(Applied) Cryptography

Week #12 Tutorial

Manuel Barbosa (mbb@fc.up.pt) Rogério Reis (rvreis@fc.up.pt)

MSI/MCC/MERSI – 2022/2023

Overview

In this class we will go through the process of building a PKI with two hierarchical levels.

The lecturer manages a root CA (see attached `root_ca.pem` certificate) and each group will manage a subordinate CA. Using this subordinate CA, each group will be able to issue personal certificates for all members and/or issue server certificates for Web servers.

The final goal is that students will be able to experiment with standard tools configured for compatibility with this PKI:

- to communicate with each other using signed and encrypted email and/or
- to establish fully authenticated TLS sessions (where both client and server are authenticated using public-key signatures) with Web servers that they manage.

Note that compatibility and interoperability between subordinate CAs of different groups will be guaranteed because they are all under the same root CA.

Students are encouraged to verify that they are indeed able to communicate seaminglessly regardless of the subordinate CA that issued the certificates. In particular, a student from one group can ask for a personal certificate to be issued by another group.

Certification process

The process of obtaining a X.509 certificate includes (at least) the following steps:

- The (future) certificate owner generates a key pair
- The (future) certificate owner creates a *certificate request* that contains the identification information and public key that will be included in the certificate; this is signed with the generated secret key (and proves possession of the secret key)
- The certification authority checks the data in the request and if no problems arise, it issues the certificate by signing it with the CA secret key.
- The certificate owner obtains the newly signed certificate and may now provide it to whoever needs to use the enclosed public key.

Note that the owner of the certificate can be a human user, an application, a server or even another certification authority.

The next steps are presented for the particular setting in which a group of students interacts with the lecturer, who manages the root CA, to obtain a public-key certificate for the subordinate CA that they will manage.

In each step we will use `openssl` to execute a necessary command. Whenever we use `XPTO` this represents an identification string that identifies the key owner (in this concrete example groups can use their Moodle identifier).

Note that essentially the same commands can be used by anyone to create a certificate request that subsequently can be used to obtain a personal or server certificate from one of the subordinate CAs.

Key pair generation and certificate request creation

To generate a key pair the future certificate owner can run

```
openssl genrsa -out groupXPTO.key 2048
```

Note: Be extremely careful not to overwrite this secret key: it will be the secret key of the subordinate CA!

Note: This secret key is **not** a personal secret key; it belongs to the group CA and should **not** be used for anything else.

Each group should construct a certificate request for the group subordinate CA by executing the following command:

```
openssl req -new -key groupXPTO.key -out newreq.pem
```

It is **mandatory** to use the following attributes for the CA identifier:

- `countryName` = `PT`
- `organizationName` = `Universidade do Porto`
- `organizationalUnitName` = `DCC FCUP`

The `commonName` should start with `SubCA` followed by the group identifier.

The resulting file `newreq.pem` should be sent to the lecturer (mbb@fc.up.pt). The lecturer will answer with the newly issued certificate `newcert.pem`.

Creating the subordinate CA

Certificate generation requires a verbose set of commands; we will be using a `perl` script that is distributed with `openssl` called `CA.pl`.

To create a new CA, move the `CA.pl` script provided with the `openssl` distribution and the received certificate `newcert.pem` to a new empty folder where the CA files will be located and run the following command:

```
./CA.pl -newca
```

Note: Some `openssl` distributions come with a minimalistic configuration that does not include CA options. An example `openssl` configuration file is provided for these cases. `CA.pl` needs to be edited to used this configuration file.

The tool will ask whether a certificate has already been issued for this CA: if this were not the case `openssl` would assume that this is a new root CA and it would create a new self-signed certificate.

In this case the group already obtained the subordinate CA certificate, so simply indicate `newcert.pem`.

Note: A new sub-folder `demoCA` will be created that will be managed by `openssl`. All CA management commands should be run from the current folder that contains the `CA.pl` script.

Before using the CA, the following steps should be concluded manually:

- move the subordinate CA key `groupXPTO.key` to sub-folder `demoCA/private` where it should *replace* the existing (empty) file `cakey.pem`.
- create a text file `demoCA/serial.txt` (in some versions with no extension) with content `00` (or any number in hexadecimal that will be used as the serial number of the first issued certificate).

Issuing public-key certificates

From this point onwards the group can generate new certificates using the subordinate CA.

Note: The subordinate CA is not an on-line entity that should interact with other applications or use certificates in any way. Its sole purpose it to be used as a command-line tool to create new certificates for new key pairs!

Given a new certificate request `newreq.pem` containing a fresh public-key for, e.g., a human user or a Web server, one can generate a certificate by:

- moving `newreq.pem` into the folder holding `CA.pl`
- running `./CA.pl -sign`

This will create `newcert.pem` in the same folder.

To check that a certificate was correctly generated by the subordinate CA, simply run:

```
openssl verify -CAfile root_ca.pem  -untrusted demoCA/cacert.pem newcert.pem
```

Note: This verifies the full certificate chain: the first argument gives an *implicitly trusted* certificate for the root CA and the intermediate certificate for the subordinate CA is declared to be *untrusted*.

Validation

To verify that you have correctly understood all the concepts, you can now check that you can use your group's subordinate CA to generate personal certificates for yourself or your colleagues (from any group). Alternatively you can generate server certificates for Web servers that you manage.

Note: Before email clients and browsers are able to validate certificates in this new PKI the root CA certificate must be imported and declared to be trusted. *This should be done in a test machine, or you should take care to uninstall it after you complete the exercises!*

You can check that things are working correctly if email clients accept the personal certificates and are able to encrypt/decrypt/sign/verify email and if browsers do not issue warnings when interacting with your Web server.

Note: For email, the personal certificates must include the special attribute `email` in the distinguished name and this *must* match the email account you will be using.

Note: In OSX all certificates are managed centrally in **KeyChain** and email certificates *must* include the S/MIME required extensions. You can configure `openssl` to include these extensions when signing a certificate by creating your own `openssl.cnf` file.

Personal certificates can be installed in browsers and email clients by packing them together with the secret key and certificate chain in a PKCS12 file (you have encountered such files in a previous tutorial).

To create a PKCS12 package, you can use the following command:

```
openssl pkcs12 -export -chain -CAfile <chain.pem>  -name <keyalias> -aes128 -inkey <privatekey.key>
-in newcert.pem -out <myp12package.p12>
```

Here, `chain.pem` should contain *both* the `root_ca.pem` certificate *and* the subordinate certificate `demoCA/cacert.pem`.