Information and Organisational Security

Guides for Practical Classes

João Paulo Barraca and Vitor Cunha

Department of Electronics, Telecommunications and Informatics
University of Aveiro

2020-2021

4

Certificate Validation

Topics:

- Certificate validity interval
- Reading certificates into and from keystores
- Building certification paths
- Validation of certification paths
- Validation using OCSP and CRL revocation mechanisms.

4.1 Introduction

Public Key certificates comply with the X.509v3 standard and are managed in the context of a PKI (Public Key Infrastructure), that define the policies under which the certificates may be used. Most of the standards that regulate the use of X.509 certificates in the Internet are produced by the IETF PKIX Working Group[1].

In this guide you will focus in the certificate validation processes. For creating Certification Authorities and certificates look the syntax of `openssl x509` or `XCA`[2]. 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].

To complete this guide you will need to obtain a X509 Public Key Certificate. They are present in the Portuguese Citizen Card.

In alternative you can also use software such as XCA to create a new Certification Authority and certificates, or simply obtain the certificate from any website using the `openssl s_client -connect host:port -showcerts` command.

4.2 Certificate Validity Interval

X.509 public key certificates contain a validity interval, that defines the temporal date interval where the certificate may be used.

Implement a small program that reads a certificate and a function that verifies its vality by analysing the attributes `not_valid_after` and `not_valid_before`.

The certificate loaded should be stored in a dictionary where the key is the certificate subject. This will speed up the implementation of the remaining guide.

HINT: Use the `cryptography.x509` class[4] to load the certificate and then

[1] `https://datatracker.ietf.org/wg/pkix/documents/`
[2] `https://sourceforge.net/projects/xca/`
[3] https://cryptography.io/en/latest/x509/tutorial/
[4] `https://cryptography.io/en/latest/x509/reference/`

check its attributes.

4.3 Trust Anchor certificates

Trust anchor certificates are the user trustable certificates, that typically are root certificates (i.e., self-signed certificates). Trust anchor certificates are important to validate certification paths (certificate chains). Every valid certification path must terminate (have a root) in a trust anchor certificate.

4.3.1 Reading trust anchor certificates

Usually trust anchor certificates are provided by the operating system or the program in use (e.g., Firefox, Chrome, etc.), but may also be provided by the user. The anchor certificates in which we trust must be protected in a `keystore`, or a restricted location (`/etc/ssl/certs`), to prevent new additions our removals, intentionally or not intentionally. Python uses the certificates present in the system, as individual files containing certificates in the PEM format.

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

Do not load certificates with that have expired (use the previous function)

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

4.4 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 till 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 certification path together with his certificate.

For this exercise, take each user certificate (from CC, downloaded, etc...), and create list with the full chain. To create the chain, load the user certificate, a dictionary of user specified intermediate roots, and the roots.

Then build a list starting on the user certificate and adding each issuer certificate.

You should stop when you get a root certificate (self signed).

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

4.5 Validate a certification path

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

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

HINT: A CRL is a file that can be downloaded and loaded using the `x509.load_pen_x509_crl` method. It is comprised by a list of Revoked Certificates, and you can search it for the certificate under validation.

The next step will be to validate the signatures in the chain. Each certificate should have a signature (`x509.signature` field) created with the private key of the issuer. 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 its purpose, common name and validity. Build a function that checks these attributes, and then call the previous functions to further validate the certificate chain.

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

4.6 Challenges

Alter your program to only use the CRL mechanism for revocation check when validating the certification path.

Alter your program to use OCSP first and to use CRL if OCSP mechanism fails.

Add caching mechanisms that store OCSP and CRL responses while they are valid.

4.7 Bibliography

https://cryptography.io/en/latest/x509 https://en.wikipedia.org/wiki/X.509