


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/esecurity/unit04>

Demo: <https://youtu.be/3n2TMpHqE18>

A RSA Encryption

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

e =

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

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.

A.2 The following defines a public key that is used with PGP email encryption:

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

Version: GnuPG v2

```
mQENBFTzi1ABCADIEWchOyqRQmU4AyQAMj2Pn68Sqo91TPdPcItwo9LbTdv1YCFZ
w3qLlp2RORMP+Kpdi92CIhdUYHdmZFhZ3IWTBgo9+y/Np9UJ6tNGocrgsq4xwz15
4vx4jJRddC7QySSh9UxDpRWF9sgqEV1pah136r95ZuyjC1EXnoNxdLJtx8PliCXc
hv/v4+kfOyZyH+HDJ4xP2bt1S07dkasYZ6cA7BHYi9k4xgEwxVvYtnjSPjTSQY5R
cTayXveGafuxmhSauZKiB/2TFerjEt49Y+p07tPTLX7bhMBVbUvojtt/JeUKV6vK
R82dmOd8seUvhwOHYB0JL+3S7PgFFsLo1NV5ABEBAAG0LkJPbGwgQnVjaGFuYW4g
KE5vbmUpIDx3LmJ1Y2hhbmFuQG5hcGllci5hYy51az6JATKEEwECACMFA1Tzi1AC
GwMHCwkIBWMCAYQVCAIJCgSEFgIDAQIEAQIXgAAKCRDSAFZRGtdPQi13B/9KHeFb
11AxqbafFGRDEvx8UfPnEww4FFqwhcr8RLWyE8/COlupB/5AS2yvojmbNFMGzURb
LGf/u1LVH0a+NHQu57u8Sv+g3bBthEPH4bKaEzBYRS/dYHOx3APFyIayfm78JVRf
zdeTOOf6PaXUTRx7iscCTkn8DUD3lg/465ZX5aH3HWFFX500JSPSt0/udqjoQuAr
WA5JqB//g2GfzZe1UzH5Dz3PBbJky8GiIfLm00XSEIqAmpvc/9NjzAgjOW56n3Mu
sjVkiBc+1ljw+roo97CfJmppmtcOvehvQv+KG0LZnpibiWvM3vT7E6kry4gEbDu
enHPDqhsvcqTDqaduQENBFTzi1ABCACzPjgZLK/sge2rMLURUQQ6l02Urs/GilGC
ofq3WPnDt5HearwMMWn65Pb0Dj0i7vnorhL+fdb/J8b8QTiyp7i03dzVhDahcQ5
8afvcjQtQstY8+K6kZFzQOBgyOS5rHAKHNSPFq45MlnPo5aadVP7s9mdMILITvlb
CFhcLoC60qy+JoahUpJqHBqGc48/5NU4qbt6fB1AQ/H4M+6og40ozohgkQb80Hox
YbJV4sv4vYMULd+FK0g2RdGenMM/awdqYo90qb/w2aHCCyXmhGHEEUok9jbc8cr/
xrWL0gDwlwpad8RfQwyVU/vZ3Eg3OseL4SedEmwOO
cr15XDis6dpABEBAAGJAR8E
GAECAAKFA1Tzi1ACGwwACgkQ7ABWURrXT0KZTgf9Fupkh3wv7ac5M2wwdEjt0rDx
nj9kxH99hhuTX2EHXunLH+SwLGHBq5O2sq3jfP+owEhs8/Ez0j1/fSKIqAdl3mB
dbqWpjzPTY/m0It+ww3epOM75uwjD35PF0rKxxZmEf6SrjZD1sk0B9bRy2v9iWN9
9ZkuvCFH4vT++PognQLTUqNX0FGpDlagrG0lXSctJWQXCXPfwdtbIdThBgZ4f1Z
ssAIBCaBlQkzfbPvrMzdTIP+AXg6++K9Sn09N/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>

A.3 Bob has a private RSA key of:

```
MIICXAIBAAKBgQCWgJkeoyCXm9v6VBnUi5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnPA
aDX3f2r4STZYIiqXGSHCUBZcI90dvZF6YiEM5OY2jgsmqBjF2Xkp/8HgN/XDw/wD2+zebYGLLY
td2u3GXx9edqJ8kQcu9LaMH+fiCFQYf9UwtjQIDAQABAoGAD7L1a6Ess+9b6G70gTANwKkJs
hVZDGB63mxKRepaJEX8sRJEqLqOYDNsc+pkK08IsfHreh4vrp9bsZuECrB10HSjwDB0S/fm3KE
wbsaaXDUAu0dQg/JBMXAKzeATreoIYJItYgwrJ++fuquKabAZumvOnWjYBis2z103kDz2ECQQ
Dnn3JpHi rmgvdf81yBbAJaXBxNIPzOcCth1zwFAS4EvRE35n2HvUQuRhy3ahUKXSKX/bGvwzmC
206kbLTfeygVAKEAwxxZnPkAAy2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC71
FbNiGHBg5srsuYdj30sloLmDVjmQJAiy7qLYOA+sCC6BtMavBgLx+bxCwFmsoZHOSX3l79smTR
AJ/HY64RREIsLIQ1q/yw7IWBzxQ5WTHglNZFjKBVQJBAL3t/vCJWRz0Ebs5FaB/8UwhhsrbtX
lGdnkOjIGsmV0vHSf6poHqUiay/DV88pvhN1lZG8zHpeUhnAQccJ9ekzkCQDHG9LYCOqTgsyY
ms//cw4sv2nuOE1uezTjUFeqO1sgO+WN96b/M5gnv45/Z3xZxZJ4HOCJ/NRWxNOTeUkw+zY=
```

And receives a ciphertext message of:

```
Pob7AQZZSm1618nMwTpx3V74N45x/rTimUqETl0yHq8F0dsekZgOT385Jl1s1HUzWCx6ZRFPFMJ
1RNYR2Yh7AkQtFLVx9lYdfb/Q+SkInBIBX59ER3/fDhrVKxIN4S6h2QmMSRblh4KdVhyY6Coxu
+g48Jh7TkQ2Ig93/nCpAnyQ=
```

Using the following code:

```
from Crypto.PublicKey import RSA
from Crypto.Util import asn1
```

```

from base64 import b64decode

msg="Pob7AQZZSm1618nMwTpx3V74N45x/rTimUqET10yHq8F0dsekZgOT385J1s1HUzWCx6ZR
FPFMJ1RNYR2Yh7AkQtFLVx91YDfb/Q+SkinBIBX59ER3/fDhrVKxIN4S6h2QmMSRb1h4KdVhyY
6cOxu+g48Jh7TkQ2Ig93/nCpAnYQ="
privatekey =
'MIICXAIBAAKBgQCWgjkQeoyCxm9v6VBnuI5ihQ2knkdxGDL3GXLIUU43/froeqk7q9mtxT4AnP
AaDX3f2r4STZYYiqXGSHCUBZcI90dvZf6YiEM5OY2jgsmqBjf2Xkp/8HgN/XDw/wD2+zebYGLL
Ytd2u3Gxx9edqJ8kQCU9LaMH+ficFQyfq9UwTjQIDAQABAoGAD7L1a6Ess+9b6G70gTANWkKJp
shVZDGB63mxKRepaJEX8sRJEqLqOYDNsc+pkK08IsfHreh4vrp9bsZuECrB1OHSjwDB0S/fm3K
EwbsaaXDUAu0dQg/JBMXAKzeATreoIYJIItYgwzrJ++fuquKabAZumvOnWJyBIs2z103kDz2ECQ
QDnn3JpHirmgvdf81yBbAJaXBXNIPzOCcthlzWfAs4EvRE35n2HvUQuRhy3ahUKXSKX/bGvwzm
C206kbLTfEygVAKEAwXXznPkaAY2vuoUCN5NbLZgegrAtmU+U2woa5A0fx6uXmShqxo1iDxEC7
1FbNIgHBg5srsuyDj3os1oLmDVjmQJAiy7qLyOA+sCc6BtMavBgLx+bxCwFmsoZH0SX3179smt
RAJ/HY64RREIsLIQ1q/yw7IWBzxQ5WTHg1iNZFjKBvQJBAL3t/vCJwRz0Ebs5FaB/8Uwhhsrbt
XlGdnkOjIGsmV0vHSf6poHquiay/DV88pvhN11ZG8zHpeUhnaQccJ9ekzkCQDHHG9LYCOqTgsy
Yms//cw4sv2nu0E1uezTjUFeqOlsgo+WN96b/M5gnv45/Z3xZxzJ4HOCJ/NRWXN0tEukw+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 using OpenSSL to perform the following:

No	Description	Result
B.1	<p>First we need to generate a key pair with:</p> <pre>openssl genrsa -out private.pem 1024</pre> <p>This file contains both the public and the private key.</p>	<p>What is the type of public key method used:</p> <p>How long is the default key:</p> <p>How long did it take to generate a 1,024 bit key?</p> <p>Use the following command to view the keys:</p> <pre>type private.pem (or cat private.pem in Linux)</pre>
B.2	<p>Use following command to view the output file:</p> <pre>cat private.pem</pre>	<p>What can be observed at the start and end of the file:</p>

B.3	<p>Next we view the RSA key pair:</p> <pre>openssl rsa -in private.pem -text</pre>	<p>Which are the attributes of the key shown:</p> <p>Which number format is used to display the information on the attributes:</p>
B.4	<p>Let's now secure the encrypted key with 3-DES:</p> <pre>openssl rsa -in private.pem -des3 -out key3des.pem</pre>	
B.5	<p>Next we will export the public key:</p> <pre>openssl rsa -in private.pem -out public.pem -outform PEM -pubout</pre>	View the output key. What does the header and footer of the file identify?
B.6	<p>Now we will encrypt with our public key:</p> <pre>openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin</pre>	
B.7	<p>And then decrypt with our private key:</p> <pre>openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt</pre>	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
AAAAB3NzaC1yc2EAAAADAQABAAQDLrriUNYTywuC1IW7H6yea3hMV+rm029m2f6Id
dtlImHrOXjNwYyt4Elkkc7AzOy899C3gpx0kJK45k/CLbPnrHvKLvtQ0AbzWEQpOKxI+
tw06PcqJNmTB8ITRLqIFQ++ZanjHWMw2Odew/514y1dQ8dccOuZeGhL2Lq9dtfhSxx+
1cBLcyoSh/1Qcs1HpXtpwU8JMxwJl409RQOVn3gOusp/P/0R8mz/RwkmsFsyDRLgQK+x
tQxbpbodpnz5lIOPwn5Lnt0si7eHmL3wikTyg+QLZ3D3m44NCeNb+b0JbfaQ2ZB+lv8C
30xy1xSp2sxzPZMbrZWqGSLPjgDiFIBL w.buchanan@napier.ac.uk
```

View the private key. What is 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
```

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	<p>First we need to generate a private key with:</p> <pre>openssl ecparam -name secp256k1 -genkey -out priv.pem</pre> <p>The file will only contain the private key (and should have 256 bits).</p> <p>Now use “cat priv.pem” to view your key.</p>	Can you view your key?
C.2	<p>We can view the details of the ECC parameters used with:</p> <pre>openssl ecparam -in priv.pem -text -param_enc explicit -noout</pre>	<p>Outline these values:</p> <p>Prime (last two bytes):</p> <p>A:</p> <p>B:</p> <p>Generator (last two bytes):</p> <p>Order (last two bytes):</p>
C.3	<p>Now generate your public key based on your private key with:</p> <pre>openssl ec -in priv.pem -text -noout</pre>	How many bits and bytes does your private key have:

		<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

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, NIST512p and SECP256k1:

NIST192p:

NIST512p:

SECP256k1:

By searching on the Internet, can you find where SECP256k1 is used?

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

E Inverse of a value mod N

E.1 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

xk0EXEOYvQECAIpLP8wFLxzgco1mpwgzcUzT1H0icggOIyuQKsHM4XNPugzu
X0NeaawrJhfi+f8hDRojJ5Fv8jBI0m/KwFMNTT8AEQEAAc0UYm1sbCA8Ym1s
bEBob211LmNvbT7CdQQQAQgAHwUCXEOYvQYLCQCIawIEFQgKAgMwAgECGQEC
GwMCHgEACgkQoNSXEDYt2ZjKTAH/b6+pdFQLi6zg/Y0tHS5PPRv1323cwoay
vMcPjnwq+VfiNyXzY+UJKR1PXskzDvHMLoyVpUcj1e5ChyT5LOW/ZM5NBFXD
mL0BAGDY1TsT06vvQxu3jmfLzKMAr4kLqqIuFFRCapRuHYLOjw1gJZS9p0bF
S0qS8zMEGpN9QZxkG8YECH3gHx1rVALtABEBAAHCXwQYAQgACQUCEYOYvQIb
DAAKCRcg2xcQNi3zmMAGAf9w/XazfELDG1w3512zw12rKwM7rK97aFrtxz5W
XwA/5gqovP0iQxklb9qpX7Rvd6rLKu7zox7F+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+fLoSLSNYP64QfNHav5O744y0MLV/EZT3gsBwO9v4XF2SsZj6+EHbk
O9gwi31BAIDgSaDsJYf7xPOhp8iEwwwrUkC+jlGpdTSGDJpeYMIsvVv8Ycam
0g7MSRSL+dYQauIgtVb3dloLMPtuL59nVAYuIgd8HXyaH2vsEgSZSqN0kfVf
+dweqJxwFM/ux5PVKcuYsroJFBE0lzas4ERfxbbwnsQgNHPjdIpueHx6/4EO
b1kmhOd6UT7BamubY7bcma1PBSv8PH31Jt8SzRRiawxsIDx1awxsQGhvbWUu
Y29tPsJ1BBABCAAFBQJcQ5i9BgSJBwgDAGQVCAoCAXYCAQIZAQIbAwIeAQAK
CRGg2xcQNi3ZmORMAf9vr6kN9AuLrOD9jS0dLk89G/XfbdzChrK8xw+Odar5
V+I3JfNj5QkpHU9eyTMO8cws7Jw1RyOV7kKHJPks7D9kx8BmBFxDmL0BAGDY
lTst06vVQxu3jmfLzKMAR4kLqqIuFFRCapRuHYLOjw1gJZS9p0bFS0qs8ZME
GpN9QZxkG8YECH3gHx1rvAltABEBAAH+CQMI2Gyk+BqVOgzgZX3C80JRLBRM
T4sLCHOUglwaspe+qatOVjeEuxA5DuSs0bVmrw7mJYQZLTjNkFAT921SwfxY
gavS/bILlw3QGA0CT5mqijKr0nurKkekKBDsgjkjVbIoPLMYHfepOju1322
Nw4V3JQ04LBh/sdgGbrnww3LhHEK4Qe70cuiert8C+S5xfG+T5RWADi5HR8u
UTYH8x1h0ZrOF7K0Wq4UcnvrUm6c35H6lC1C4Zaar4JSN8fZPqVKLlHTVcl9
lpDzXxqxKjS05KXXZBh5w18EGAElAAkFA1xDmL0CGwwACgkQoNsXEDYt2ZjA
BgH/cP12s3xCwxtVt+Zds8NdqysDO6yve2ha7cc+V18AP+YKqFT9IkMZJW/a
qV+0Vxeqyyru86F+xfrEKHdbAlqzMA==
=5NaF
-----END PGP PRIVATE KEY BLOCK-----

```

F.2 Using the code at the following link, generate your own key:

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

F.3 An import element in data loss prevention is encrypted emails. In this part of the lab we will use an open source standard: PGP. On your instance, install GPG from:

<http://gpg4win.org/download.html>

No	Description	Result
1	<p>Create a key pair with (RSA and 2,048 bit keys):</p> <p>gpg --gen-key</p> <p>Now export your public key using the form of:</p> <p>gpg --export -a "Your name" > mypub.key</p> <p>Now export your private key using the form of:</p> <p>gpg --export-secret-key -a "Your name" > mypriv.key</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:</p> <p>gpg --import theirpubickey.key</p> <p>Now list your keys with:</p> <p>gpg --list-keys</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>What does the <code>-a</code> option do:</p>

	<pre>gpg -e -a -u "Your Name" -r "Your Lab Partner Name" hello.txt</pre>	<p>What does the <code>-r</code> option do:</p> <p>What does the <code>-u</code> 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 <code>myfile.asc</code>) and decrypt the email using the public key received from them with:</p> <pre>gpg -d myfile.asc > myfile.txt</pre>	<p>Can you decrypt the message:</p>
5	<p>Next using this public key file, send Bill (w.buchanan@napier.ac.uk) a question (http://asecuritysite.com/public.txt):</p> <pre>-----BEGIN PGP PUBLIC KEY BLOCK----- Version: BCPG C# v1.6.1.0 mQENBFTVF+gBCACKpcMPybSe1NTE1hDg86gPcQqoT8kD9oS/ankG wbB4R5ZT+3Ny MZWZWT431L99R7sfk1ug1wvkqko74Lemy9pBF/rbwewev6mCR3z1 V3yTTv3zP1V5 tLcz3K65f1RHPQU/FzxqH1T4kaH6DiL/UuKKcyYMxXNnqERitJP U7ZJVhqem3gi 4cG4znKY5fw8bdSpNC//pgkDzEawYJFdyq/KqCWRK5r/Egj7FVHa LGC371DgZKR5 dBoIVaOTfxykJLe3Vc3dIv9LU58U3YHqsc/w6X4E5R/eEnp0IwKy b7oXdrFOM5ud DSOj7aT24IqZW678vNtufGdr4OD+BF5r2UZpABEBAAG0GHcuYnVj aGFuYW5hQG5h cG11ci5hYy51a4kBHAQQAQIABgUCVO8X6AAKCRBOV4Uk9xMSXJgN B/4jfAnXLHjZ +I4z3Hhqn9UMoku6Q4cQtrGX00he1ymKZTMXNoSKht5fB9GB1Iib wMkZHxcUNmUB PuAwq+RAhFqtRkch3x1a5eNBhEvcfi9hs21s43gfsrxjMzekY6dy zD/ePM7HvihJ vrsQNZNI7ZiaP5viCZFgQqmwYQA1LCrEy/xpSXBnrqRowuti+2+x eZsswitYLAZA ryDMgCG9GPuSfkmvatYJJr15QAhj1p0FKERhL1/h3bh18i8L1h1K 9tEBXIjF4ZiY ivV1bx5G36jciOrKCLi7/m6xhHh86brRQA++qwUdXU/3MMqvRwui nSO9NYeVcf6Y V66cJqTgdR1F =uiw7 -----END PGP PUBLIC KEY BLOCK-----</pre>	<p>Did you receive a reply:</p>
6	<p>Next send your public key to Bill (w.buchanan@napier.ac.uk).</p>	

--	--	--

G TrueCrypt

No	Description	Result
1	<p>Go to your Ubuntu instance. Now Create a new volume and use an encrypted file container (use <code>tc_yourname</code>) with a Standard TrueCrypt volume.</p> <p>When you get to the Encryption Options, run the 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 RIPMD-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	<p>Create some files your TrueCrypt drive and save them.</p> <p>Next dismount your drive, and copy the file to the provided USB stick. Give the USB stick to your neighbour, and see if they can view the file contents.</p>	<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
http://asecuritysite.com/tctest01.zip						
http://asecuritysite.com/tctest02.zip						

http://asecuritysite.com/tctest03.zip						
---	--	--	--	--	--	--

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?

2. If someone takes our elliptic curve public key, how might they determine our public key?

G 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.

Notes

To setup your Python to run Python 2.7:

```
sudo update-alternatives --set python /usr/bin/python2.7
```

To install a Python library use:

```
easy_install libname
```

or:

```
pip install libname
```