMF </pre>	Did you receive a reply:

	<pre> AlxdYmECGwwFCQPCZwAACgkQd5kNec0XwJO89Af/R11nf4Ty4MjgdbRvo43crcn+ Z17LPt+IBpPXoyV/a//5CDZCWSECJ7ijPmAx5Zgyw8SGt10EW2k0CEhDwPCds32r 6iEIwaoMT7NXK0gZxYfAjT0iYE1cR6zxZVcPkU5561TB5yzt51+H6GshQ5eUIH+ fs6DMRGrWTEZENJ2EVof08DUJanaTi4ImIJF6Gidwmt+YoL1d5THZEWBXYNVRIeZ K+FwAZm7a5gBTCgeafvUDbw3Drecm6y7YTuoFHF321aHNK8/9Lu0T5JTX9jhYvTr 1BrwqYijj2gvKYWAK5gkJdgUuOdNVLCn1Rael1GetiL3BEVZsfE3bHANFS107Bw== =DvMI -----END PGP PUBLIC KEY BLOCK----- </pre>	
6	Next send your public key to Bill (w.buchanan@napier.ac.uk), and ask for an encrypted message from him.	

## G TrueCrypt

No	Description	Result
1	<p>Go to your <b>Kali</b> instance (User: root, Password: toor). Now <b>Create a new volume</b> and use an <b>encrypted file container</b> (use <code>tc_Yourname</code>) with a Standard TrueCrypt volume.</p> <p>When you get to the Encryption Options, run the benchmark tests and outline the results:</p> 	<p>CPU (Mean)</p> <p>AES:  AES-Twofish:  AES-Two-Seperent  Serpent -AES  Serpent:  Serpent-Twofish-AES  Twofish:  Twofish-Serpent:</p> <p>Which is the fastest:</p> <p>Which is the slowest:</p>
2	Select AES and RIPEMD-160 and create a 100MB file. Finally select your password and use FAT for the file system.	What does the random pool generation do, and what does it use to generate the random key?
3	Now mount the file as a drive.	Can you view the drive on the file viewer and from the console? [Yes][No]
4	Create some files your TrueCrypt drive and save them.	<p>Without giving them the password, can they read the file?</p> <p>With the password, can they read the files?</p>

The following files have the passwords of “Ankle123”, “foxtrot”, “napier123”, “password” or “napier”. Determine the properties of the files defined in the table:

File	Size	Encryption type	Key size	Files/folders on disk	Hidden partition (y/n)	Hash method
<a href="http://asecuritysite.com/tctest01.zip">http://asecuritysite.com/tctest01.zip</a>						
<a href="http://asecuritysite.com/tctest02.zip">http://asecuritysite.com/tctest02.zip</a>						
<a href="http://asecuritysite.com/tctest03.zip">http://asecuritysite.com/tctest03.zip</a>						

Now with **truecrack** see if you can determine the password on the volumes. Which TrueCrypt volumes can truecrack?

## H Reflective statements

1. In ECC, we use a 256-bit private key. This is used to generate the key for signing Bitcoin transactions. Do you think that a 256-bit key is largest enough? If we use a cracker what performs 1 Tera keys per second, will someone be able to determine our private key?

## I What I should have learnt from this lab?

The key things learnt:

- The basics of the RSA method.
- The process of generating RSA and Elliptic Curve key pairs.
- To illustrate how the private key is used to sign data, and then using the public key to verify the signature.