SIO Lab
X.509 Certificates

version 1.0

Author: João Paulo Barraca, Hélder Gomes, André Zúquete, Vitor Cunha

Version log:

- 1.0: Initial version

1 Introduction

Public Key certificates comply with the X.509v3 standard. They are managed in the context of a PKI (Public Key Infrastructure) that defines the policies under which the certificates may be used. The IETF PKIX Working Group[1] produces most standards regulating the use of X.509 certificates on the Internet.

In this guide, you will focus on the certificate validation processes. For creating Certification Authorities and certificates, explore the `xca`[2] tool (GUI) or look at the syntax of `openssl x509` (command line). A certificate is considered valid for a **specific purpose** if the following conditions are verified: (i) if the certificate is used during its validity interval, (ii) if the certificate was issued (signed) by an entity (CA) in which the user trusts and (iii) if the policies in the certificate allow its use for the intended specific purpose. Here, we are going to handle the first two conditions. For additional information on the subject of this guide, you may consult the Python Cryptography X509 PKI Tutorial[3].

You must obtain an X.509 Public Key Certificate to complete this guide. These certificates are widely available on HTTPS-enabled websites. You can obtain the certificate from any website. The following command exemplifies how to obtain the certificates from the university's main webpage:

```
openssl s_client -connect www.ua.pt:443 -showcerts < /dev/null
```

The example output should show you two certificates, that you can identify by the `BEGIN CERTIFICATE` and `END CERTIFICATE` lines:

```
-----BEGIN CERTIFICATE-----
MIIG5TCCBM2gAwIBAgIRANpDvRObOli7TdYcrMTz2+AwDQYJKoZIhvcNAQEMBQAw
gYgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpOZXcgSmVyc2V5MRQwEgYDVQQHEwtK
ZXJzZXkgQ2l0eTEeMBwGA1UEChMVVGhlIFVTRVJUUlVTVCBOZXR3b3JrMS4wLAYD
(...)
vBYIi1k2ThVh0Dx88BbF9YiP84dd8Fkn5wbE6FxXYJ287qfRTgmhePecPc73Yrzt
apdRcsKVGkOpaTIJP/l+lAHRLZxk/dUtyN95G++bOSQqnOCpVPabUGl2E/OEyFrp
Ipwgu2L/WJclvd6g+ZA/iWkLSMcpnFb+uX6QBqvD6+RNxul1FaB5iHY=
-----END CERTIFICATE-----
```

If you copy and paste each certificate to a file, you can see more certificate details with the command:

```
openssl x509 -text -in <certificate_file>
```

[1] Public-Key Infrastructure (X.509) (pkix)

[2] XCA (install with: `apt install xca`)

[3] X.509 Tutorial

We will explore later in the guide why, in this example, you get two certificates from the server.

X.509 Public Key Certificates are also present in the Portuguese Citizen Card, and you can create your own Certification Authority and certificates with XCA (useful for testing). The code and patterns you will develop in this guide will also apply in those contexts.

2 Certificate Validity Interval

X.509 public key certificates are not perpetually valid. That is, a temporal validity interval will always define when the certificate may be used, and that is a basic validity check that must always be performed.

Task: Implement a small program that reads a certificate and a function that verifies its validity **at this moment** by analyzing the attributes `not_valid_after` and `not_valid_before`.

We advise storing the loaded certificates in a dictionary where the key is each certificate's subject. This option will speed up the implementation of the remaining guide.

HINT: In Python, use the `cryptography.x509` class[4] to load the certificate and then check its attributes.

3 Trust Anchor certificates

Trust anchor certificates are the certificates the user (or application) already trusts. These are typically root certificates (i.e., self-signed certificates). Trust anchor certificates are essential to validate certification paths (certificate chains) because these define when you have reached trust through transitive properties. A valid certification path can only be trusted if a trust anchor certificate exists in that path. You may encounter a valid certification path that is not trusted. Trust management and anchor certificates are essential for establishing trust in X.509.

3.1 Reading trust anchor certificates

Usually, trust anchor certificates are provided by the operating system or the application in use (e.g., Firefox, Chrome, etc.). Sometimes, the user may also provide new trust anchors, depending on the application and the security model. The trust anchor certificates must be protected in a `keystore` or a restricted location (`/etc/ssl/certs`) to prevent unwanted additions or removals, be these intentional (e.g., adversarial) or unintentional (i.e., accidents). You can use the certificates in your Linux system as individual files containing certificates in the PEM format.

Task: Implement a small program that reads all system-trusted certificates into a dictionary of **trusted certificates**, with the subject as the key.

Do not load certificates that have already expired (use the previously developed function).

HINT: Use the `os.scandir` object to scan for all certificates in `/etc/ssl/certs`.

4 Build a certification path

A certificate chain, or certification path, is the set of certificates that composes a chain of trust, from a trust anchor certificate to an end user certificate. Frequently, to validate an end-user certificate, it is necessary to build the certification path from a set of several candidate intermediary certificates. Other times, the entity to be validated (e.g., web server) provides the respective certification path with its certificate.

Task: For this exercise, take each **user-provided certificate** (from the Citizen's Card, XCA, downloaded, etc.) and recreate its validation chain in a list. To make the chain, load the user certificate, a dictionary of user-specified intermediate authority certificates, and the roots.

[4] X.509 Reference

You should stop when you get a root certificate (i.e., self-signed). The root certificate can be missing from that list. That usually means the root certificate is already in your trusted certificates. Otherwise, the chain will be untrusted unless you already trust the user or intermediate certificates.

HINT: Remember that the user and root certificates are loaded into a dictionary with the subject as the key. Therefore, it is simple to obtain the issuer of a certificate.

5 Validate a certification path

Given a certification path, we can validate it by verifying each certificate and the relations between certificates.

Task: As the first step, create a function that validates the revocation status of each certificate in the chain. Validation should be made using available CRL, Delta-CRL, or OCSP. Each certificate specifies the validation methods and endpoints to use. A certificate can support all or only some methods.

HINT: The CRL file can be downloaded and loaded using the `x509.load_pen_x509_crl` method. A list of Revoked Certificates comprises it. You can search this list for the certificate under validation.

Task: The next step will be to validate the signatures in the chain. Each certificate should have a signature (`x509.signature` field) created with the issuer's private key. Use the issuer `x509.public_key` to validate the signature. The signed data is present in the attribute `x509.tbs_certificate_bytes`.

Validating the certificates also includes checking their purpose, common name, and validity interval. Build a function that checks these attributes, then call the previously developed functions to validate the certificate chain further.

NOTE: You can use Wireshark to verify the messages exchanged with the OCSP servers to get the revocation status of certificates.

6 To explore

You can do several other tasks to explore how the X509 certificates are used in usual systems. Some are:

1. Alter your program only to use the CRL mechanism for revocation checks when validating the certification path.
2. Alter your program to use OCSP first and then the CRL if the OCSP mechanism fails.
3. Add caching mechanisms that store OCSP and CRL responses while they are valid.

7 Closing note

Establishing proper time is critical for X.509 validations. You cannot confidently validate the status at a past time unless you can establish fully trusted timestamps for each moment (i.e., you must trust all clocks before validating the paths). Otherwise, there is no other time like the present.