

# Networks and Internet Applications

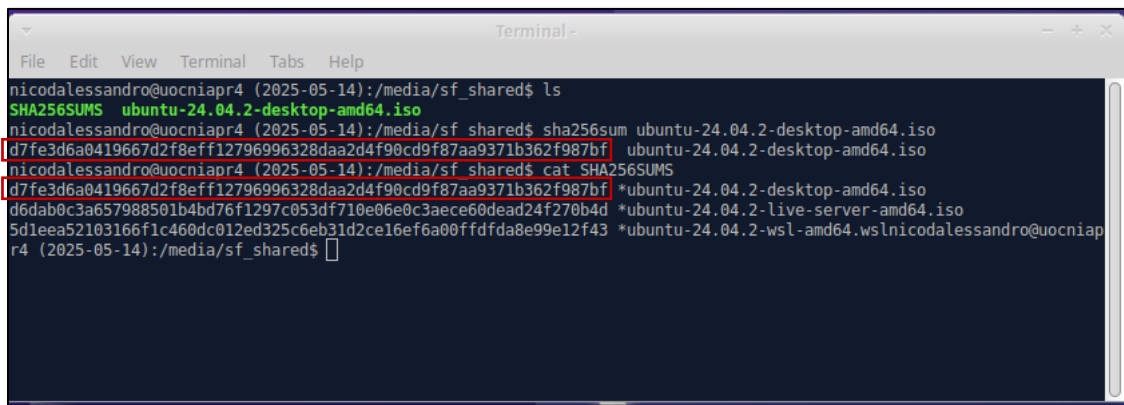
## Practical Exercise 4 – PR4

Nicolas D'Alessandro Calderon

### First Part (maximum qualification C-): integrity

#### Exercise 1

- a) We can observe that the hash calculated with sha256 sum matches the one published in SHA256SUMS for this ubuntu image downloaded.



```

nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ ls
SHA256SUMS  ubuntu-24.04.2-desktop-amd64.iso
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ sha256sum ubuntu-24.04.2-desktop-amd64.iso
d7fe3d6a0419667d2f8eff12796996328daa2d4f90cd9f87aa9371b362f987bf  ubuntu-24.04.2-desktop-amd64.iso
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ cat SHA256SUMS
d7fe3d6a0419667d2f8eff12796996328daa2d4f90cd9f87aa9371b362f987bf *ubuntu-24.04.2-desktop-amd64.iso
d6dab0c3a657988501b4bd76f1297c053df710e06e0c3aece60dead24f270b4d *ubuntu-24.04.2-live-server-amd64.iso
5d1eea52103166f1c460dc012ed325c6eb31d2ce16ef6a00ffdfda8e99e12f43 *ubuntu-24.04.2-wsl-amd64.wsl
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$

```

- b) The SHA family or Secure Hash Algorithm includes several different versions:
- **SHA-0:** Is the first version, but it is currently not used because it has security problems.
  - **SHA-1:** Is more secure than the previous release, but it is also currently broken and not recommended to use.
  - **SHA-2:** That includes *SHA-224*, *SHA-256*, *SHA-384* and *SHA-512* are stronger and still used today.
  - **SHA-3:** Is the newest version designed as an alternative to *SHA-2*.

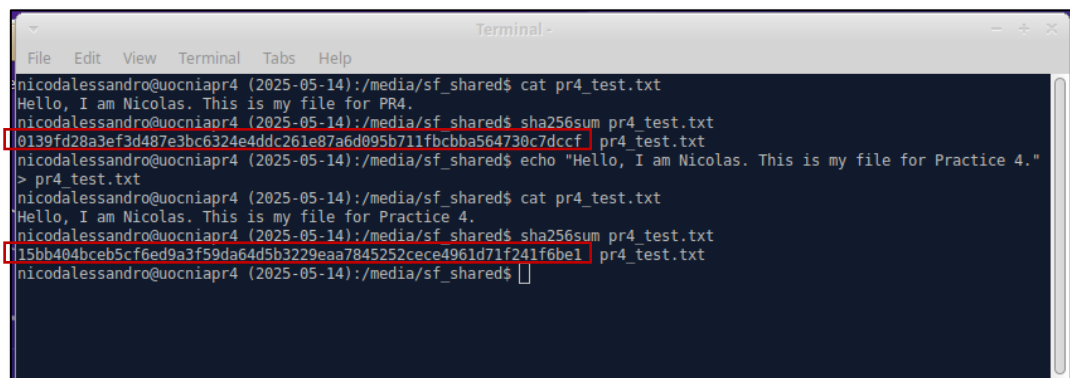
Each version creates a checksum (hash) with a specific length:

SHA Version	Hash Length
SHA-0	160 bits
SHA-1	160 bits
SHA-224	224 bits
SHA-384	384 bits
SHA-512	512 bits
SHA-3 (256)	256 bits

c)

- **SHA-0** has a design problem, and it has been proved with several vulnerabilities. It is not safe because the attackers can find what is called collision (two different files with the same hash). It has been deprecated.
- **SHA-1** was safer than **SHA-0**, but it is also broken. In 2017 Google showed a collision attack on SHA-1, so it has been proved to have the same vulnerabilities as the previous version. Today, SHA-1 is not secure and shouldn't be used.

d)



```

Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ cat pr4_test.txt
Hello, I am Nicolas. This is my file for PR4.
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ sha256sum pr4_test.txt
0139fd28a3ef3d487e3bc6324e4ddc261e87a6d095b711fbcba564730c7dccf pr4_test.txt
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ echo "Hello, I am Nicolas. This is my file for Practice 4."
> pr4_test.txt
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ cat pr4_test.txt
Hello, I am Nicolas. This is my file for Practice 4.
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$ sha256sum pr4_test.txt
15bb404bceb5cf6ed9a3f59da64d5b3229eaa7845252cece4961d71f241f6be1 pr4_test.txt
nicodalessandro@uocniapr4 (2025-05-14):/media/sf_shared$

```

- e) No, it is not possible to obtain the original text from the hash, because a hash function like SHA-256 is **one-way**. This means that we can create a hash from a text, but we can't go back from the hash to the original text.

Hash function does not save information about the original file, this means that hashes are good to check integrity but not for reading or recovering the original data.

## Second Part (maximum qualification C+): symmetric encryption

### Exercise 2

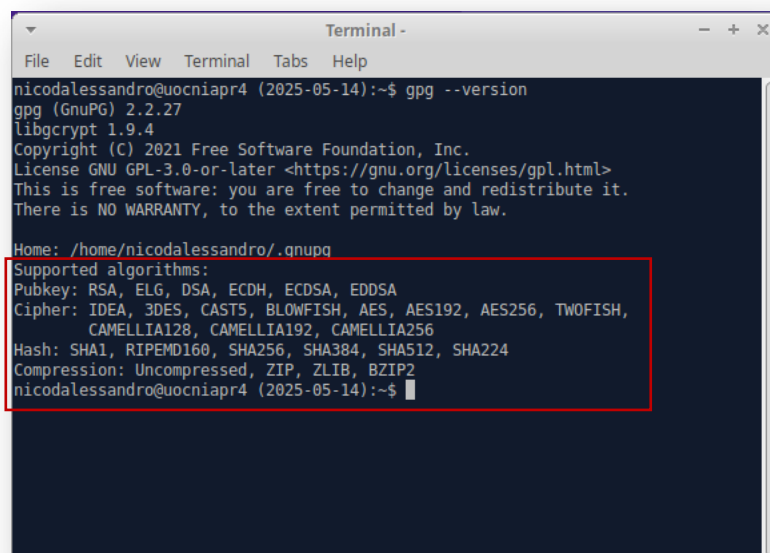
a) A strong symmetric key should be hard for the attackers to guess or break. Two features that can make the key difficult to break are:

1. **Long key:** If the key is longer, it will be harder to guess or break. For instance, a 128-bit key has more possible combinations than a 64-bit key making it stronger against brute force attacks (when someone tries many keys until one works).
2. **Random key:** A good key is random, meaning that it should not be based on patterns, names or something easy to guess. For example, a key generated with random characters such as **B9lfZ@73\*Lx#** is going to be much harder to guess than something like **nicolas123**.

b) Three commonly used symmetric algorithms for encryption are:

1. **AES Advanced Encryption Standard - Key length 128, 192 or 256 bits.**  
This is probably the most used symmetric algorithm today. It is known for being fast and secure. It is used in HTTPS, VPNs and Wi-Fi encryption.
2. **Camellia - Key length 128, 192 or 256 bits**  
This is similar to AES in terms of security and speed, but it is used in some systems where AES is not supported such as license or law restrictions.
3. **Blowfish - Key length up to 448 bits.**  
This algorithm was very popular in the 90's. Even if it's still safe to use, it's slower than AES so it is less used today.

c) We can observe in the image the supported algorithms:



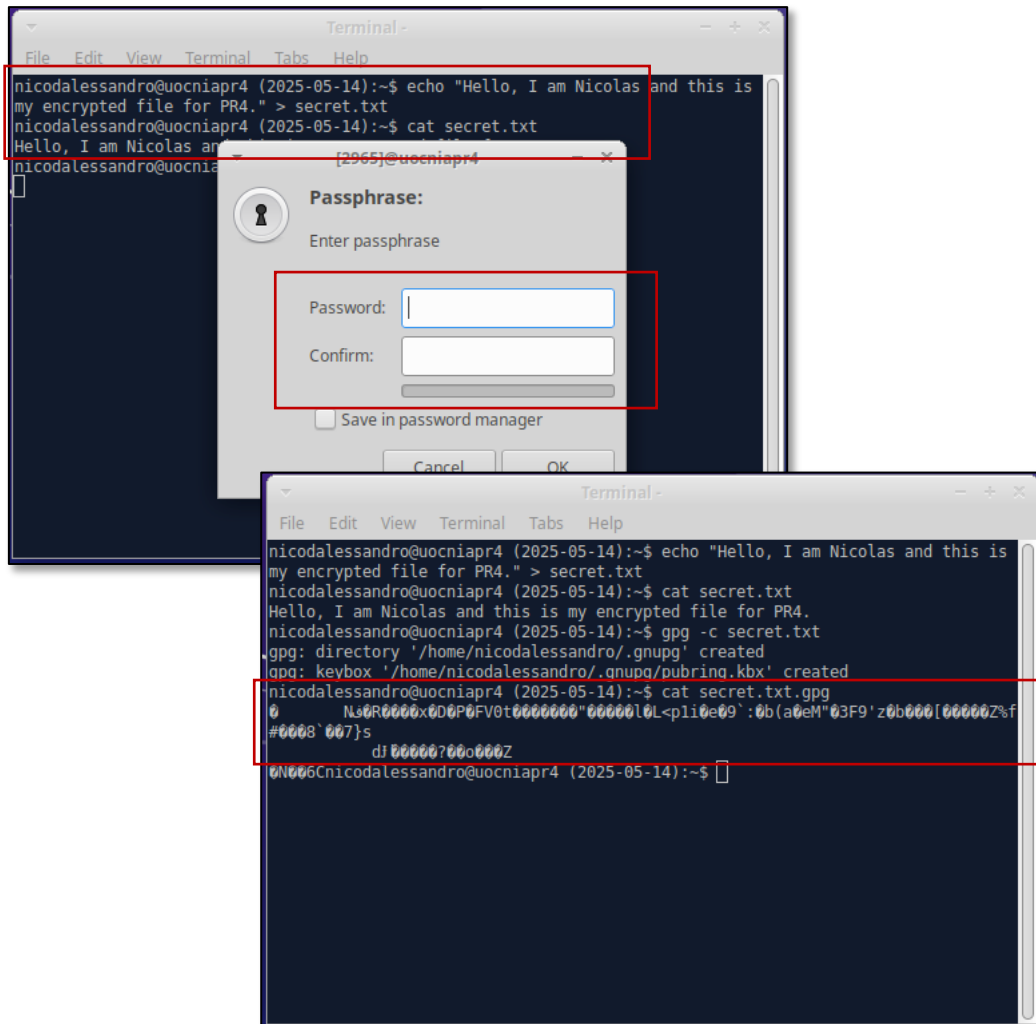
```

Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --version
gpg (GnuPG) 2.2.27
libgcrypt 1.9.4
Copyright (C) 2021 Free Software Foundation, Inc.
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

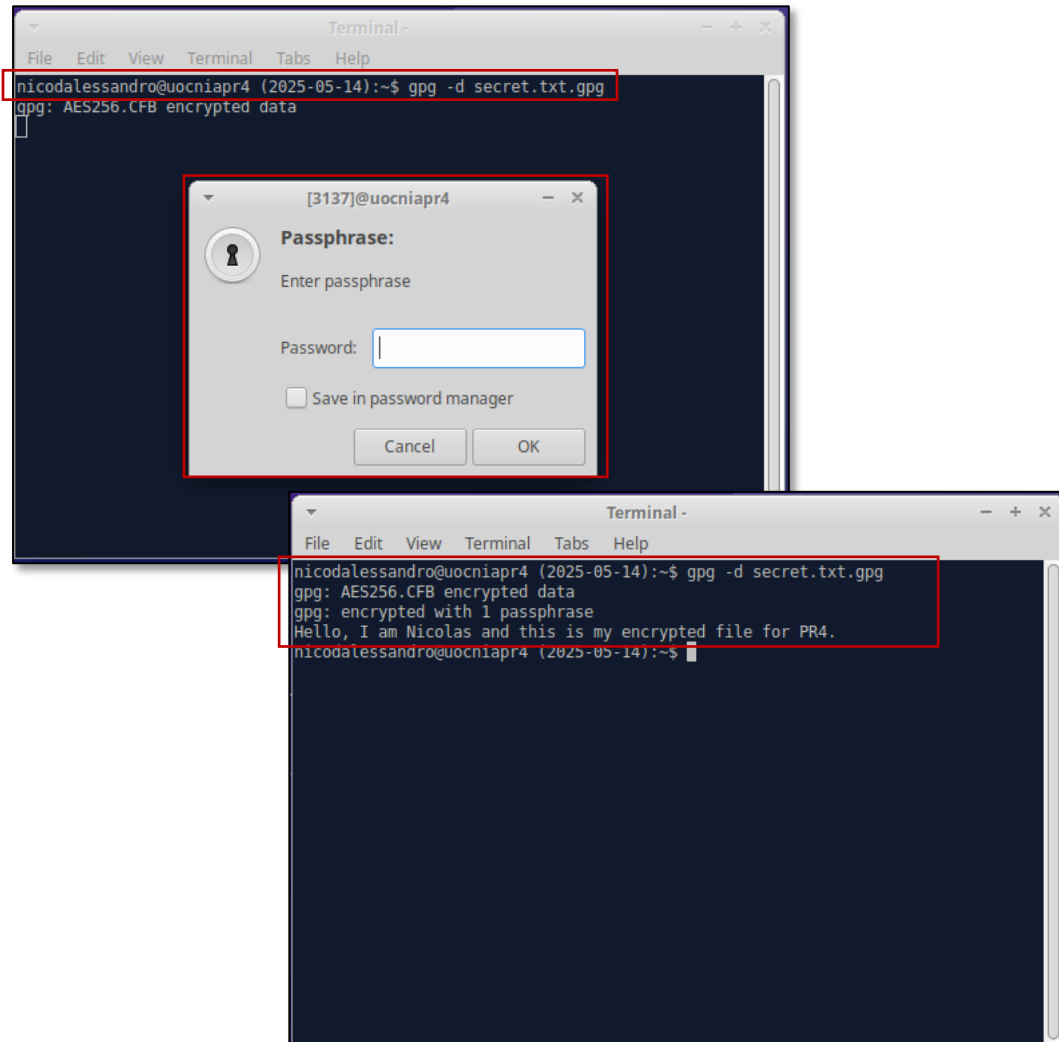
Home: /home/nicodalessandro/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
nicodalessandro@uocniapr4 (2025-05-14):~$

```

- d) First, I created a **secret.txt** text file with the echo command. Then I encrypted the created file using GPG and symmetric encryption. GPG asked for a password and created the **secret.txt.gpg** file. When I try to open the **.gpg** file with **cat** command I see unreadable characters, showing that the file is encrypted and can't be read without the password:



- e) We can use the command **gpg -d [our-encrypted-file.gpg]**. GPG asks for the password, and after entering it correctly, we will see the original message:



- f) As we saw in the previous exercise, the default algorithm used when none a specific parameter added is AES256 (AES algorithm with a 256-bit key):

```
Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg -d secret.txt.gpg
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
Hello, I am Nicolas and this is my encrypted file for PR4.
nicodalessandro@uocniapr4 (2025-05-14):~$
```

1. To encrypt a text file using the CAMELLIA256 algorithm, I first created the text file and then we add the parameters **gpg -c --cipher-algo CAMELLIA256 camelliaPR4.txt**

**gpg** is the encryption program.

**-c** is to indicate symmetric encryption with a password.

**--cipher-algo CAMELLIA256** tells to gpg to use the CAMELLIA256 algorithm instead of the default AES256.

**[filename-to-be-encrypted.txt]** is the file we want to encrypt with the specified algorithm.

2. Again, I tried to open the encrypted file with **cat** but I saw random characters meaning that the text file was correctly encrypted.

3. Finally, I used **gpg -d** to decrypt with the given password and see the original file:

```
Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ echo "hello! this file will was encrypted with CAMELLIA256." > camelliaPR4.txt
nicodalessandro@uocniapr4 (2025-05-14):~$ cat camelliaPR4.txt
hello! this file will was encrypted with CAMELLIA256.
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg -c --cipher-algo CAMELLIA256 camelliaPR4.txt
[3258]@uocniapr4 - x CAMELLIA256 camelliaPR4.txt
Passphrase:
Enter passphrase
Password:
Confirm:

nicodalessandro@uocniapr4 (2025-05-14):~$ cat camelliaPR4.txt.gpg
hello! this file will was encrypted with CAMELLIA256.
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg -d camelliaPR4.txt.gpg
gpg: CAMELLIA256.CFB encrypted data
gpg: encrypted with 1 passphrase
hello! this file will was encrypted with CAMELLIA256.
nicodalessandro@uocniapr4 (2025-05-14):~$
```

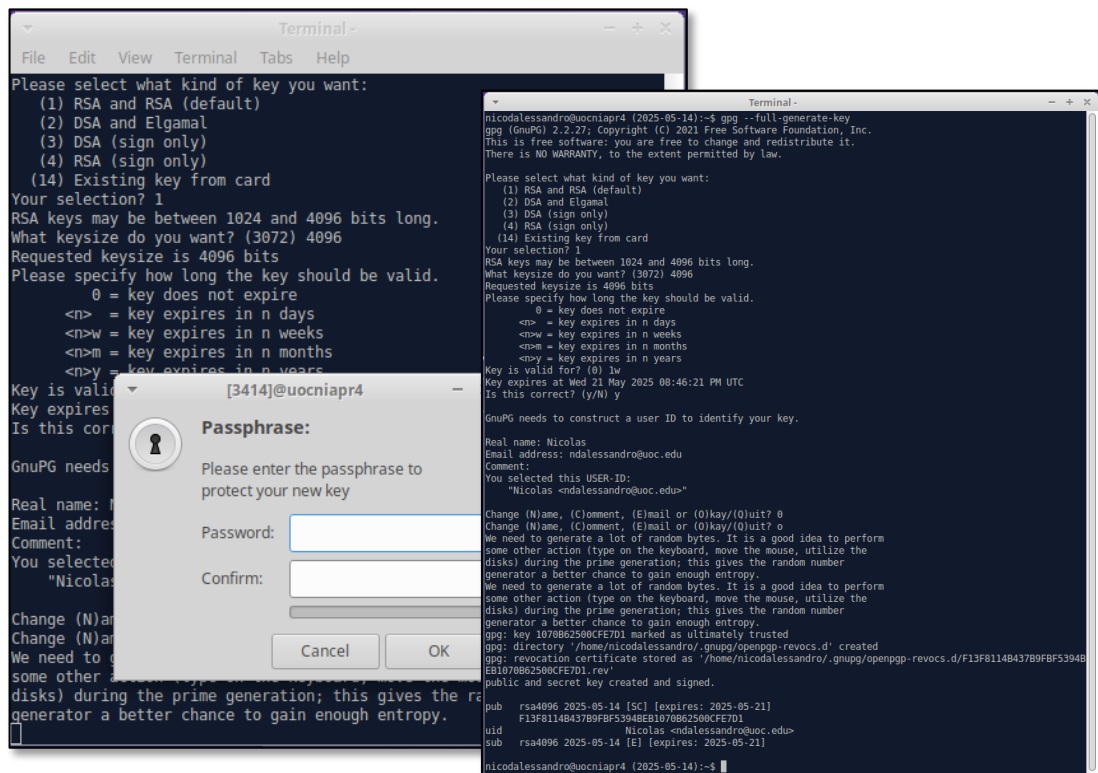
## Third Part (maximum qualification B): asymmetric encryption

### Exercise 3

a) We can observe that we have no keys generated:

```
Terminal-
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --list-keys
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --list-secret-keys
nicodalessandro@uocniapr4 (2025-05-14):~$
```

b) I used the command **gpg --full-generate-key** to generate the keys:



```
Terminal-
File Edit View Terminal Tabs Help
Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
(14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 1w
Key expires Wed 21 May 2025 08:46:21 PM UTC
Is this correct? (y/N) y
GnuPG needs to construct a user ID to identify your key.
Real name: Nicolas
Email address: ndalessandro@uoc.edu
Comment:
You selected this USER-ID:
"Nicolas <ndalessandro@uoc.edu>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(O)uit? 0
Change (N)ame, (C)omment, (E)mail or (O)kay/(O)uit? 0
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 1070862508CFE7D1 marked as ultimately trusted
gpg: directory '/home/nicodalessandro/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/nicodalessandro/.gnupg/openpgp-revocs.d/F13F81148437B9FBF53948EB1070862508CFE7D1.rev'
public and secret key created and signed.
pub rsa4096 2025-05-14 [SC] [expires: 2025-05-21]
F13F81148437B9FBF53948EB1070862508CFE7D1
uid Nicolas <ndalessandro@uoc.edu>
sub rsa4096 2025-05-14 [E] [expires: 2025-05-21]
nicodalessandro@uocniapr4 (2025-05-14):~$
```

Passphrase:

Please enter the passphrase to protect your new key

Password:

Confirm:

Cancel OK

Then I choose:

- Option 1 (RSA and RSA)
- Key size: 4096 bits
- Expiration: 1w (1 week)
- Name: Nicolas
- Email: ndalessandro@uoc.edu
- Comment: (empty)
- Confirm with **o** and entered the passphrase to protect the private key.

GPG successfully created the public and private keys.

c) I used **gpg --list-keys** to check the generated key. We can see:

**pub** (public key)  
**rsa4096** (a 4096-bit RSA key)  
 Created on May 14, 2025.  
**SC** (the key is valid for Signing and Certify)  
**E** (expires on May 21, 2025).  
 The **uid** is Nicolas ndalessandro@uoc.edu  
 sub (subkey generally used to encrypt E)

Additionally, I ran **gpg --list-private-keys**. This command shows the **secret keys** stored. These are the private keys that match the public keys you have generated. We should **never share** the secret key (It is used to **decrypt messages** and to **sign** them):

```

Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2025-05-21
/home/nicodalessandro/.gnupg/pubring.kbx
-----
pub   rsa4096 2025-05-14 [SC] [expires: 2025-05-21]
      F13F8114B437B9F8F53948EB1070862500CFE7D1
uid   [ultimate] Nicolas <ndalessandro@uoc.edu>
sub   rsa4096 2025-05-14 [E] [expires: 2025-05-21]

nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --list-secret-keys
/home/nicodalessandro/.gnupg/pubring.kbx
-----
sec   rsa4096 2025-05-14 [SC] [expires: 2025-05-21]
      F13F8114B437B9F8F53948EB1070862500CFE7D1
uid   [ultimate] Nicolas <ndalessandro@uoc.edu>
ssb   rsa4096 2025-05-14 [E] [expires: 2025-05-21]

nicodalessandro@uocniapr4 (2025-05-14):~$

```

## Exercise 4

a) To export the public key, I used this command:

**gpg --output ndalessandro.asc --armor --export ndalessandro@uoc.edu**

```

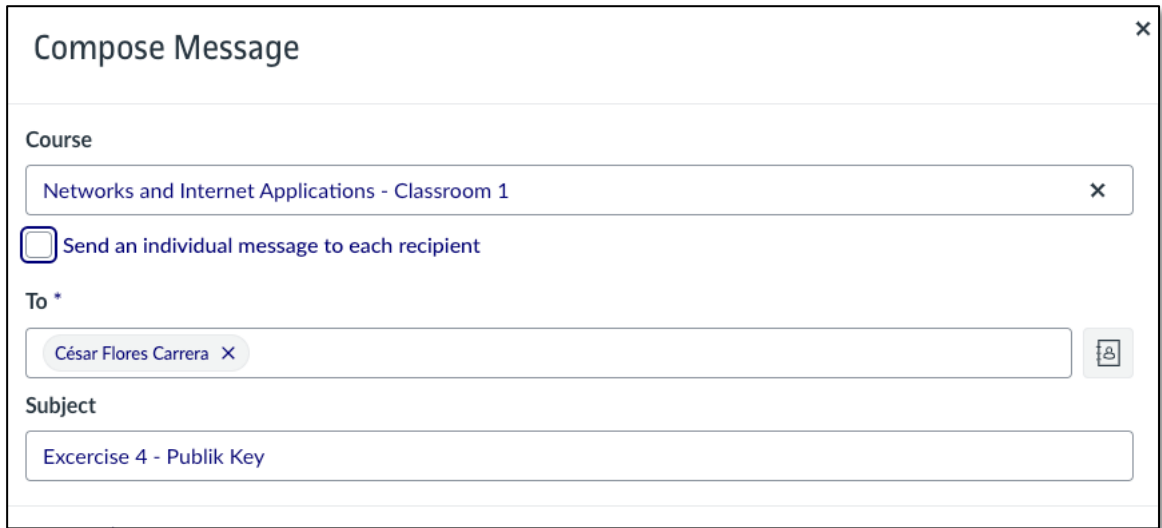
Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-14):~$ gpg --output ndalessandro.asc --armor
--export ndalessandro@uoc.edu
nicodalessandro@uocniapr4 (2025-05-14):~$ cat ndalessandro.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGgLAxsBEADzkLwqsvFHt6WRteRzzchLUXQhFSziUd+YR4cb1dh+W+jcA1sa
2iPVDtFcV2iIU1a8B//NvbC83h6Pfsa1+jCdI0omLRjp502SE2eJNe8+TFL0A6b
C4yo6p/V3o3BKP0Mofk9R0xwAW5vG50tW16DRCFjWbDnLRMXWaBp/fpm564sGE97
g0i2hCP0Cqm3+46mGjJJGtYqricDSVvtFnY1D/hd5p6Zud/1KV4oiF330EM88o3r
d4VQwE3TUMPYe5xGgw3ENGqnxZ37D8IG67wRjwF1hNoKd8YBpV3pBCLbIdIQdyU
BVhAN+0JXZ4bm80jsYSE8awllrM4U+qpJzx23bG1kxg5Y0GpAucTUIE+tLwIam1l
JkxU+VNNxldxxRnZeflJxeVp4ujg+W0YfdArkYNL/cRtp+s0KNZFR91A0SLBpkB6
te814PD5EwZV0cx1Ch3l18mPss2KsT/KgRMmbjdW+iVmZK7IKLw75psjKtTSiq0
G0/3r0GpM6UrnNmBpgHqI/rfZAbAMkM42Uu+YKE4GKfP1/5x7GbbVJ4Y3ydt9p9c4
t8GADtmJi5UaZJ/DvNQVij+Sbrf4TQI5EF81DcQYXYx+feKxCn7W+7puMu7D1fDS
lPFGADIXNN0x0d7mRBx0bqQ3l0eGBt1PQohqv3T2UwkQU0jnicQSDUDs4wARAQAB
tB50aWVnbGZlIDxuZGFsZXNzYm99cm9AdW9jLmVkdT6JALQEEwEKAD4WIQTxP4EU
tDe5+/U5s+sQcLYLAM/n0QUCaCUBewIbAwJAAK6gAULC0gHagYVCgkICwIEFgID
AQIeAQIXgAAKCRACLYLAM/n0Wz4D/9kaGF940AfEWOMGqct/U1q89atSsLiB1l
F+NS1lvZwF0241qs0yDVCtG4G0V6+UtgfNexSuWUaywtD9GsU4oXFZDQ0aREaE
1McDj54wdhJUm0ZT2W0ciC5g8Z0fjjzw+/o2FiFdJxwdZGkCA8tpt209gWama5Nr5
1Kp0sD9XNT1q16Afd1feyMqgdEL1oDkIarF00q/Bru441gADtsIGN/6SNI7L5Tv
z5xL5nHACWytX3dp4XQpnz5pN+Pn0tH4aoEHkNumHX89Qg70Nu8ud2LC2MBX8BK
uePoFMJ2/vc4F98p4wQyZFHjocGxvEMoGz/410/HyYb3miG6pjJs3c1ucjJVK0xP

```



- b) I sent this file to a classmate by email as an attachment.



## Exercise 5

- a) I have imported my colleague public key using the `gpg --import [colleague-file].asc`
- b) I verified that the key was imported with the command `gpg --list-keys`
- c) I have created an encrypted file for my colleague and used its id to encrypt with its public key.
- d) I sent the encrypted file by email.

## Exercise 6

- a) TODO
- b) TODO
- c) TODO

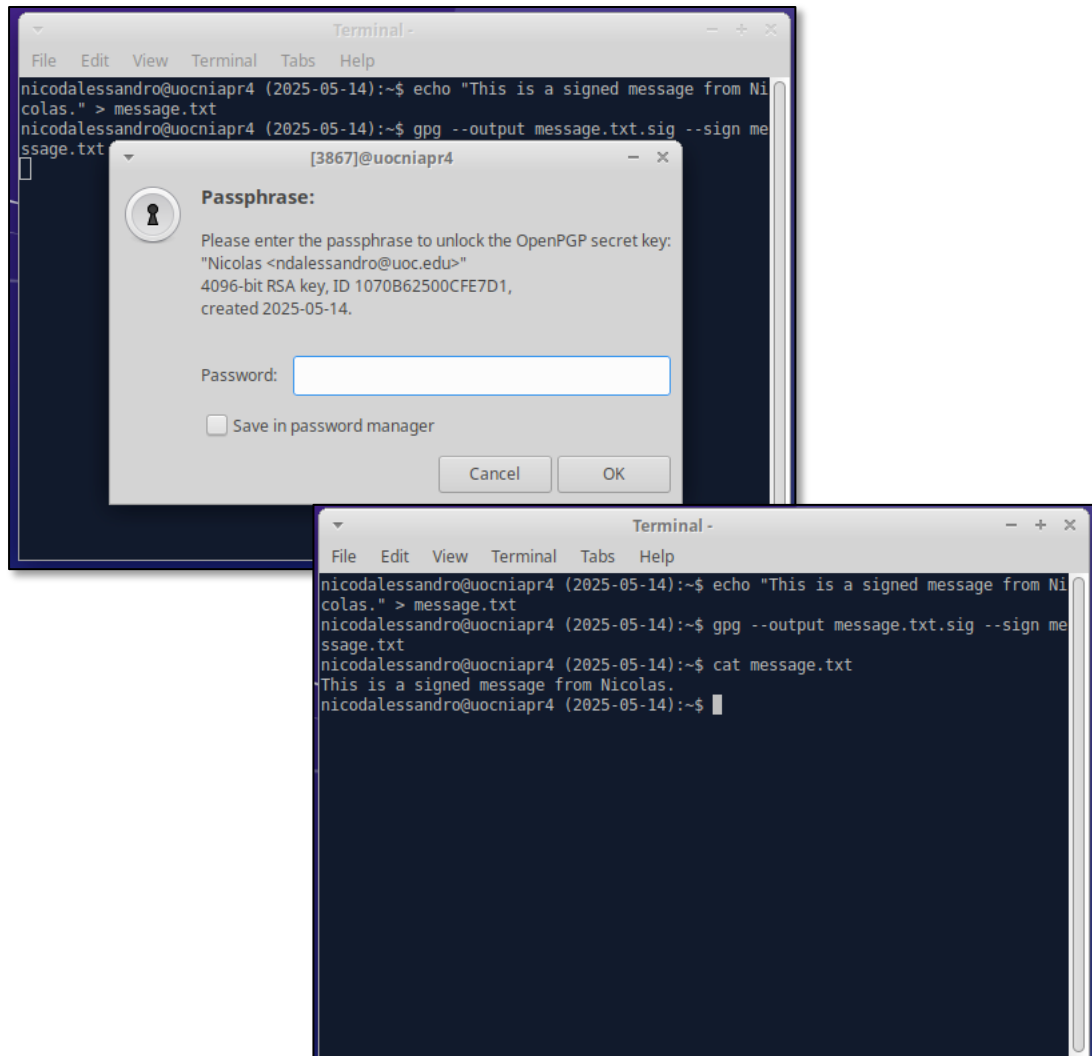
## Exercise 7

- a) To delete a public key from our key ring, we can run the command:  
`gpg --delete-key email@address.com`

If we also want to delete the private key, we can run:

`gpg --delete-secret-key email@address.com`

- b) To sign a message without encrypting it we can run the command **gpg --output message.txt.sig --sign message.txt**

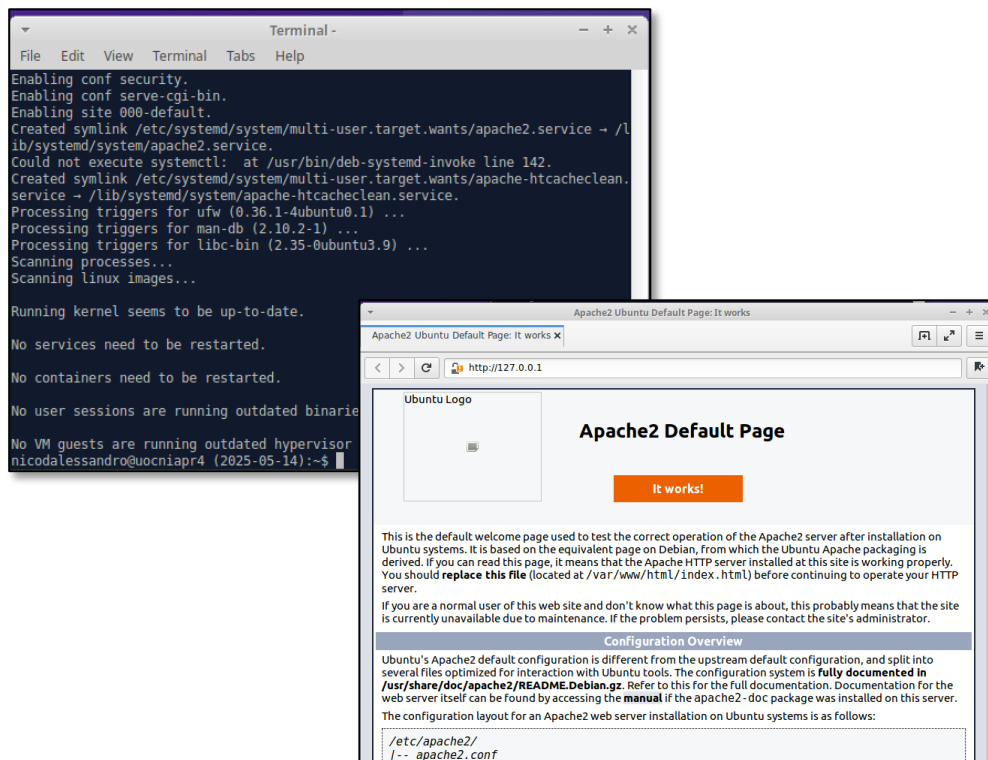


- c) It makes sense to sign a message without encrypting because it proves that I am the author and also confirms that the message has not been modified. Even anyone can read the message, they can verify that it is authentic. This is useful for public documents, announcements, or code that must be trusted (but not hidden or encrypted).

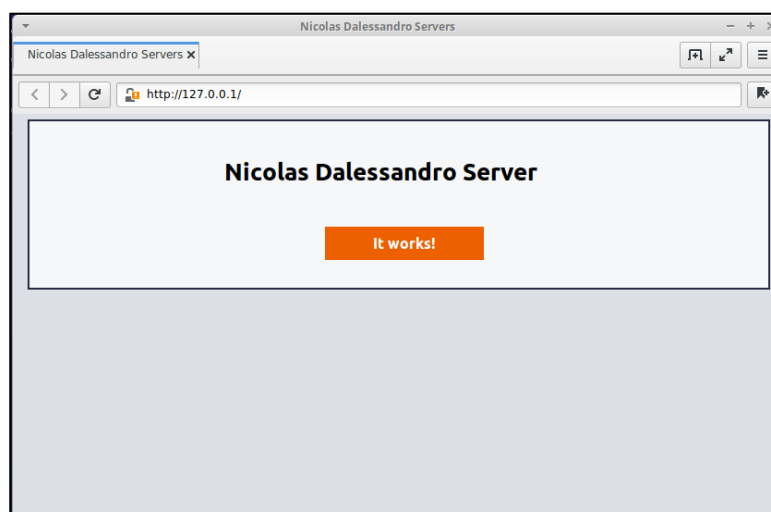
## Fourth Part (maximum qualification A): Apache secure configuration, client connection and SSL/TLS explanation

### Exercise 8

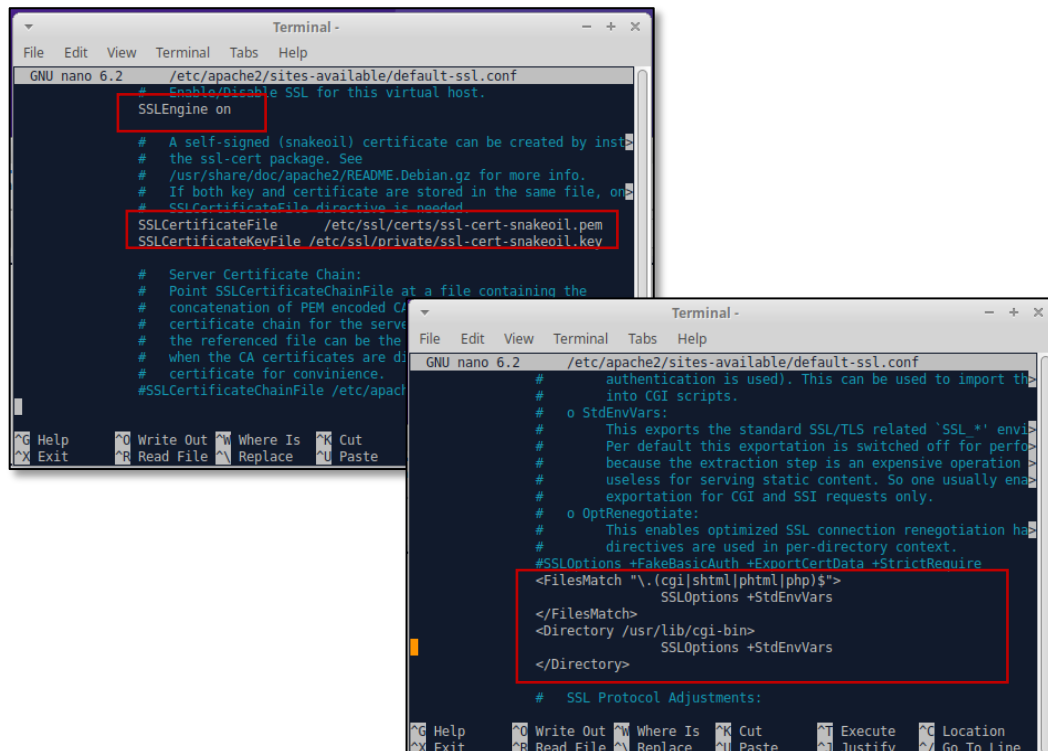
a) I installed Apache and check it is working:



b) I modified the default page to show the requested message:



### c) SSL directives in **default-ssl.conf**:



```

/etc/apache2/sites-available/default-ssl.conf
# Enable/Disable SSL for this virtual host.
SSLEngine on

# A self-signed (snakeoil) certificate can be created by installing
# the ssl-cert package. See
# /usr/share/doc/apache2/README.Debian.gz for more info.
# If both key and certificate are stored in the same file, one
# SSLCertificateFile directive is needed.
SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

# Server Certificate Chain:
# Point SSLCertificateChainFile at a file containing the
# concatenation of PEM encoded CA certificates that form the
# certificate chain for the server. The file can be the same as
# when the CA certificates are distributed.
#SSLCertificateChainFile /etc/ssl/certs/ssl-cert-snakeoil.pem

# This directive must be placed at the top of the configuration file,
# before the <VirtualHost> section, because <FilesMatch> and
# <Directory> directives are used in per-directory context.
#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire

<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
</Directory>

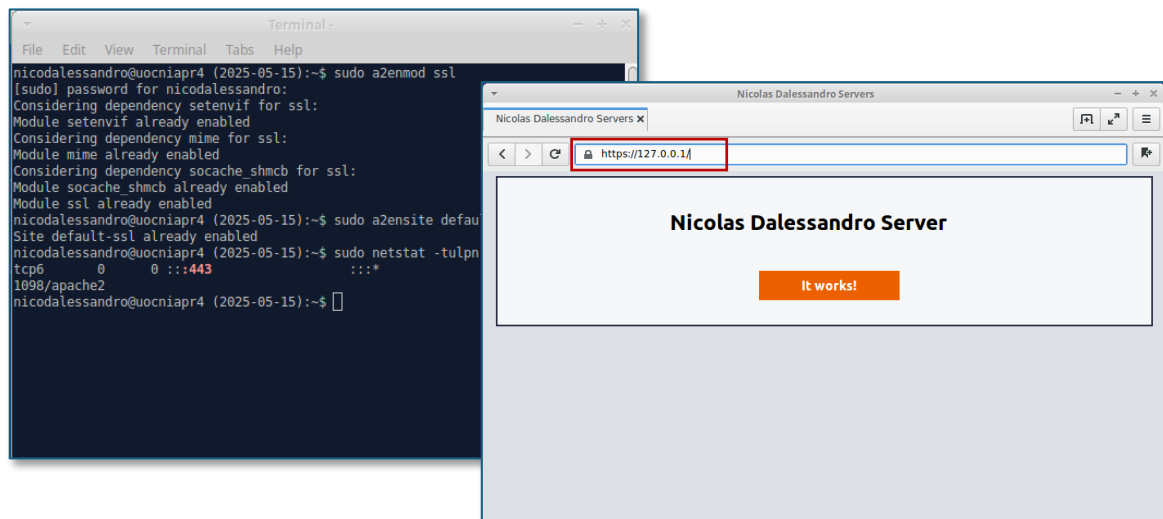
# SSL Protocol Adjustments:

```

We can observe:

1. **SSLEngine on** - Enables SSL/TLS for this virtual host.
2. **SSLCertificateFile** - Path to the certificate document used to encrypt the connections.
3. **SSLCertificateKeyFile** - Path to the private key that matches the certificate.
4. **<FilesMatch> and <Directory>** - Blocks to define environments and permissions for CGI scripts when using SSL (small programs that run on the web server).

### d) Enable HTTPS:



```

nicodalessandro@uocniapr4 (2025-05-15):~$ sudo a2enmod ssl
[sudo] password for nicodalessandro:
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
nicodalessandro@uocniapr4 (2025-05-15):~$ sudo a2ensite default-ssl
Site default-ssl already enabled
nicodalessandro@uocniapr4 (2025-05-15):~$ sudo netstat -tulnp
tcp6      0      0 :::443              :::*
1098/apache2
nicodalessandro@uocniapr4 (2025-05-15):~$

```

- e) Confirm with **netstat -ntl** that HTTPS is enabled:

```
Terminal -
File Edit View Terminal Tabs Help
nicodalessandro@uocniapr4 (2025-05-15):~$ netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp6       0      0 :::443                  :::*                     LISTEN
nicodalessandro@uocniapr4 (2025-05-15):~$
```

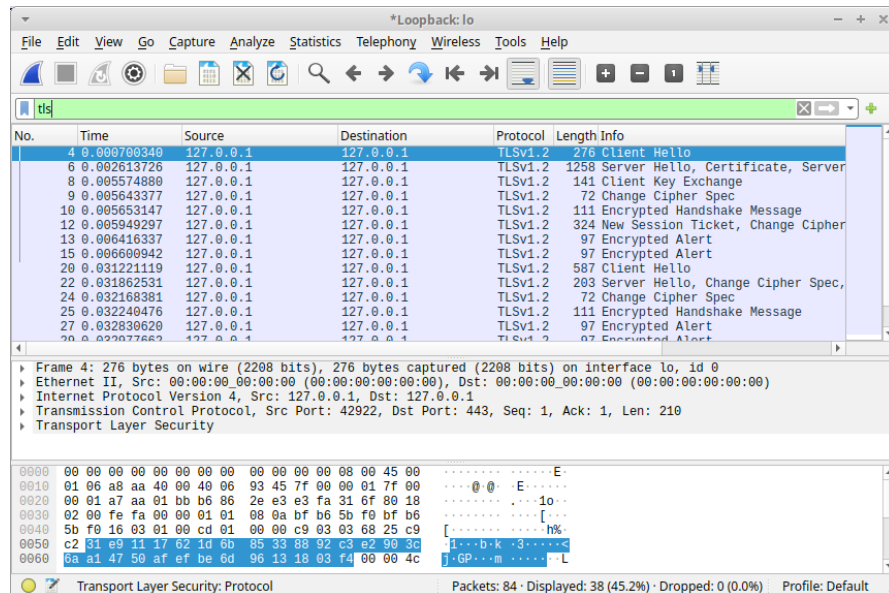
## Exercise 9

```
Terminal -
File Edit View Terminal Tabs Help
...
read R BLOCK
...
Post-Handshake New Session Ticket arrived:
SSL-Session:
  Protocol : TLSv1.3
  Cipher   : TLS_AES_256_GCM_SHA384
  Session-ID: E5A1E9AC44E17038EBC37DC188E8E66E472515BD175506AF9D6BE18CD7176884
  Session-ID-ctx:
  Resumption PSK: 1CC06701BC0D099A58C7F4168A7FCD49F0323032D08EF96DFFB8B344AF8C22167A67AD05F8E8E2D09EC1B7D14914A7A
  PSK identity: None
  PSK identity hint: None
  SRP username: None
  TLS session ticket lifetime hint: 300 (seconds)
  TLS session ticket:
0000 - 68 57 4b 19 f9 c3 38 da-2e f9 26 e6 6d 74 56 e1  hWk...8...&.mtV.
0010 - fd c0 cd 41 86 97 b6 d9-ca ab f7 4d 40 98 7b 1a  ...A.....M0.(.
0020 - 73 2e 6e c0 ec b6 50 e4-71 7f 49 e2 44 70 b0 65  s.n...P.q.IOp.e
0030 - 7d d1 d8 fa 98 f2 b5 ce-7c 42 63 55 b0 9b 9b 90  }.....|Bcll.6..
0040 - 23 62 b4 d7 48 21 f8 88-b2 3c 26 07 05 fc 2a 1f  #0..H1...<6...*
0050 - e8 31 38 0d a9 57 a8 70-c7 65 0c ce d5 e5 61 7f  .18..W.p.e....a
0060 - 2f 8a 20 d0 dd 35 b6 5f-44 0b 17 c3 0d fc cb f6  /. ..5..D.....
0070 - b3 c1 0b fd ea 39 7e 36-6f 2b c0 24 35 05 da 05  ....9-60+.$5...
0080 - df e3 3e 1d 5e 0c 95 b3-a0 6a 67 33 40 62 44 ce  .>...j9$800.
0090 - b0 65 5d 18 08 90 12 35-98 e6 d6 c7 6c 3b 55 57  .e]...5...l;LW
00a0 - 0e ce f9 fd 7c 17 ba 53-8c 1e d3 e3 de 66 2e 29  ....|.S.....f.)
00b0 - 92 57 d8 95 c6 45 13 4a-74 a9 a0 8d 75 17 52 01  .W...E..Jt...u.R.
00c0 - 4f dd f5 2c 1e 9e 84 ce-e4 52 46 cd 0c 4e cb 8d  0.....RF...N..
00d0 - 8c 02 ea fe 84 e0 28 72-63 06 68 74 19 3f 04 40  ....(rcfht.7.@
00e0 - f3 42 f2 63 45 67 dd b7-6b 6f ff 47 14 0d 95 92  .B.cEg...ko.G....

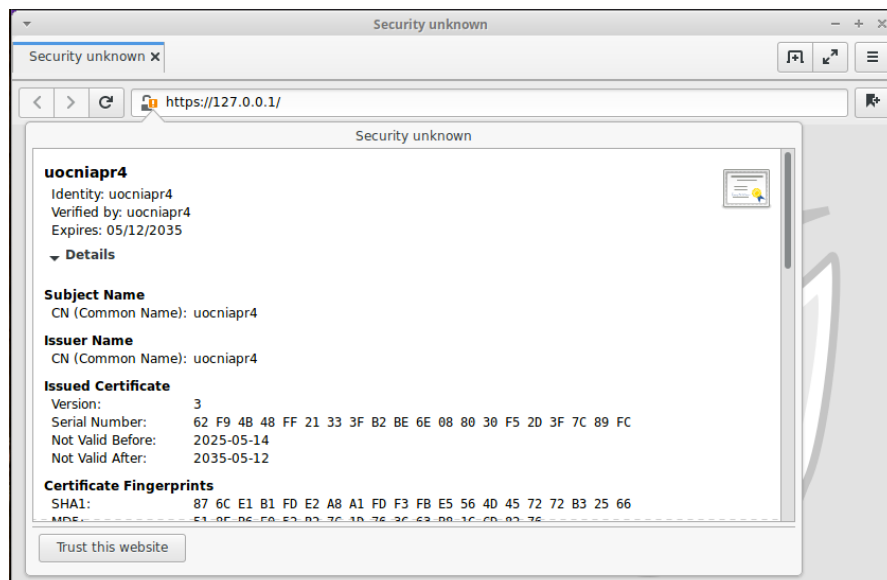
Start Time: 1747304583
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
...
read R BLOCK
closed
nicodalessandro@uocniapr4 (2025-05-15):~$
```

1. **Protocol (example Protocol : TLSv1.3)** - This shows the version of the TLS that was used in the connection. As we already know, TLS is the modern name for SSL. The 1.3 version is currently the latest and more secure. This means we are using the latest and more secure encryption protocol in this connection.
2. **Cipher (example Cipher : TLS\_AES\_256\_GCM\_SHA384)** - This shows the full elements of encryption algorithms that were used in this connection. We can see the **AES256** (AES encryption algorithm with a 256-bit key). **GCM** which means Galois/Counter Mode (a secure and fast way to encrypt blocks of data), and finally **SHA384**, which is a hash algorithm that is used for authentication and integrity to detect changes in data.
3. **Session ID (example Session-ID: 1DE87BBD507C...)** - This is the unique ID that was assigned to this session between the client and the server. If the same client connects relatively soon, the ID will allow for session reuse to make the connection faster, since it will avoid repeating the full TLS handshake, saving time and resources.
4. **STLS Session ticket (example 0000 - 76 00 b3 bc c4 14 0a c8 ...)** - This is a block of encrypted data that is being sent by the server. It contains all the session info (including the keys) so the client can resume the session later. It is kind of a saved "cookie" but for TLS session. This component is part of a feature called session resumption, that improves speed and performance in future connections but without losing security.

## Exercise 10



- a) As we can observe, when opening the localhost, the browser shows a “Security unknown” because the certificate was not signed by a trusted Certificate Authority or CA, but instead self-signed by the server (CN = ucniapr4), hence requesting me to trust it manually:

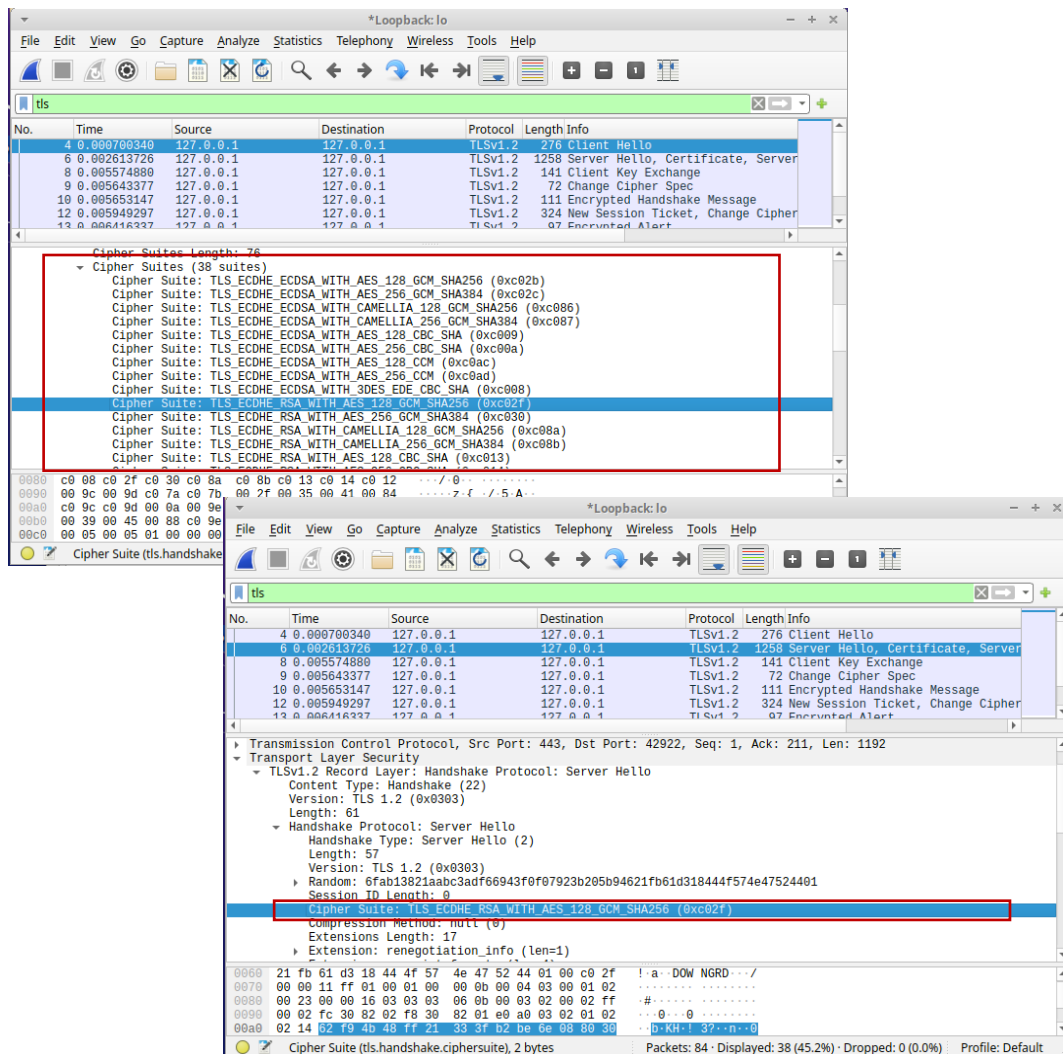


- b) When using Wireshark (as we can see in the image above) I filtered the TLS packages and identify the following algorithms for the TLS 1.2 protocol used.

We can observe in the Server Hello response to the Client Hello options, that the algorithm used in the connection was:

### TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

- Key exchange: ECDHE
- Authentication: RSA
- Encryption: AES 128 in GCM mode
- Integrity: SHA256



The image displays a Wireshark capture of a TLS handshake. The top pane shows the packet list with the following entries:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000700340	127.0.0.1	127.0.0.1	TLSv1.2	276	Client Hello
6	0.002613726	127.0.0.1	127.0.0.1	TLSv1.2	1258	Server Hello, Certificate, Server Key Exchange, Change Cipher Spec, Encrypted Handshake Message
8	0.005574880	127.0.0.1	127.0.0.1	TLSv1.2	141	Client Key Exchange
9	0.005643377	127.0.0.1	127.0.0.1	TLSv1.2	72	Change Cipher Spec
10	0.005653147	127.0.0.1	127.0.0.1	TLSv1.2	111	Encrypted Handshake Message
12	0.005949297	127.0.0.1	127.0.0.1	TLSv1.2	324	New Session Ticket, Change Cipher Spec, Encrypted Alert

The middle pane shows the Cipher Suites (38 suites) list. The selected cipher suite is TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02f).

The bottom pane shows the TLSv1.2 Record Layer details. The Handshake Protocol: Server Hello is expanded, showing the following details:

- Handshake Type: Server Hello (2)
- Length: 57
- Version: TLS 1.2 (0x0303)
- Random: 6fab13821aabc3adcf66943f0f07923b205b94621fb61d318444f574e47524401
- Session ID Length: 0
- Cipher Suite: TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02f)
- Compression Method: null (0)
- Extensions Length: 17
- Extension: renegotiation\_info (len=1)

c) We can clearly see in the image the three main steps of the TLS handshake:

4	0.000700340	127.0.0.1	127.0.0.1	TLSv1.2	276 Client Hello
6	0.002613726	127.0.0.1	127.0.0.1	TLSv1.2	1258 Server Hello, Certificate, Server
8	0.005574880	127.0.0.1	127.0.0.1	TLSv1.2	141 Client Key Exchange
9	0.005643377	127.0.0.1	127.0.0.1	TLSv1.2	72 Change Cipher Spec
10	0.005653147	127.0.0.1	127.0.0.1	TLSv1.2	111 Encrypted Handshake Message
12	0.005949297	127.0.0.1	127.0.0.1	TLSv1.2	324 New Session Ticket, Change Cipher

1. **Client Hello:** The client says “hello” to the server and send the supported TLS version and cipher suites (Wireshark capture line 4)
2. **Server Hello + Certificate:** The server replies with the selected cipher and the certificate. This certificate includes the public key and the identity (Wireshark line 6)
3. **Key Exchange + Finished:** The client sends the key exchange and change the cipher spec. Then, the server responds with it own encrypted messages. Finally, the encrypted communication begins (Wireshark capture lines 8, 9, 10 and 12).

d) A self-signed certificate offers some protection but not all:

1. What it guarantees (Encryption): The data is private, and it is encrypted between the browser and the server. This means that no one can read it if they are watching the network.
2. What it does not guarantee (Identity - Trust): We don't know who the owner of the website is, and since anyone can create a certificate with any name, the browser does not trust the certificate (as we already explained is not signed by a trusted company (Certificate Authority)).

In summary, even a self-signed certificate may be good for this learning purposes, development or personal servers it won't be safe for the public without some extra steps.