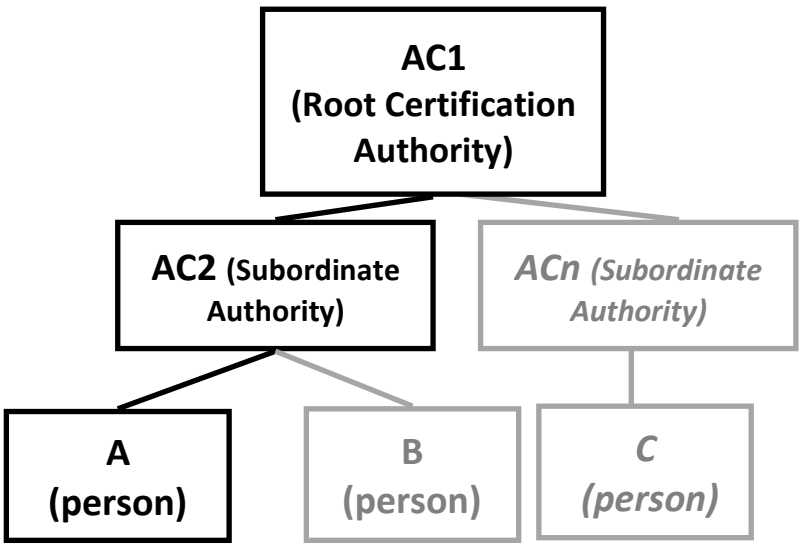
Laboratory 1. Demo PKI with OpenSSL

Hierarchical trust model

The objective of this practice is to understand the concepts underlying a public key infrastructure based on the hierarchical trust model.
Specifically, the objectives are the following:

- Understand required steps in order to create a mini PKI
- Understand required steps in order to an Authority issuing a certificate
- Understand what the certificate role is regarding the signing and verifying of documents

To achieve these objectives each student (or student group) becomes a ROOT CERTIFICATION AUTHORITY (equal to the role of "Fábrica Nacional de Moneda y Timbre" in the real world). Such Authority (AC1), according to organizational reasons (for example, to have a local office in all different districts), has some SUBORDINATE CERTIFICATION AUTHORITIES (AC2, AC3,…ACn). Moreover, these last Authorities are in charge of issuing public key certificates to people (A, B, C…)
The group of all the Authorities composes a Public Key Infrastructure (PKI).



To reduce the amount of work, in this assignment it is only managed the root certification authority (AC1), a single subordinate authority (AC2) and a person (A).

Description and preparation of the working environment

Linux Environment

OpenSSL is usually installed in Linux distributions.

Open a console, and execute -help command of openssl:

```
#currentpath> openssl -help
```

After executing the openssl -help command, you'll see in the screen the summary of the commands offered by the tool.

Windows Environment

It is possible to perform the lab assignment on a Windows machine. You can download the pre-compiled packages from https://ftp.openbsd.org/pub/OpenBSD/LibreSSL/libressl-2.5.5-windows.zip. If you have administrator privileges in the machine, you can install OpenSSL from https://wiki.openssl.org/index.php/Binaries .

Additionally, you have to make the following changes regarding the commands provided in each step:

- The bar used in Windows is inclined to the left: "\"
- When using echo command, you must not include the quotation marks around 01, instead use echo as follows: "cmd> echo 01 > serial"
- There is no command "touch", but instead you can create an index.txt file directly in the explorer (using mouse right click button and New file…). Do not open the file. Confirm that the name of the file is index.txt and not index.txt.txt
- Copy command is not "cp" but "copy"
- File concatenation is not "cat" but "type": "cmd> type file1 file2 > file3"

Unzip the file in a folder inside Documents folder (or any other folder you have write privilege).

Open a console (cmd).

Once in the console, to execute openssl without any other change, you have to execute the openssl executable by writing the full path to the exe file. Check everything is working by executing -help command:

```
#currentpath> folder-where-pack-was-unzip\x64\openssl -help
```

For example:

```
folder-where-pack-was-unzip == "C:\Users\2003450188\Documents\libressl"
```

After executing the openssl -help command, you'll see in the screen the summary of the commands offered by the tool.

Configuration files

Download ficheros.zip file. Unzip it. You'll find two configuration files, one for each of the AC considered in the lab.

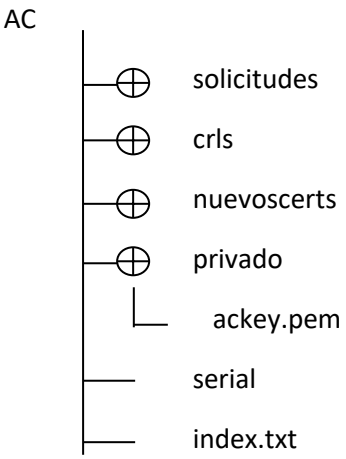
Directories

Create a lab base folder. To help with the organization of the practice, **create inside the previous foder 3 directories**, one for each entity**: AC1, AC2 and A** .

```
# assignment> mkdir AC1 AC2 A
```

In order to issue the certificates, the Authorities use a certification POLICY. Copy the configuration files `openssl_AC1.cnf` and `openssl_AC2.cnf` to the corresponding directories AC1 and AC2.

First, files and folders must have next structure. Guidance for creating it is provided later.



OpenSSL help

https://man.openbsd.org/openssl.1

https://www.openssl.org/docs/man1.1.1/

Creating your own mini-PKI

Important: We are going to use the OpenSSL command line tool. Pay special attention to the exact folder where the pointed out commands are executing.

Configuration of AC1 (root AC)

We will apply the following commands: `ca, req, x509 and verify`.

The **ca** command is a minimal Certification Authority application. It can be used to sign certificate requests in a variety of forms. Its characteristics can be established by means of the file `openssl.cnf`. Moreover, we must prepare the working environment as a specific directory structure.

The **req** command primarily creates and processes certificate requests. It can additionally create self signed certificates for use as root CAs for example.

Command **x509** can be used to display certificate information, convert certificates to various forms, and sign certificate requests.

Verify command verifies certificate chains.

1. Generate the directory structure necessary for AC1 and initialize the files `serial` and `index.txt`.

```
# AC1> mkdir solicitudes crls nuevoscerts privado
# AC1> echo '01' > serial
# AC1> touch index.txt
```

2. Generate the RSA key-pair and the **self signed** certificate for AC1. Analyze the output.

```
# AC1> openssl req -x509 -newkey rsa:2048 -days 360 -out ac1cert.pem -outform PEM -config openssl_AC1.cnf
```

The command requests a passphrase to generate AC1 private key. Remember it when you want to use this key.

```
# AC1> openssl x509 -in ac1cert.pem -text -noout
```

Configuration of AC2 (subordinate AC)

3. Generate the directory structure necessary for AC2 and initialize the files `serial` and `index.txt`.

```
# AC2> mkdir solicitudes crls nuevoscerts privado
# AC2> echo '01' > serial
# AC2> touch index.txt
```

4. Generate the RSA key-pair for AC2 and the certificate request which will be sent to AC1 and 'send' it to AC1. Analyze the results.

```
# AC2> openssl req -newkey rsa:2048 -days 360 -out ac2req.pem
-outform PEM -config openssl_AC2.cnf
```

The command requests a passphrase to generate AC2 private key. Remember it when you want to use this key.

```
# AC2> openssl req -in ac2req.pem -text -noout
# AC2> cp ac2req.pem ../AC1/solicitudes
```

Generation of AC2's certificate by AC1

5. Verify the request "sent" by AC2.

```
# AC1> openssl req -in ./solicitudes/ac2req.pem -text -noout
```

6. Generate the corresponding certificate for AC2 and 'send' it back to AC2. Rename 01.pem into `ac2cert.pem`, because AC2 has this name in its configuration file.

```
# AC1> openssl ca -in ./solicitudes/ac2req.pem -notext -extensions
v3_subca -config openssl_AC1.cnf
```

AC1 needs its private key to generate AC2 certificate. AC1 passphrase is request

```
# AC1> cp ./nuevoscerts/01.pem ../AC2/ac2cert.pem
```

Requesting an end entity's certificate

Generation of keys for entity A as well as its corresponding certificate request

7. For entity A, generate an RSA key-pair as well as a certificate request and "send" it to AC2 (when generating the certificate requests, fill in ALL the requested fields and indicate "ES" as country, "MADRID" as province, "UC3M" as organization, and any common name (eg, one NIA) and the email is your **student email**.

```
# A> openssl req -newkey rsa:1024 -days 360 -sha1 -keyout Akey.pem
-out Areq.pem
```

The command requests a passphrase to generate A private key. Remember it when you want to use this key.

```
# A> openssl req -in Areq.pem -text -noout
```

```
# A> cp Areq.pem ../AC2/solicitudes
```

Generation of A certificate by AC2

8. Verify the request "sent" by A.

```
# AC2> openssl req -in ./solicitudes/Areq.pem -text -noout
```

9. Generate certificate for A and "send" it back to this entity.

```
# AC2> openssl ca -in ./solicitudes/Areq.pem -notext -config
./openssl_AC2.cnf
```

AC2 needs its private key to generate A certificate. AC2 passphrase is request

```
# AC2> cp ./nuevoscerts/01.pem ../A/Acert.pem
```

10. Analyze changes in AC2 directory and check the resulting certificate:

```
# A> openssl x509 -in Acert.pem -text -noout
```

Verification of A certificate

11. Obtain a copy of the public key certificates of AC1 and AC2 and verify (you need to concatenate both AC1 and AC2 certificates in a single file).

```
# A> cp ../AC1/ac1cert.pem ./
```

```
# A> cp ../AC2/ac2cert.pem ./
```

```
# A> cat ac1cert.pem ac2cert.pem > certs.pem
```

```
# A> openssl verify -CAfile certs.pem Acert.pem
```

Joining the certificate and the private key to sign in common applications (Word/ Email)

12. **Export** the certificate of entity A, its private key and both AC1 and AC2 certificates (file certs.pem) to PKCS12 format.

```
# A> openssl pkcs12 -export -in Acert.pem -inkey Akey.pem -certfile
certs.pem -out Acert.p12
```

A passphrase is request to export A private key, and a new passphrase is request for Acert.p12

Use of A's private key to sign a document

13. Create a Microsoft Word Office document and sign it electronically signed using A private key. In order to do that, you have to import "Acert.p12" in the browser (**Tools > Internet options > Content > Certificates > Import…**) and, then, using Microsoft Word make use of **"Office button">Prepare> Add digital signature…**

Questions

a) What is the file “serial” used for?

b) What is the file “index” used for?

c) Could AC2 create a certificate applying step 2 of this script?

d) If you or your lab group become a Certification Authority, explain and justify (i.e., advantages, disadvantages, alternatives…) the values you would use to configure the following parameters: default_days, default_crl_days, countryName

e) When you open the Word document, once signed, you may notice that a it gives a “Verification error”. Why does it happen? How can it be solved?