X.509 certificates and PKI

Laboratory for the class "Cybersecurity" (01UDR)
Politecnico di Torino – AA 2021/22
Prof. Antonio Lioy

prepared by:
Diana Berbecaru (diana.berbecaru@polito.it)
Andrea Atzeni (andrea.atzeni@polito.it)

v. 1.0 (22/10/2021)

Contents

1 Purpose of this laboratory

In this laboratory, you will perform exercises to experiment more in depth with PKIs and X.509 certificates.

The laboratory uses the OpenSSL (http://www.openssl.org/) open-source library and tools, available for various platforms, including Linux and Windows.

Most of the proposed exercises basically use the OpenSSL command line program, which allows the use of several cryptographic functions by means of the OpenSSL shell that can be started with the following command:

```
openssl command [ command_opts ] [ command_args ]
```

openssl x509

To sign and view an X.509 certificate, you can use the OpenSSL command `x509`. Actually, the `x509` command is a multi purpose certificate utility: it can be used to display certificate information, convert certificates to various forms, sign certificate requests (behaving thus as a "mini CA"), or edit certificate trust settings. The simplified syntax of this command for the purpose of exercises proposed is:

```
openssl x509 [-inform DER|PEM] [-outform DER|PEM] [-in file] [-out file]
     [-noout] [-req] [-text]
```

where the main options have the following meaning:

`-inform DER|PEM` specifies the input format; normally the command will expect an X.509 certificate but this can change if other options (such as `-req`) are present. The DER value indicates that the input certificate is encoded with the DER encoding, PEM that the certificate is encoded in PEM, which is the base64 encoding of the DER encoding with header and footer lines added.

`-outform DER|PEM` specifies the output format (same possible values as with the `-inform` option).

`-in` *filename* specifies the input file to read a certificate from (standard input is used otherwise).

`-out` *filename* specifies the output file to write to (standard output is used otherwise).

`-noout` indicates not to produce in output the Base64 representation of the certificate.

`-req` indicates that the file *file* passed in with the option `-in` contains a request and not a certificate.

`-text` prints out the certificate in text form. Full details are shown including the public key, signature algorithms, issuer and subject names, serial number any extensions present and any trust settings.

`-ext` *extensions* prints out the certificate extensions in text form. Extensions are specified with a comma separated string, e.g. "subjectAltName,subjectKeyIdentifier".

`-ocsp_uri` Outputs the OCSP responder address(es) if any

To find out more details about the `x509` command execute:

```
man x509
```

openssl verify

This command allows the verification of a certificate and its certification chain. The verify operation consists of a number of separate steps:

1. First a certificate chain is built up starting from the supplied certificate and ending in the root CA

(that is the first self-signed certificate being found). If the whole chain cannot be built up, an error is signalled. The chain is built up by looking up the issuers certificate of the current certificate. If a certificate is found which is its own issuer it is assumed to be the root CA.

In practice, to find the issuer, all certificates whose subject name matches the issuer name of the current certificate are subject to further tests. The relevant authority key identifier components of the current certificate (if present) must match the subject key identifier (if present) and issuer and serial number of the candidate issuer, in addition the keyUsage extension of the candidate issuer (if present) must permit certificate signing.

The lookup first looks in the list of untrusted certificates and if no match is found the remaining lookups are from the trusted certificates. The root CA is always looked up in the trusted certificate list: if the certificate to verify is a root certificate then an exact match must be found in the list.

2. The second operation is to check every untrusted certificate's extensions for consistency with the supplied purpose. If the -purpose option is not included then no checks are done. The supplied or "leaf" certificate must have extensions compatible with the supplied purpose and all other certificates must also be valid CA certificates. The precise extensions required are described in more detail in the CERTIFICATE EXTENSIONS section of the x509 utility.

3. The third operation is to check the root trust settings, i.e. check that the root CA is trusted.

4. The final operation is to check the validity of the certificate chain. The validity period is checked against the current system time and the *notBefore* and *notAfter* dates in the certificate. The certificate signatures are also checked at this point.

If all operations complete successfully then the certificate is considered valid. If any operation fails then the certificate is invalid. When a verify operation fails, the output messages can be somewhat cryptic. The general form of the error message is:

```
server.pem:  /C=AU/O=CryptSoft Ltd/CN=Test CA (1024 bit)
error 24 at 1 depth lookup:invalid CA certificate
```

The first line contains the name of the certificate being verified, followed by the Subject in the certificate. The second line contains the error number and the depth. The depth is number of the certificate being verified when a problem was detected, starting with zero for the certificate being verified itself, then 1 for the CA that signed this certificate, and so on. Finally a text version of the error number is presented. The most common error codes returned are listed in the manual page of this command (`man verify`). The simplified syntax of this command is:

```
openssl verify [-CAfile file] [-crl_check] [-CRLfile file] [-verbose]
[certificates]
```

where:

`-CAfile` *file* indicates a file (whose name is given in `file`) of trusted certificates. The file should contain multiple certificates in PEM format concatenated together;

`-help` prints out a usage message;

`-verbose` prints extra information about the operations being performed;

`certificates` indicates one or more certificates to verify. If no certificate filenames are included, then an attempt is made to read a certificate from standard input. They should all be in PEM format;

`-crl_check` checks end entity certificate validity by attempting to look up a valid CRL. If a valid CRL cannot be found an error occurs;

`-CRLfile` *file* The file should contain one or more CRLs in PEM format. This option can be specified more than once to include CRLs from multiple files;

For more details on the options of this command, you should run:

```
man verify
```

openssl crl

The crl command processes CRL files in DER or PEM format. Among the other, it allow for CRL validity verification and conversion from DER to PEM format and vice-versa
The simplified syntax of this command is:

```
openssl crl [-inform DER|PEM] [-outform PEM|DER] [-text] [-in filename]
[-out filename] [-noout] [-issuer] [-lastupdate] [-nextupdate] [-CAfile
file]
```

where:

`-inform DER|PEM` this specifies the input format. DER format is DER encoded CRL structure. PEM (the default) is a base64 encoded version of the DER form with header and footer lines;

`-outform DER|PEM` this specifies the output format, the options have the same meaning and default as the -inform option;

`-in` *filename* this specifies the input filename to read from or standard input if this option is not specified;

`-text` print out the CRL in text form;

`-noout` don't output the encoded version of the CRL;

`-issuer` output the issuer name;

`-lastupdate` output the lastUpdate field;

`-nextupdate` output the nextUpdate field;

`-CAfile` *file* verify the signature on a CRL by looking up the issuing certificate in file.

openssl ocsp

The ocsp command performs many common OCSP tasks. It can be used to print out requests and responses, create requests and send queries to an OCSP responder, and behave like a mini OCSP server itself.
The simplified syntax of this command is:

```
openssl ocsp [-out file] [-issuer file] [-cert file] [-resp_text] [-url
URL] [-CAfile file]
```

where:

`-out` *file* specify output filename, default is standard output;

`-issuer` *filename* this specifies the current issuer certificate. This option can be used multiple times. The certificate specified in *filename* must be in PEM format. This option MUST come before any -cert options;

`-cert` *filename* add the certificate filename to the request. The issuer certificate is taken from the

previous issuer option, or an error occurs if no issuer certificate is specified;

`-resp_text` print out the text form of the OCSP response

`-url` *responder_url* specify the responder URL. Both HTTP and HTTPS (SSL/TLS) URLs can be specified;

`-CAfile` *file* file containing trusted CA certificates. These are used to verify the signature on the OCSP response.

2 Analysis and status checking of X.509 certificates

2.1 Getting and analysing an X.509 certificate

The first step consists in downloading the certificate you want to analyse. Open your browser (such as Firefox, Chrome, or Edge) and connect to the URL: https://www.polito.it.

NOTE

You can alternatively use also other web sites, such as https://www.repubblica.it, or https://www.sony.com and try to perform the operations below.

Click on the padlock sign in the address bar, that allows you to check the exchanged certificates (for example, in Firefox select "Connection secure", "More information", "View certificate") and view the details of the certificate and of the certificate chain.

Save the server certificate and the signing CA's certificate in PEM format (look for the "Download" entry in the Miscellanous section), let's suppose to call them `www-polito-it.pem` and `www-polito-it-CA.pem` respectively.

NOTE

Some browser try to save those certificates using misleading names. For example, Firefox try to save both the server certificate AND the CA certificate with the same name `www-polito-it.pem`. Pay attention to this issue and properly rename them

Next, you can analyse their content with OpenSSL commands, as follows.

View the certificate/chain content:

```
openssl x509 -in www-polito-it.pem -text -noout
```

```
openssl x509 -in www-polito-it-CA.pem -text -noout
```

To save the public key out of the certificate:

```
openssl x509 -in www-polito-it-CA.pem -pubkey -noout >terena.pubkey.pem
```

To view the subject you can use the command:

```
openssl x509 -in www-polito-it.pem -subject -noout
```

To view the `subjectAltName` and the `subjectKeyIdentifier` extensions you can use the command:

```
openssl x509 -in www-polito-it.pem -ext "subjectAltName,subjectKeyIdentifier"
-noout
```

What is the purpose of the `subjectAltName` extension ?

```
→
```

To view the dates you can use the commands:

```
openssl x509 -in www-polito-it.pem -dates -noout
```

To display the CRL pointer in the `crlDistributionPoint` extension, you can use the command:

```
openssl x509 -in www-polito-it.pem -ext crlDistributionPoints -noout
```

To display the URL of the OCSP responder in the Authority Information Access (AIA) extension, you can use the command:

```
openssl x509 -in www-polito-it.pem -ocsp_uri -noout
```

To display the whole certificate content with OpenSSL asn1parse:

```
openssl asn1parse -in www-polito-it.pem
```

To convert from PEM format to DER format:

```
openssl asn1parse -in www-polito-it.pem -out www-polito-it.der
```

2.2 Certificate status checking

2.2.1 CRL verification

Verifying certificate status with a CRL consists basically of the following steps:

- obtain the certificate you wish to check for revocation
- obtain the issuer certificate (CA)
- download and verify the CRL (for authentication, integrity, and trust)
- check the certificate status with the CRL.

Now proceed as follows. Try to download the CRL (for the Polito web server certificate) through the browser. Connect to `www.polito.it`, click on the padlock sign in the address bar, and view the details of the certificate. To find out the URL of the CRL, check once again the `crlDistributionPoint` extension in the server certificate. Insert the CRL's URL is the browser address bar and save it locally. You should have a *filename.crl* (e.g. `GEANT OV RSACA4.crl`). By double-clicking on the *filename.crl*, in Windows you should be able to import and see the content of the CRL imported in the dedicated trust store of Windows. In Fig. 1 we show an example of a CRL in Windows:

At this point, you can inspect the CRL with the dedicated OpenSSL command:

```
openssl crl -inform DER -in filename.crl -text
```

Look at the first revoked certificate in this list: it indicates "Revocation Date: Mar 24 18:11:09 2020 GMT". If we receive on Nov 6 20:40 2021 a document signed with that certificate, should we accept that document as valid?

```
→
```

Look at the certificate in the list that indicates "Revocation Date: Oct 21 11:53:12 2021 GMT"

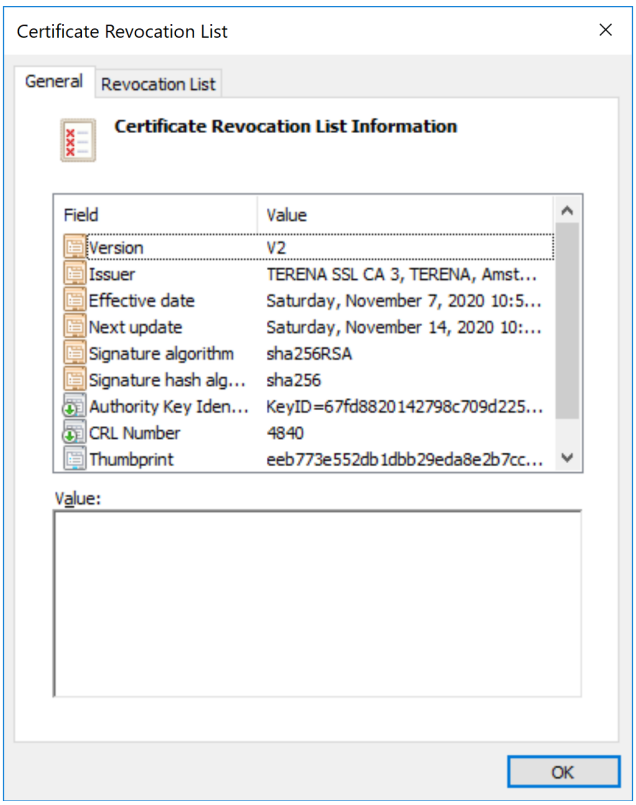

Figure 1: Viewing the CRL content in Windows.

Let's assume that we are on Oct 21 12:05:12 2021 and we have just received a document signed with the above certificate . Should we accept the document as valid?

→

Convert the CRL in PEM format:

```
openssl crl -inform DER -in filename.crl >filename.crl.pem
```

Verify that the CRL is valid (i.e., signed by the issuer certificate):

```
openssl crl -in filename.crl -inform DER -CAfile www-polito-it-CA.pem -noout
```

You should see a "verify OK" message.

Verify the validity of the certificate (no revocation check):

```
openssl verify -CAfile www-polito-it-CA.pem www-polito-it.pem
```

Verify the server certificate, including the checking of certificate revocation status with the CRL:

```
openssl verify -CAfile www-polito-it-CA.pem -crl_check -CRLfile filename.crl.pem
www-polito-it.pem
```

You should see a "www-polito-it.pem: OK" message indicating that the verification completed successfully.

2.2.2 OCSP verification

To check the status of the server certificate with OCSP you can use the command:

```
openssl ocsp -issuer www-polito-it-CA.pem -cert www-polito-it.pem -url <ocsp_uri>
-resp_text
```

where the *ocsp_uri* can be obtained querying the server certificate with the command

```
openssl x509 -in www-polito-it.pem -ocsp_uri -noout
```

And you should see a "Response verify OK" message.

Now, to experiment with a revoked certificate, connect to https://revoked.badssl.com/ and redo the same CRL and OCSP verification operations above, for the revoked certificate.

2.2.3 OCSP Stapling

OCSP stapling is an optional feature that allows a server certificate to be accompanied by an OCSP response that proves its validity. Because the OCSP response is delivered over an already existing connection, the client does not have to fetch it separately.

OCSP stapling is used only if requested by a client, which submits the `status_request` extension in the TLS handshake request. A server that supports OCSP stapling will respond by including an OCSP response as part of the TLS handshake.

You can use OpenSSL s_client tool to check if a server supports OCSP stapling. For example, the following server supports it:

```
$echo | openssl s_client -connect www.sony.com:443 -status
```

You can also try out:

```
openssl s_client -connect ritter.vg:443 -status
```

The OCSP-related information will be displayed at the very beginning of the connection output. In case the server supports stapling, you will see the entire OCSP response in the output. For example, with a server that does not support stapling, you will see an "OCSP response: no response sent".

Check out with the above command some famous website (e.g. www.google.com) to verify whether they support or not OCSP stapling.

2.3 Extended Validation (EV), Organization Validation (OV), Domain Validated (DV)

Some CAs issue TLS server certificates with a particular extension named Extended Validation. During administrative verification of an EV certificate request, the owner of the website passes a thorough and globally standardized identity verification process (a set of vetting principles and policies ratified by the CA/Browser forum) to prove exclusive rights to use a domain, confirm its legal, operational, and physical existence, and prove the entity has authorized the issuance of the certificate. This verified identity information is included within the certificate.

The verification process for OV and DV certificates is not as comprehensive as for EV. DV certificates only require a proof that the website owner has administrative control over the domain. OV certificates include some identity information about the site operator, but it is not as extensive as for EV.

Some example of TLS server certificates with EV, OV, and DV extensions may be found below. For each of the examples, connect to the URL, view the server certificate and analyse the Certificate Policies extensions.

A server certificate with DV extension: https://www.nist.org/

A server certificate with OV extension: https://www.sony.com/

A server certificate with EV extension: https://www.globalsign.com/

3 Certificate Transparency

3.1 Analysing SCT extensions in an X.509 certificate

In the browser, open the website (e.g. https://www.polito.it), click on the padlock icon next to the URL bar and view the certificate details. On the `Certificate Details` page click on `Details` and expand the SCT List.

There you should see the date and time when the signed certificate stamp (SCT) was added to the public CT Log Servers.

Then, run the command:

```
openssl x509 -in www-polito-it.pem -text
```

Analyse the output, in particular the "CT Precertificate SCTs" part. You should be able to note a finer time precision in the "Timestamp" field.

Respond to the questions: Why the SCTs have been proposed, and in particular the Certificate Transparency occurred? What does it mean "Precertificate"?

→

3.2 Checking certificate presence in CT logs

You can check whether a certain domain's certificate is present in the active public certificate transparency log.

With your browser, go to https://transparencyreport.google.com/https/certificates. You should see a page as illustrated in Fig. 2.

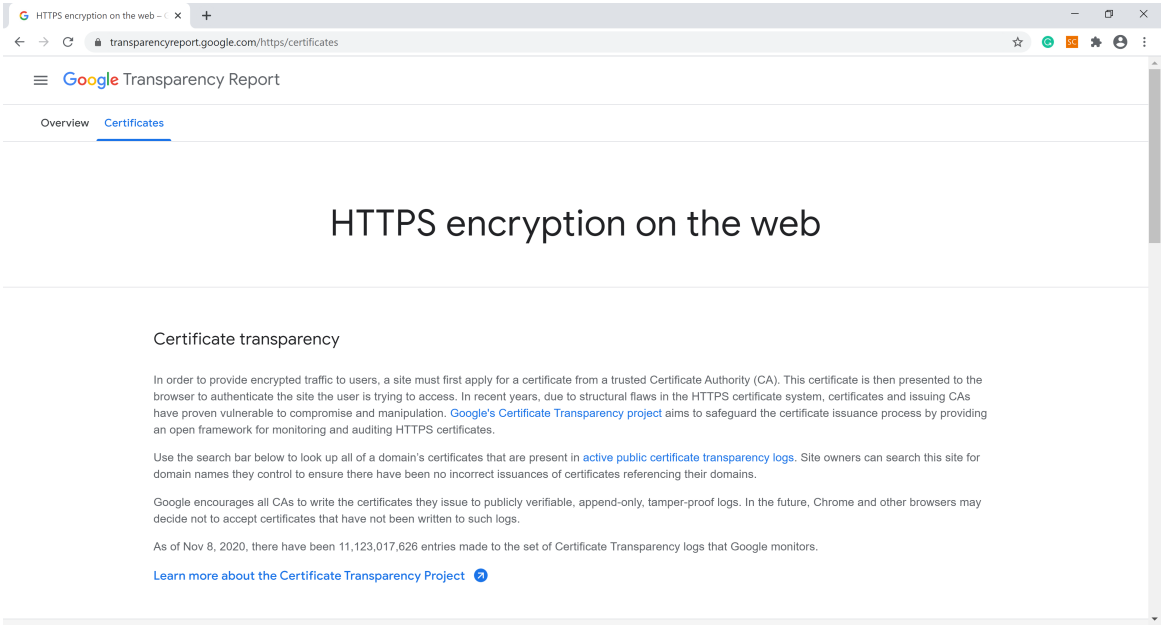

Figure 2: Google Transparency Report.

In the section "Search certificates by hostname", insert the domain www.polito.it You should be able to see the current status of the certificates issued for this domain, but also the past ones, as illustrated in Fig. 3.

Now, look for yourself in the Google Transparency Report for the certificate in the `www-globalsign-com.pem` that you can download by connectecting to https://www.globalsign.com.

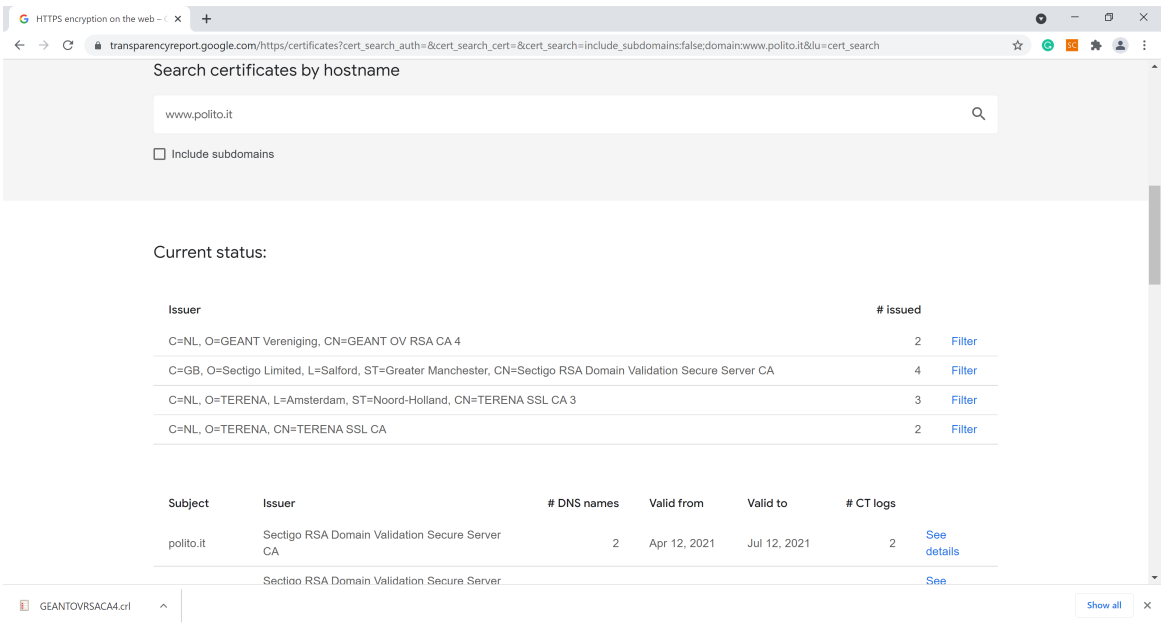

Figure 3: Google Transparency Report for www.polito.it.

3.3 Known CT Logs

At the link `http://www.certificate-transparency.org/known-logs` you find information about the CT Logs that are currently compliant with Chrome's CT policy: `https://www.gstatic.com/ct/log_list/v2/log_list.json`

But you can check also a list of all known and announced CT Logs: `https://www.gstatic.com/ct/log_list/v2/all_logs_list.json`

4 Certificate chains and PKI models

4.1 Viewing and verification of simple certificate chains

In this exercise we will explain first how you can view the certificate chain. Open Google Chrome browser, and navigate for example to the URL `https://globalsign.com`. Click on the padlock, then on the Certificate (to view its details). Finally select the tab "Certification Path". You should be able to see the entire certificate chain, from the server certificate up to the Root CA, as illustrated in Fig. 4.

Which fields in the certificate have been exploited to construct the certificate chain you see?

→

Why the verification of this chain (as you see it in the browser) is successful? Hint: Check out the List of trusted Root CAs in your browser.

→

Now, take the files provided as support material for this laboratory (`CYB_lab02_support.zip`):

- `www-globalsign-com.pem`,

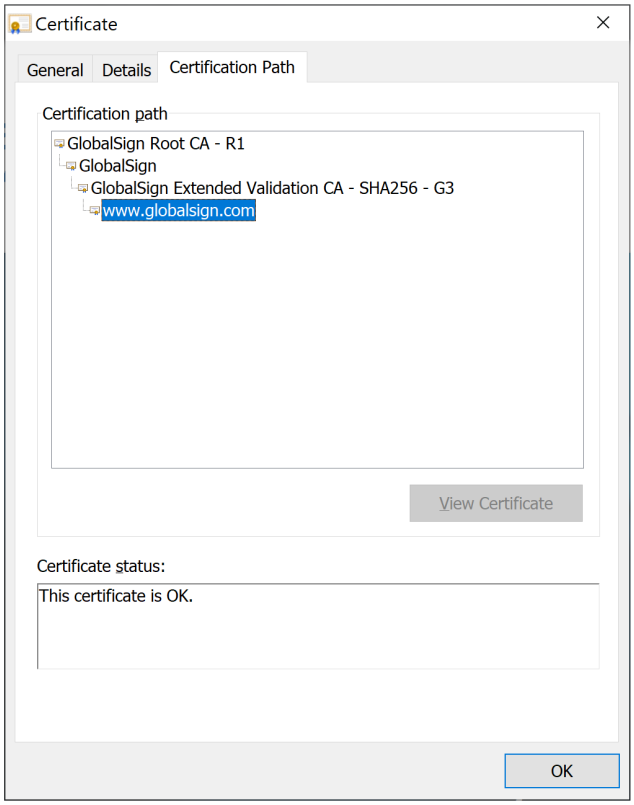

Figure 4: Example of certificate chain viewed in Google Chrome.

- `GlobalSign_Extended_Validation_CA.pem`,
- `GlobalSign_CA.pem`, and
- `GlobalSign_Root_CA.pem`.

Then, run the following command:

```
openssl verify -CAfile GlobalSign_Root_CA.pem www-globalsign-com.pem
```

Did you get any error? If yes, can you explain its cause? Note that we did specify a Root CA in the command, so the trust anchor is present.

→

Now try out the following command:

```
openssl verify -verbose -CAfile <(cat GlobalSign_Extended_Validation_CA.pem
GlobalSign_CA.pem GlobalSign_Root_CA.pem) www-globalsign-com.pem
```

Do you still get any error ?

→

4.2 Viewing of a real Federal PKI

To have an idea about how complex the Federal PKIs can become, we suggest you to visit: https://playbooks.idmanagement.gov/fpki/tools/fpkigraph/. At this site, which is an official site of United States Government, you can see the (complex) connections of various CAs across US Government.

It's interesting to note for example a Hierarchical PKI: the "US Treasury Root CA " (which is rooted in the "Federal Common Policy CA"), issued certificates to the "Social Security Administration Certification Authority" and "US Treasury Fiscal Service". This allows Relying Parties running fiscal applications to validate certificates (e.g. signed documents exploiting such certificates) originating from the Social Security Administration area.

Look for yourself in the graph, try to find the above mentioned connection between these PKIs.

It's also interesting to note for example that the "Federal Bridge CA G4" issued a certificate to "Dod Interoperability Root CA2", which in turn issued a certificate to the "Dod Root CA3", which issued certificates to several other DoD SW CAs (DoD stands for Department fo Defense). On the other hand the "Federal Bridge CA G4" issued a certificate to "Symantec Class SSP Intermediate CA GA3", which issued a certificate to "Eid Passport LRA 2 CA". Thus, applications in the Dod SW area can validate (through the Bridge CA) certificate issued by the Eid Passport LRA CA". Look for yourself in the graph, and find out the other interesting connections. For example, " Boeing PCA G3" is connected through 2 Bridge CAs (CertiPath Bridge CA - G3 and Federal Bridge CA G4) and 2 Root CAs (Dod Interoperability Root CA2 and Dod Root CA3) to the DoD CAs, e.g. DoD EMAIL CA-41.

Now let's assume a user exploits an application (e.g. email) that uses a certificate issued by DoD EMAIL CA-41, for example a signed e-mail. Then, he send the signed e-mail to another user, which configured as trusted the "Boeing PCA G3".

Is the validation of the certificate (attached to the signed e-mail) successful? Which is the certificate path?

→