

Premises

Lab Activity - Liviu-Ioan ZECHERU

All assignments will be structured as snapshot of the assignment then snapshot of my result.

Certificates

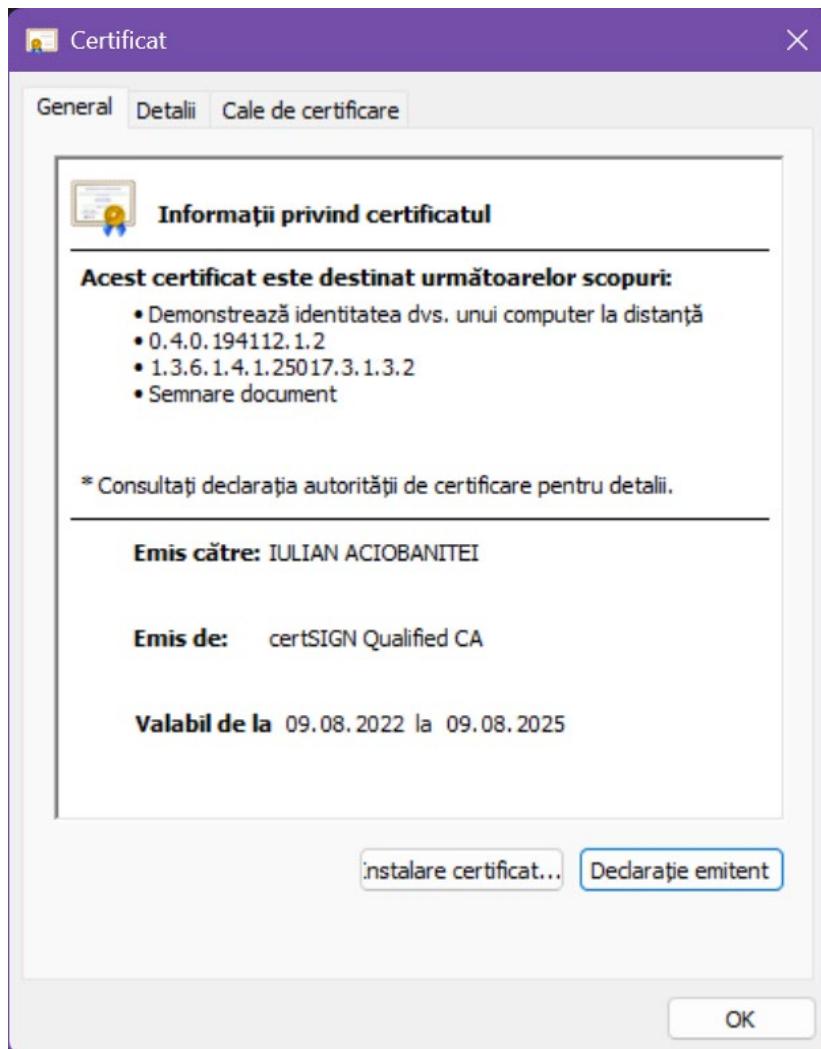
1

Download [this certificate](#)

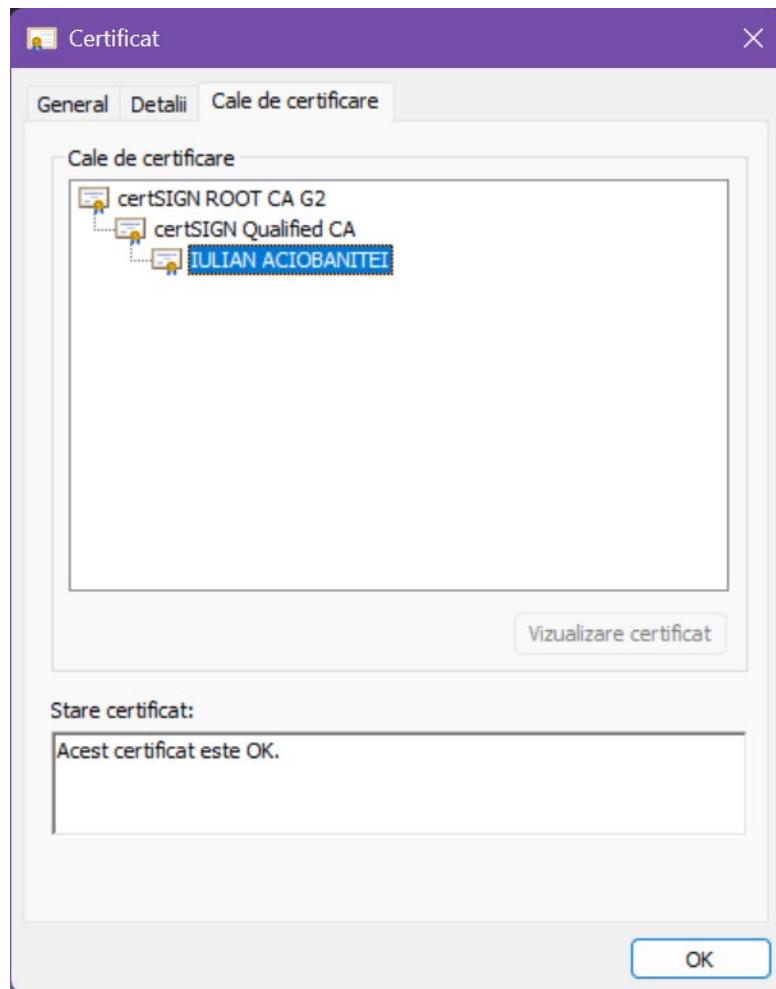
(link

behind

https://drive.google.com/file/d/16KXExKkzw4b350edxrJ7hAA6YTLbQk72/view?usp=share_link)



② Download all the certificates in the certification path



The certificates in the certification path are: certSIGN Qualified CA and certSIGN ROOT CA G2. Here they are:

The image contains two side-by-side screenshots of the 'Certificat' dialog box, both showing the 'Detalii' tab.

Left Screenshot (certSIGN Qualified CA):

- Informații privind certificatul:**
 - Acest certificat este destinat următoarelor scopuri:**
 - Demonstrează identitatea dvs. unui computer la distanță
 - Protejează mesajele de e-mail
 - Asigură identitatea unui computer la distanță
 - Toate politice de emisie
 - Semnare document
 - * Consultați declarația autorității de certificare pentru detalii.

Emit către: certSIGN Qualified CA
Emit de: certSIGN ROOT CA G2
Valabil de la: 06.02.2017 la 06.02.2027

Instalare certificat... **Declarație emitent**

OK

Right Screenshot (certSIGN ROOT CA G2):

 - Informații privind certificatul:**
 - Acest certificat este destinat următoarelor scopuri:**
 - Demonstrează identitatea dvs. unui computer la distanță
 - Protejează mesajele de e-mail
 - Asigură identitatea unui computer la distanță
 - Toate politice de emisie
 - Semnare document

Emit către: certSIGN ROOT CA G2
Emit de: certSIGN ROOT CA G2
Valabil de la: 06.02.2017 la 06.02.2042

Instalare certificat... **Declarație emitent**

OK

3 Download the CRL manually

According to the details of the certificate, the CRL is located here: <http://crl.certsign.ro/certsign-qualifiedca.crl>. Accessing the link will give us this file:

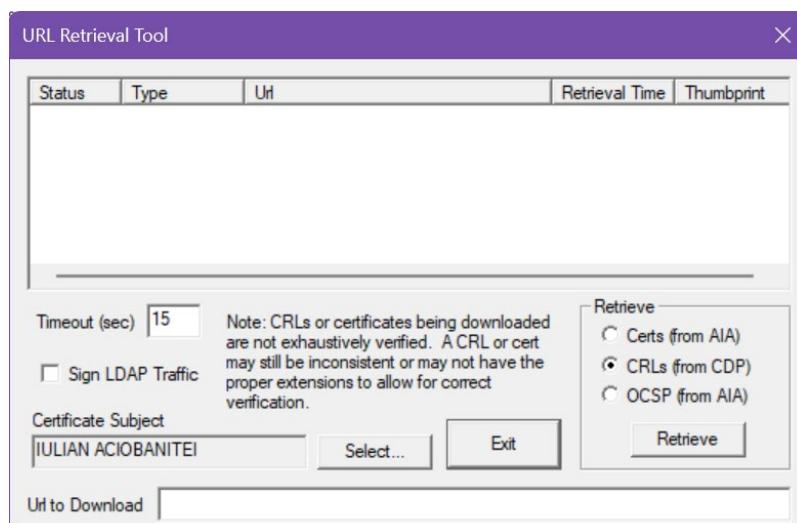
The left window displays information about the revocation list, including the issuer (VATRO-18288250, certSIGN Qualified CA) and the date it became valid (Wednesday, November 26, 2024 18:11:03).

The right window lists the revoked certificates:

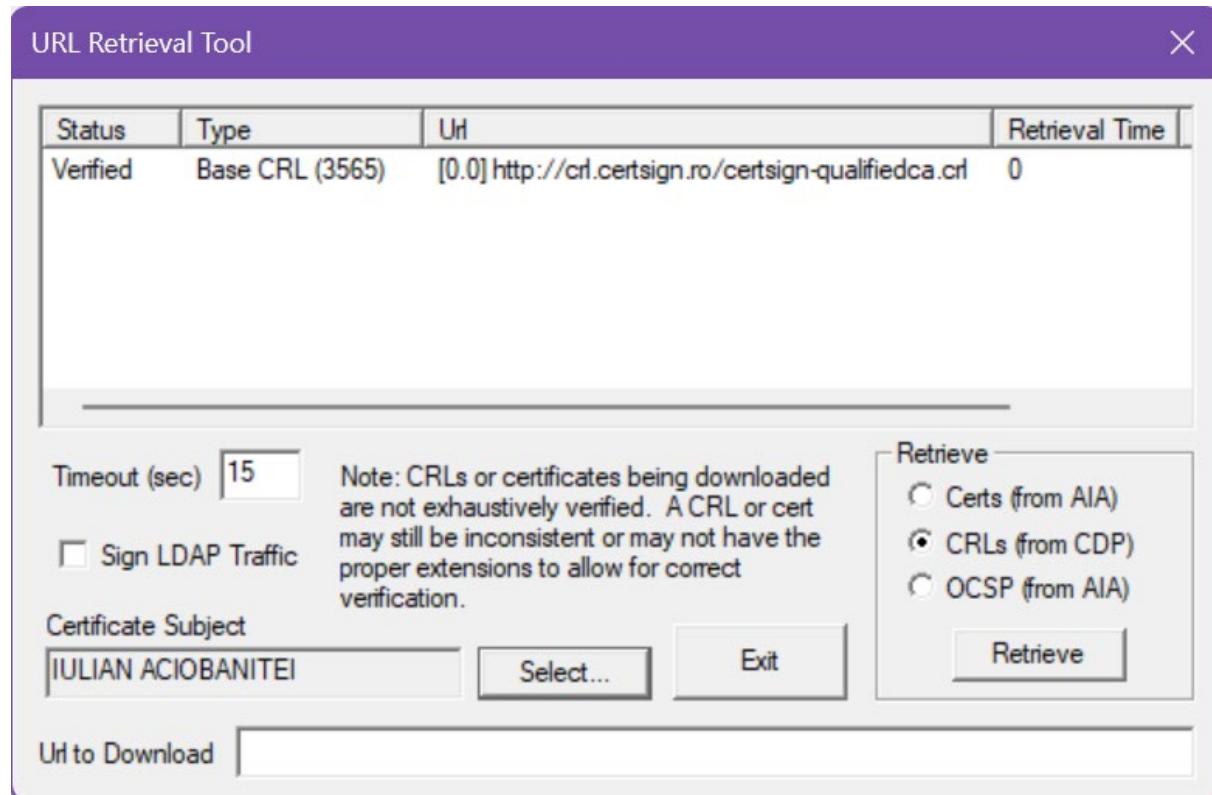
Număr de serie	Data revocării
22038c53229887288ce3	miercuri, 5 aprilie 2017 ...
220108e45ffa4dc6c8d3	vineri, 7 aprilie 2017 16:...
22092896a1defd558aaa	marți, 11 aprilie 2017 13...
220a349c857c447f9f59	marți, 11 aprilie 2017 13...

4 Verify the certificate using certutil and CRL

We open a terminal in the path of the certificate and use the command **certutil -url cert.cer** (because that's the name of the certificate). A window like this will open:

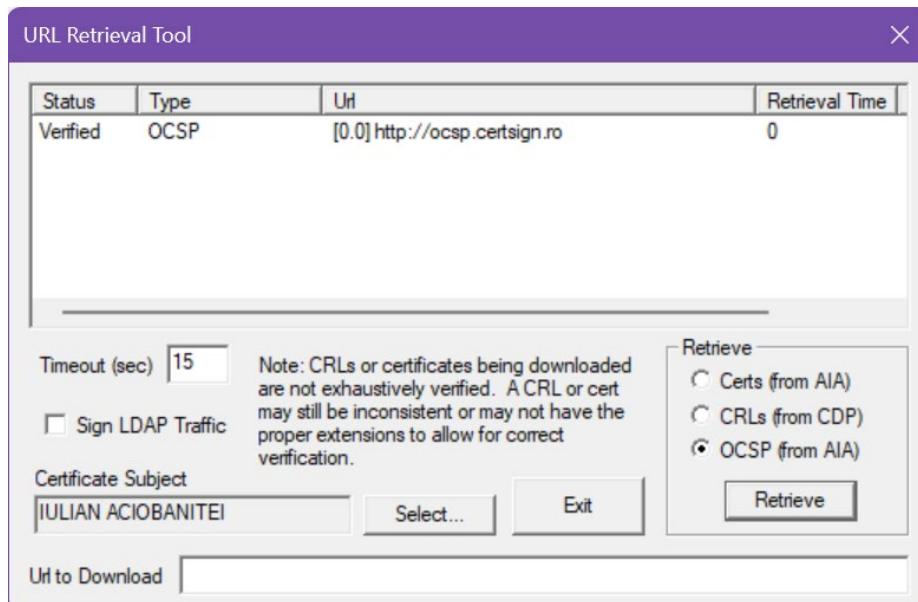


To verify the certificate using the CRL we select the **CRLs (from CDP)** as retrieval method then click on **Retrieve** button. Then we can see the status in the view area.



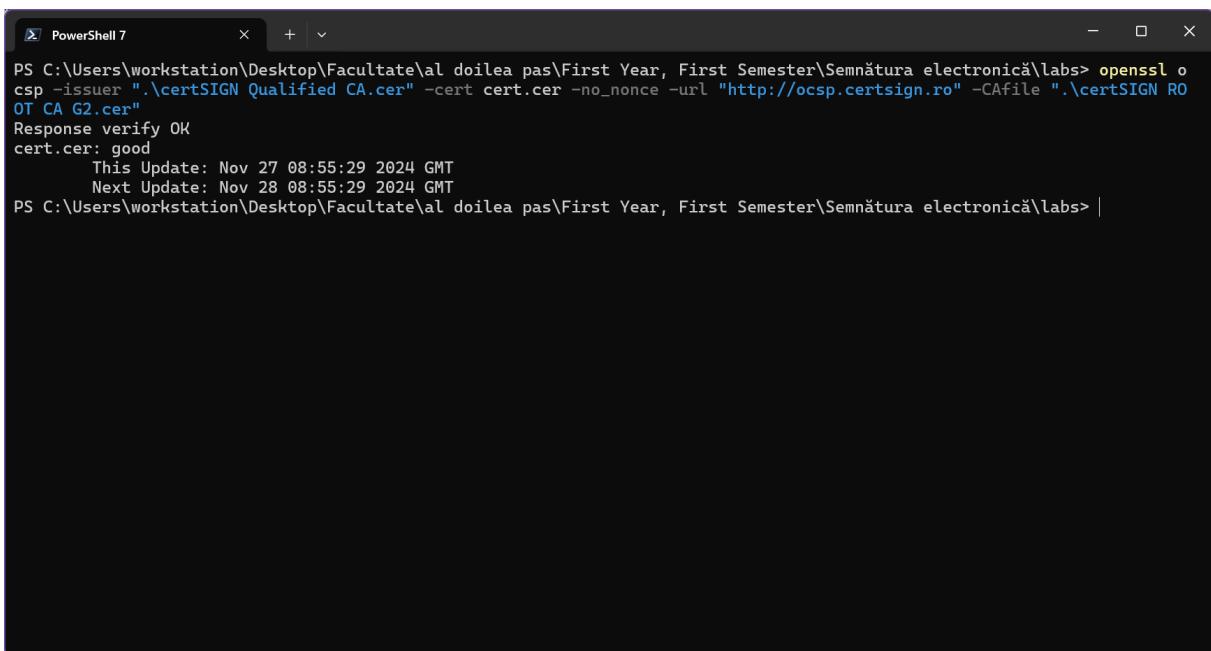
⑤ Verify the certificate using certutil and OCSP

Similarly, we select the **OCSP (from AIA)** as retrieval method then click on **Retrieve** button. Then we can see the status in the view area.



6 Verify the certificate using openssl*

After we install the **OpenSSL** package, we open up a terminal in the path of the certificate and use the given command: `openssl ocsp -issuer ca_cert.cer -cert cert.cer -no_nonce -url "ocsp url" -CAfile root.cer` adapted to our naming. In my case it will be `openssl ocsp -issuer ".\certSIGN Qualified CA.cer" -cert cert.cer -no_nonce -url "http://ocsp.certsign.ro" -CAfile ".\certSIGN ROOT CA G2.cer"`. Here is the result:



```
PowerShell 7
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> openssl ocsp -issuer ".\certSIGN Qualified CA.cer" -cert cert.cer -no_nonce -url "http://ocsp.certsign.ro" -CAfile ".\certSIGN ROOT CA G2.cer"
Response verify OK
cert.cer: good
    This Update: Nov 27 08:55:29 2024 GMT
    Next Update: Nov 28 08:55:29 2024 GMT
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> |
```

7 View certificate using an ASN.1 viewer (openssl cmd or [lapo.it](#))

I choose to view it using the **OpenSSL** package directly in the command prompt. The command I use is (from the location of the certificate): `openssl asn1parse -inform DER -in .\cert.cer`. The resulted output is:

```

PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> openssl asn1parse -inform DER -in .\cert.cer
0:d=0 hl=4 l=1781 cons: SEQUENCE
4:d=1 hl=4 l=1245 cons: SEQUENCE
8:d=2 hl=2 l= 3 cons: cont [ 0 ]
10:d=3 hl=2 l= 1 prim: INTEGER :02
13:d=2 hl=2 l= 12 prim: INTEGER :220936B818E89FF45D5C2664
27:d=2 hl=2 l= 13 cons: SEQUENCE
29:d=3 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
40:d=3 hl=2 l= 0 prim: NULL
42:d=2 hl=2 l= 92 cons: SEQUENCE
44:d=3 hl=2 l= 11 cons: SET
45:d=4 hl=2 l= 10 cons: SEQUENCE
48:d=5 hl=2 l= 3 prim: OBJECT :countryName
53:d=5 hl=2 l= 2 prim: PRINTABLESTRING :RO
57:d=3 hl=2 l= 30 cons: SET
59:d=4 hl=2 l= 18 cons: SEQUENCE
61:d=5 hl=2 l= 3 prim: OBJECT :organizationName
66:d=5 hl=2 l= 11 prim: PRINTABLESTRING :CERTSIGN SA
79:d=3 hl=2 l= 30 cons: SET
81:d=4 hl=2 l= 28 cons: SEQUENCE
83:d=5 hl=2 l= 3 prim: OBJECT :commonName
88:d=5 hl=2 l= 21 prim: PRINTABLESTRING :certSIGN Qualified CA
111:d=3 hl=2 l= 23 cons: SET
113:d=3 hl=2 l= 21 cons: SEQUENCE
115:d=5 hl=2 l= 3 prim: OBJECT :organizationIdentifier
120:d=5 hl=2 l= 14 prim: PRINTABLESTRING :VATRO-18288250
136:d=3 hl=2 l= 30 cons: SEQUENCE
137:d=3 hl=2 l= 13 prim: UTCTIME :220809093106Z
153:d=3 hl=2 l= 13 prim: UTCTIME :250809093106Z
158:d=2 hl=3 l= 140 cons: SEQUENCE
171:d=3 hl=2 l= 11 cons: SET
173:d=4 hl=2 l= 9 cons: SEQUENCE
175:d=5 hl=2 l= 3 prim: OBJECT :countryName
180:d=5 hl=2 l= 20 cons: SET
184:d=3 hl=2 l= 20 cons: SET
186:d=4 hl=2 l= 18 cons: SEQUENCE
188:d=5 hl=2 l= 3 prim: OBJECT :organizationName
193:d=5 hl=2 l= 11 prim: UTF8STRING :CERTSIGN SA
206:d=3 hl=2 l= 27 cons: SET
208:d=4 hl=2 l= 25 cons: SEQUENCE
216:d=5 hl=2 l= 3 prim: OBJECT :commonName
215:d=5 hl=2 l= 18 prim: UTF8STRING :IULIAN ACIOBANITEI
237:d=3 hl=2 l= 14 cons: SET
239:d=4 hl=2 l= 5 cons: SEQUENCE
240:d=5 hl=2 l= 5 prim: PRINTABLESTRING :AI916
251:d=3 hl=2 l= 15 cons: SET
253:d=4 hl=2 l= 13 cons: SEQUENCE
255:d=5 hl=2 l= 3 prim: OBJECT :givenName
260:d=5 hl=2 l= 6 prim: UTF8STRING :IULIAN
268:d=3 hl=2 l= 20 cons: SET
270:d=4 hl=2 l= 18 cons: SEQUENCE
272:d=5 hl=2 l= 3 prim: OBJECT :surname
277:d=5 hl=2 l= 11 prim: UTF8STRING :ACIOBANITEI
290:d=3 hl=2 l= 19 cons: SET
292:d=4 hl=2 l= 17 cons: SEQUENCE
294:d=5 hl=2 l= 3 prim: OBJECT :organizationIdentifier
298:d=5 hl=2 l= 10 prim: UTF8STRING :RO18288250
311:d=2 hl=4 l= 299 cons: SEQUENCE
315:d=3 hl=2 l= 3 cons: SEQUENCE
317:d=4 hl=2 l= 9 prim: OBJECT :rsaEncryption
328:d=4 hl=2 l= 0 prim: NULL
330:d=3 hl=4 l= 271 prim: BIT STRING
605:d=2 hl=4 l= 644 cons: cont [ 3 ]
609:d=3 hl=4 l= 640 cons: SEQUENCE
613:d=4 hl=2 l= 120 cons: SEQUENCE
615:d=5 hl=2 l= 8 prim: OBJECT :Authority Information Access
625:d=5 hl=2 l= 108 prim: OCTET STRING [HEX DUMP]:306A302306082B060105050730018617687474703A2F2F6373702E636572747369676E2E726F304306082B060105050730028637687474703A2F
2E777772E636572747369676E2E726F263657274636572747369676E2D7175616C696669656463612E637274
735:d=4 hl=2 l= 14 cons: SEQUENCE
737:d=5 hl=2 l= 3 prim: OBJECT :X509v3 Key Usage
742:d=5 hl=2 l= 1 prim: BOOLEAN :255
745:d=5 hl=2 l= 4 prim: OCTET STRING [HEX DUMP]:030206C0
751:d=4 hl=2 l= 31 cons: SEQUENCE
752:d=5 hl=2 l= 2 prim: OBJECT :X509v3 Authority Key Identifier
753:d=5 hl=2 l= 24 prim: OCTET STRING [HEX DUMP]:301680148F4087515E117FE199C391F1684C3FAC5904B188
784:d=4 hl=2 l= 29 cons: SEQUENCE
786:d=5 hl=2 l= 3 prim: OBJECT :X509v3 Subject Key Identifier
791:d=5 hl=2 l= 22 prim: OCTET STRING [HEX DUMP]:04149A1FDF16A093A5B3A95D7A3525C2DE6385C77638
815:d=4 hl=3 l= 134 cons: SEQUENCE
818:d=5 hl=2 l= 3 prim: OBJECT :X509v3 Certificate Policies
823:d=5 hl=2 l= 27 prim: OCTET STRING [HEX DUMP]:307D393A0607040888EC400102302F302D06082B060105050702011621687474703A2F2F777772E636572747369676E2E726F2F7265706F736974
6F7279303F060C2B06010940181C33903010302302F302D06082B060105050702011621687474703A2F2F777772E636572747369676E2D7175616C696669656463612E63726C
952:d=4 hl=2 l= 64 cons: SEQUENCE
954:d=5 hl=2 l= 3 prim: OBJECT :X509v3 CRL Distribution Points
959:d=5 hl=2 l= 57 prim: OCTET STRING [HEX DUMP]:3037303A03A031862F687474703A2F2F63726C2E636572747369676E2E726F2F636572747369676E2D7175616C696669656463612E63726C
1018:d=4 hl=2 l= 89 cons: SEQUENCE
1020:d=5 hl=2 l= 3 prim: OBJECT :X509v3 Subject Alternative Name
1025:d=5 hl=2 l= 82 prim: OCTET STRING [HEX DUMP]:3050A02E060A2B060104018237140203A0200C1E69756C69616E2E613696F62616E6974656940636572747369676E2E726F811E69756C69616E2E
6163696F62616E6974656940636572747369676E2E726F
1189:d=4 hl=2 l= 31 cons: SEQUENCE
1191:d=5 hl=2 l= 24 prim: OBJECT :X509v3 Extended Key Usage
1195:d=5 hl=2 l= 24 prim: OCTET STRING [HEX DUMP]:3016806082B060105050702060A2B0601040182370A030C
1192:d=4 hl=2 l= 109 cons: SEQUENCE
1194:d=5 hl=2 l= 8 prim: OBJECT :qcStatements
1194:d=5 hl=2 l= 97 prim: OCTET STRING [HEX DUMP]:305F3008060604008E4601013008060604008E4601063009960704008E460106013034060604008E460105302A30281622
68747470733A2F2F777772E636572747369676E2E726F2F7265706F7369746F72791302656E
1253:d=5 hl=2 l= 13 cons: SEQUENCE
1255:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
1266:d=2 hl=2 l= 0 prim: NULL
1268:d=1 hl=4 l= 513 prim: BIT STRING
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs>

```

⑧ Extract the OID of an attribute from Subject extension, at will.

The Subject field in an X.509 certificate contains a series of attributes such as:

Common Name (CN)

Organization (O)

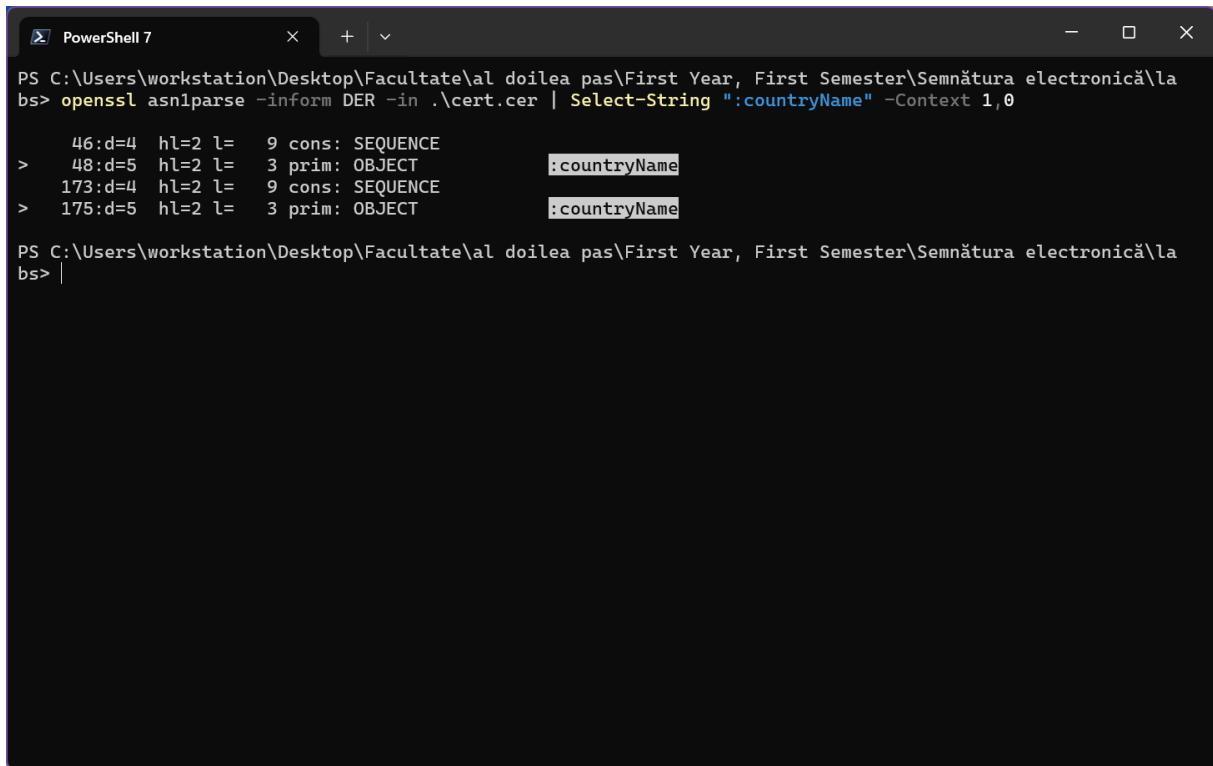
Organizational Unit (OU)

Country (C)

State or Province Name (ST)

Locality Name (L)

Each attribute is identified by an OID (e.g., CN = 2.5.4.3). Let's try to extract the OID of the Country. We first need to get the offset of the zone where OpenSSL interprets the raw binary data as `countryName` by its internal mapping. We use the command `openssl asn1parse -inform DER -in .\cert.cer | Select-String ":countryName" -Context 1,0`. The first part displays the previous output and the command after the pipe searches for the attribute we need using the PowerShell cmdlet `Select-String`. The result is the following:

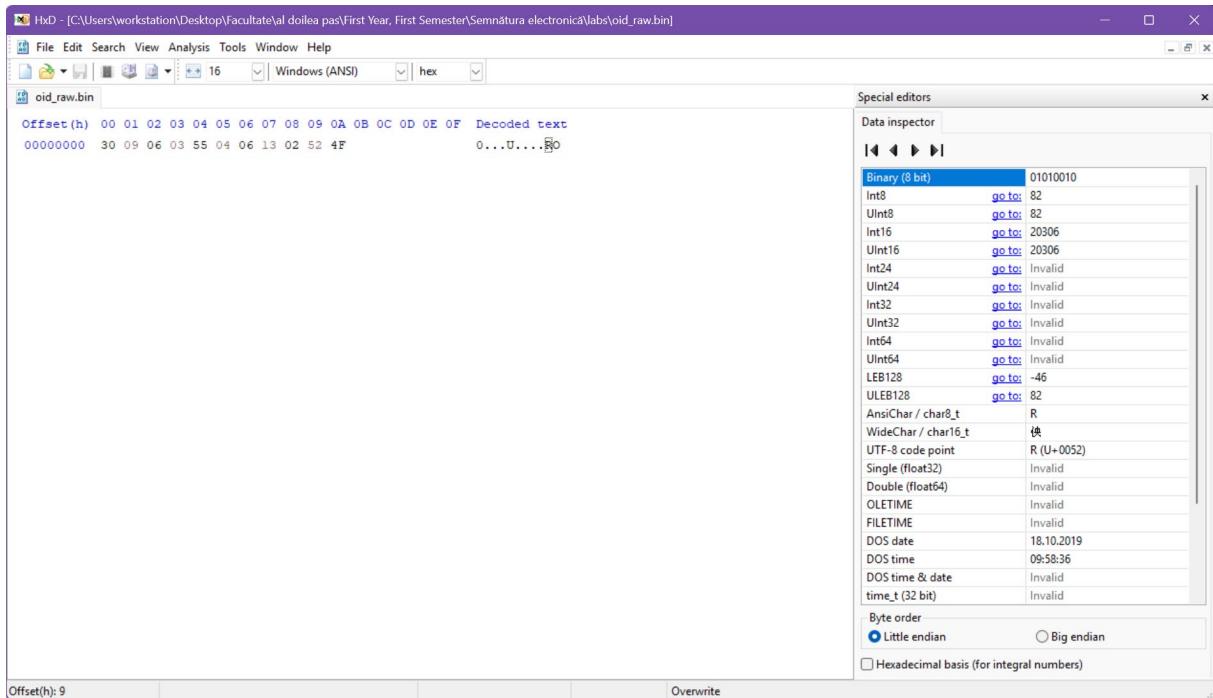


```
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\la
bs> openssl asn1parse -inform DER -in .\cert.cer | Select-String ":countryName" -Context 1,0

 46:d=4 hl=2 l=   9 cons: SEQUENCE
>  48:d=5 hl=2 l=   3 prim: OBJECT            :countryName
 173:d=4 hl=2 l=   9 cons: SEQUENCE
>  175:d=5 hl=2 l=   3 prim: OBJECT            :countryName

PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\la
bs> |
```

Now we see that it appears in two places. We can analyze any of those two. Let's take, for instance, the second one. We will dump the binary data from the offset 173, as that is the starting point, using the command `openssl asn1parse -inform DER -in .\cert.cer -strparse 173 -out oid_raw.bin`. Now we analyze the bytes in a hex viewer like HxD:



We know that OID for the countryName is **2.5.4.6**. Considering the ASN1 encoding, the first byte encodes the first two components by the formula $40 * X + Y$. So, in our case it will be $40 * 2 + 5 = 85$ which is **55** in hexadecimal format. The subsequent components are encoded as a series of variable-length integers. Each integer is encoded using a base-128 system, where the high bit indicates whether the encoding continues to the next byte. We see that we have the sequence **55 04 06**, so we found our OID for the country name attribute!

- ① From EJBCA's admin interface, create a new end entity of type User_Sign or TLS_Client
- ② From EJBCA's public interface, actually generate the certificate and obtain the .p12 file.
- ③ Import the generated certificate into the Windows Store.

I did this part in the live lab at our first meeting. I will just provide the proof of the pages I had to go through and the imported certificate into the Windows Store.

Here is the end entity creation page where, me, as CA, had to issue a new certificate for an end user:

Add End Entity

End Entity Profile	User	Required
Username	<input type="text"/>	<input checked="" type="checkbox"/>
Password (or Enrollment Code)	<input type="password"/>	<input checked="" type="checkbox"/>
Confirm Password	<input type="password"/>	<input checked="" type="checkbox"/>
E-mail address	<input type="text"/> @ <input type="text"/>	<input type="checkbox"/>
Subject DN Attributes		
CN, Common name	<input type="text"/>	<input checked="" type="checkbox"/>
givenName, Given name (first name)	<input type="text"/>	<input type="checkbox"/>
surname, Surname (last name)	<input type="text"/>	<input type="checkbox"/>
C, Country (ISO 3166)	<input type="text"/>	<input checked="" type="checkbox"/>
O, Organization	<input type="text"/>	<input type="checkbox"/>
Main Certificate Data		
Certificate Profile	TLS_Client	<input checked="" type="checkbox"/>
CA	IssuingCA	<input checked="" type="checkbox"/>
Token	P12 file	<input checked="" type="checkbox"/>
		<input type="button" value="Add"/> <input type="button" value="Reset"/>

© 2002-2022 PrimeKey Solutions AB. EJBCA® is a registered trademark of PrimeKey Solutions AB.

Here, as the end user, on the public page, I logged on with the created credentials and generated my PKCS#12 file.

The EJBCA Public Web has been deprecated and will be removed in an upcoming version of EJBCA. Please move your workflows to the EJBCA RA UI

EJBCA Token Certificate Enrollment

Welcome to keystore enrollment.

If you want to, you can manually install the CA certificate(s) in your browser, otherwise this will be done automatically when your certificate is retrieved.

Install CA certificates:

Certificate chain

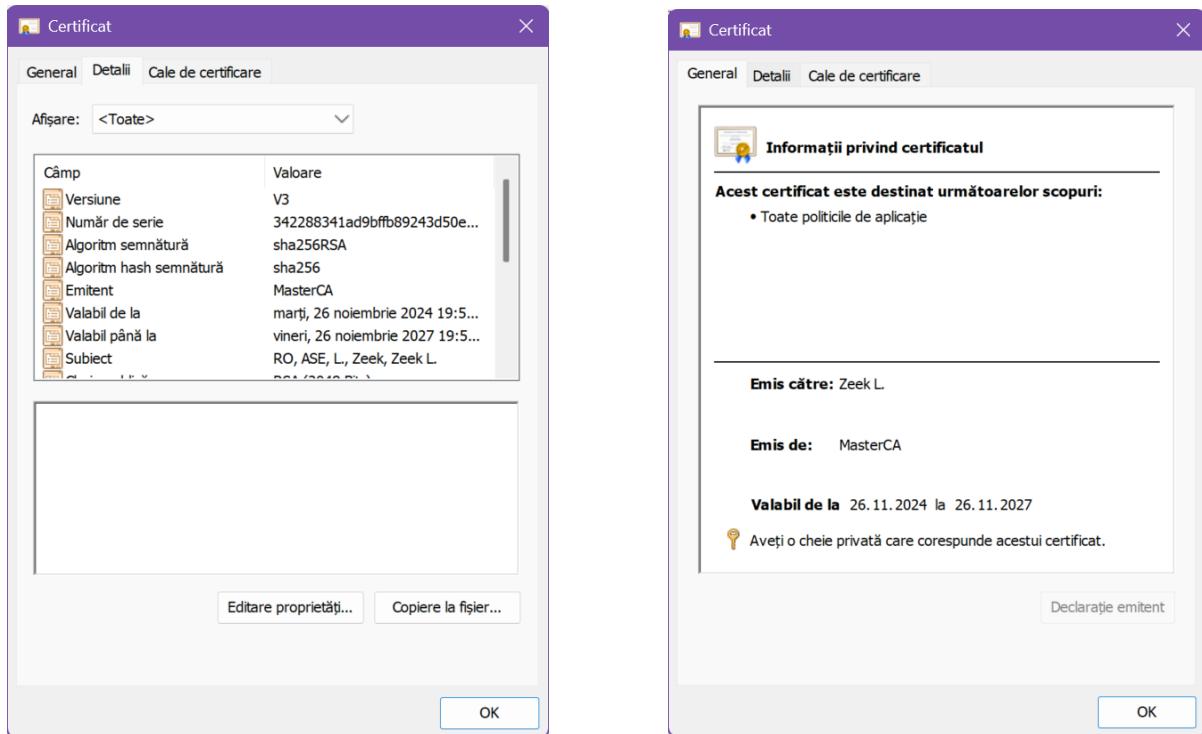
Please choose a key length, then click 'Enroll' to fetch your certificate.

Options

Leave values as default if unsure.

Key specification:	RSA 2048 bits
Certificate profile:	User_Sign
<input type="button" value="Enroll"/>	

Then, I imported the certificate into Windows Store:



certmgr - [Certificates - Current User\Personal\Certificates]

Fisier	ACTIONE	Vizualizare	Ajutor
Certificates - Current User			
Personal			
Certificates			
> Autorități radacina de certificare			
> Aprobate de firmă			
> Autorități intermedie de certificare			
> Obiect utilizator Active Directory			
> Editori de încredere			
> Certificate care nu sunt de încredere			
> Autorități terțe radacina de certificare			
> Persoane de încredere			
> Emitenți autentificare client			
> Alte persoane			
> AdobeCertStore			
> Rădăcini de încredere pentru dispozitive			
Trusted Devices			

Issued To Issued By Expiration Date Intended Purposes Friendly Name Status Certif.

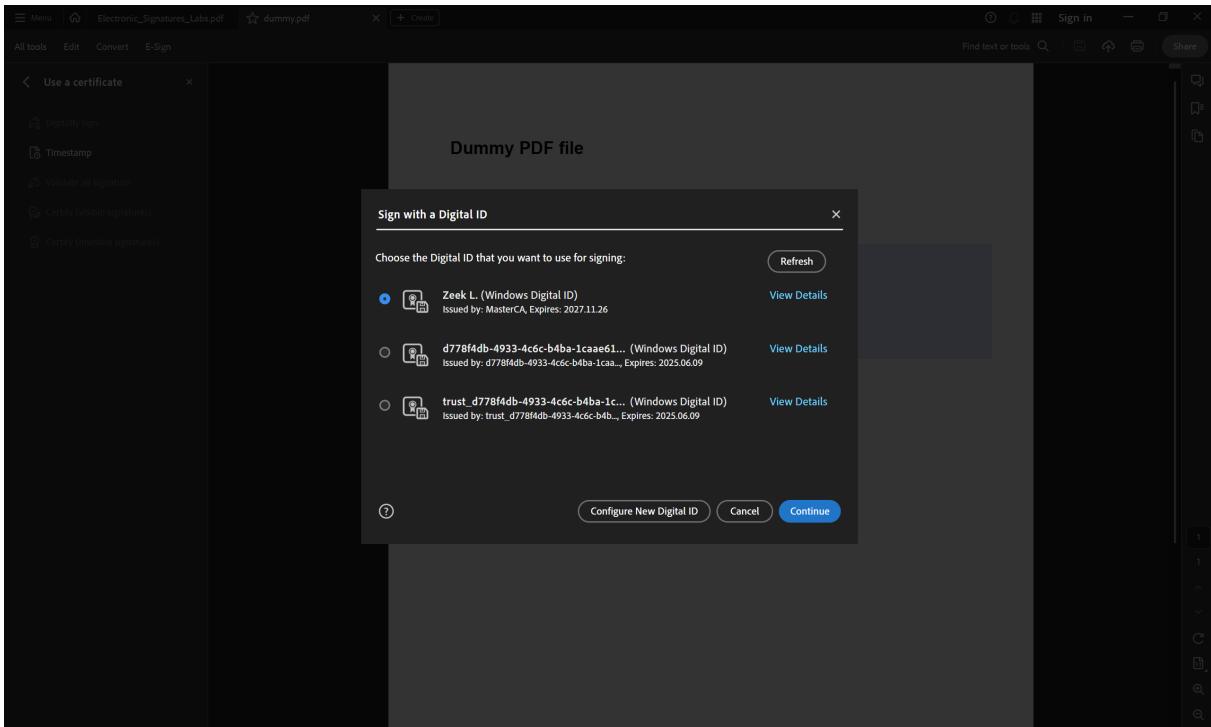
Adobe Content Certificate 10-5	Adobe Intermediate CA 10-3	18.08.2025	<All>	<None>		
Adobe Content Certificate 10-6	Adobe Intermediate CA 10-4	18.08.2025	<All>	<None>		
Adobe Content Certificate 10-7	Adobe Intermediate CA 10-15	05.08.2030	<All>	<None>		
Adobe Content Certificate 10-8	Adobe Intermediate CA 10-19	05.08.2030	<All>	<None>		
Adobe Intermediate CA 10-15	Adobe Root CA 10-3	04.08.2068	<All>	<None>		
Adobe Intermediate CA 10-19	Adobe Root CA 10-3	04.08.2068	<All>	<None>		
Adobe Intermediate CA 10-3	Adobe Root CA 10-3	04.08.2068	<All>	<None>		
Adobe Intermediate CA 10-4	Adobe Root CA 10-3	04.08.2068	<All>	<None>		
d778f4db-4933-4c6c-b4ba-1caa...	d778f4db-4933-4c6c-b4ba-1caa...	09.06.2025	<All>	Microsoft Your Phone		
trust_d778f4db-4933-4c6c-b4ba...	trust_d778f4db-4933-4c6c-b4ba-1...	09.06.2025	<All>	Microsoft Your Phone		
Zeek L.	MasterCA	26.11.2027	<All>	Zeek L.		

Personal store contains 11 certificates.

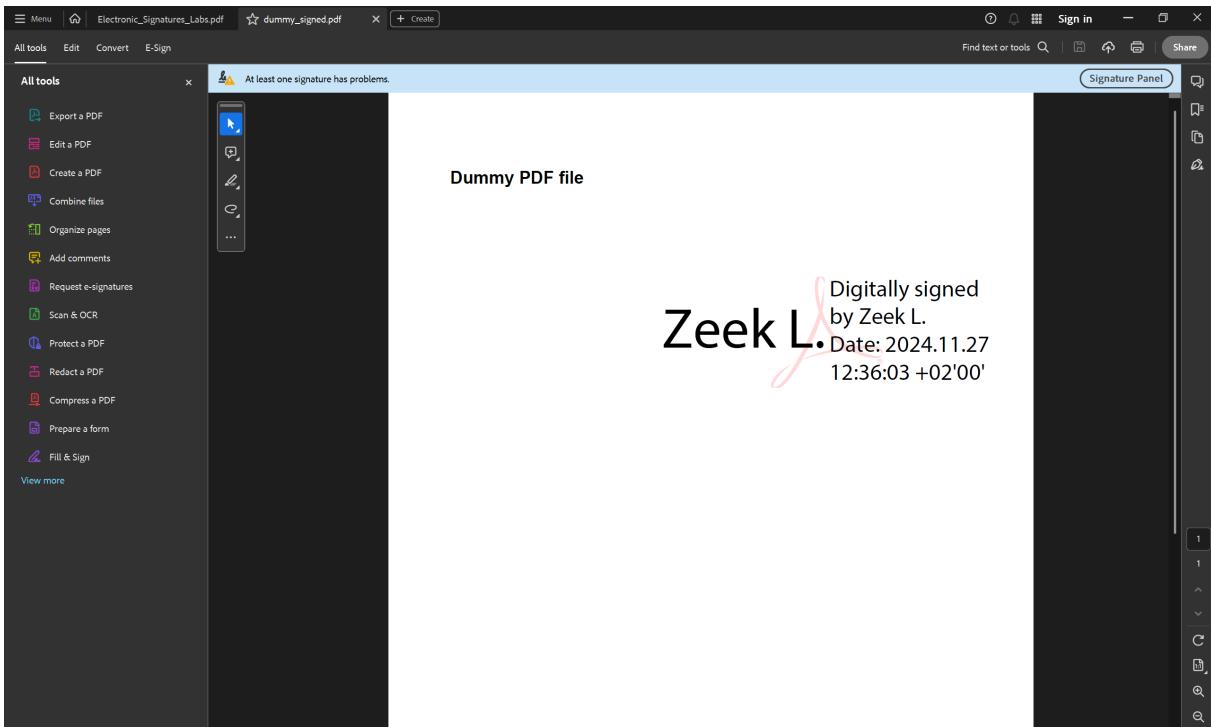
PDF Signatures

1 Sign a document

The PDF document can be signed as shown on the live lab. I will choose a dummy [PDF](#) found online from official W3C website.

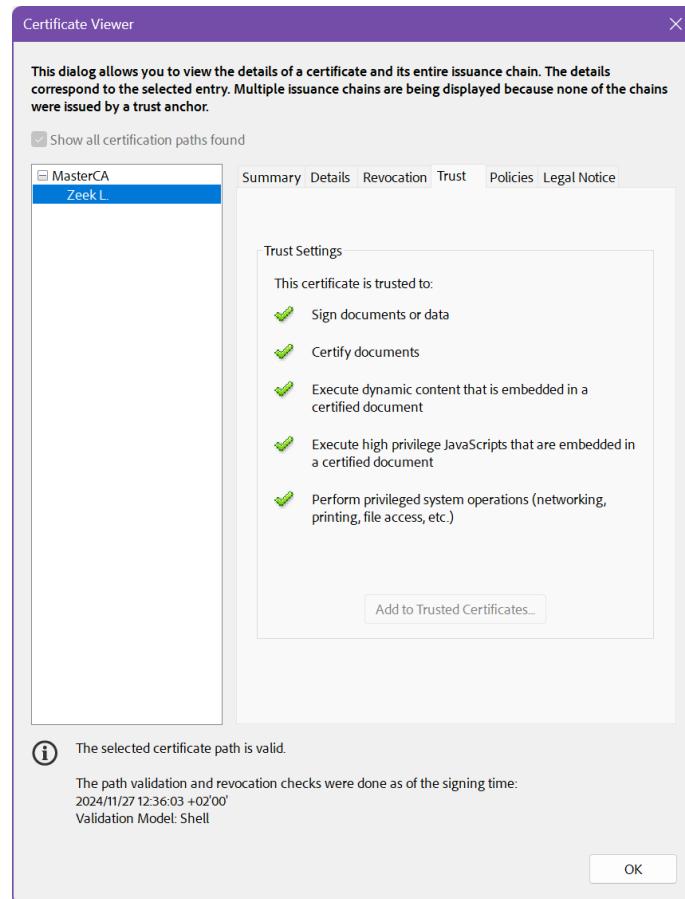


I am signing with the certificate generated from the ejbca CA and imported to Windows Store. This is the result.



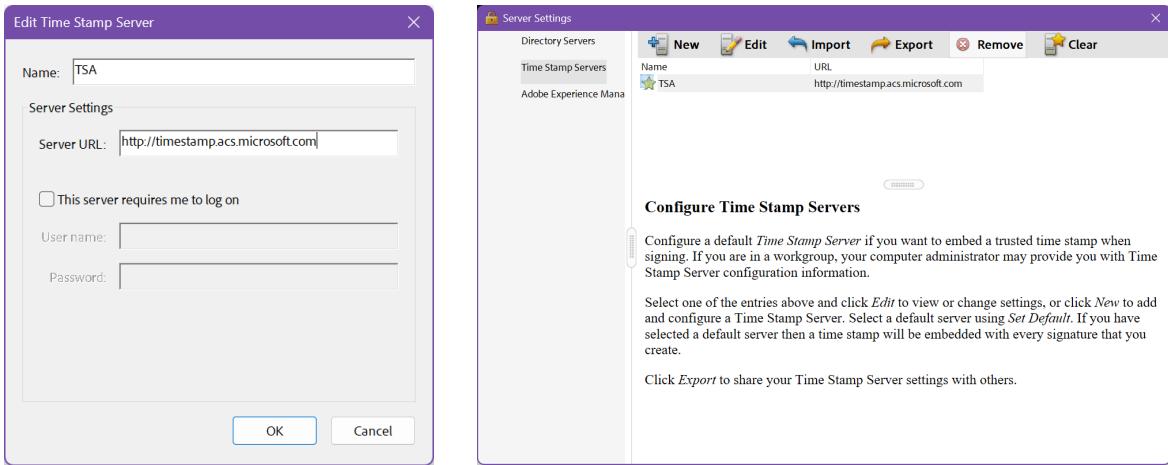
Validate a document

To validate the document, we go to the validation panel. I port-forwarded the required port for OCSP and the signature is validated. The only problem is that the revocation status of the MasterCA cannot be established due to no specification of a CLR or another method for the CA. If we add MasterCA to the Trusted CAs, everything goes smoothly and the chain is fully validated.



-
- ③ Search for a free timestamp server. Hint: github
 - ④ Apply a timestamp to the document
 - ⑤ Validate the document and check the signature format this time

Disclaimer: I did not succeed the timestamping, but I will document the process as discussed in the live lab. First, we will have to choose a TSA server, like freetlsa.org or timestamp.acs.microsoft.com. Add it to Adobe Acrobat timestamping settings and set it as default:

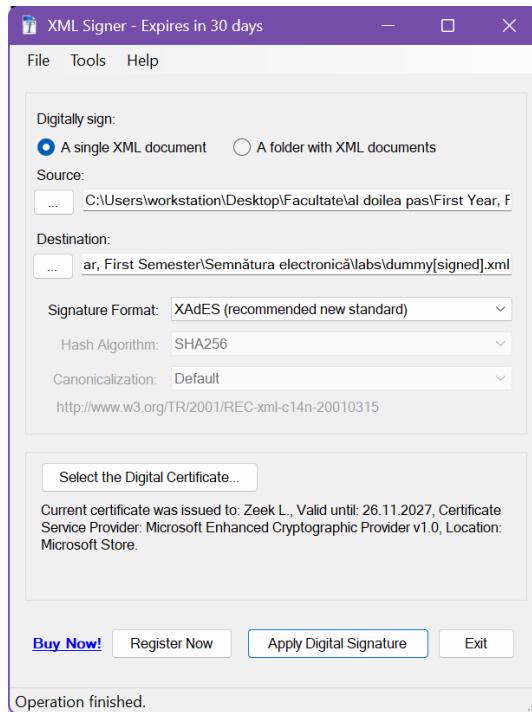


After that use the **Apply a document timestamp** from the **Use a certificate** sub-menu. The result will be a timestamped signed document.

XML Signatures

1 Sign an XML document

We use the XML Signer tool to do that on a dummy XML from the official Microsoft [site](#).



2 Validate the signed document

To validate the signed XML, I will use a Linux tool called `xmlsec1`. After I added the full chain of certificates to the Kali's trust store, I executed the command `xmlsec1 --verify --trusted-pem full_chain.pem dummy\[signed\].xml` from the directory of the signed XML. The result:

```
[root@kali] [/home/kali/Desktop/shared]
# xmlsec1 --verify --trusted-pem full_chain.pem dummy\[signed\].xml

OK
SignedInfo References (ok/all): 2/2
Manifests References (ok/all): 0/0
```

3 Validate an altered version of the document

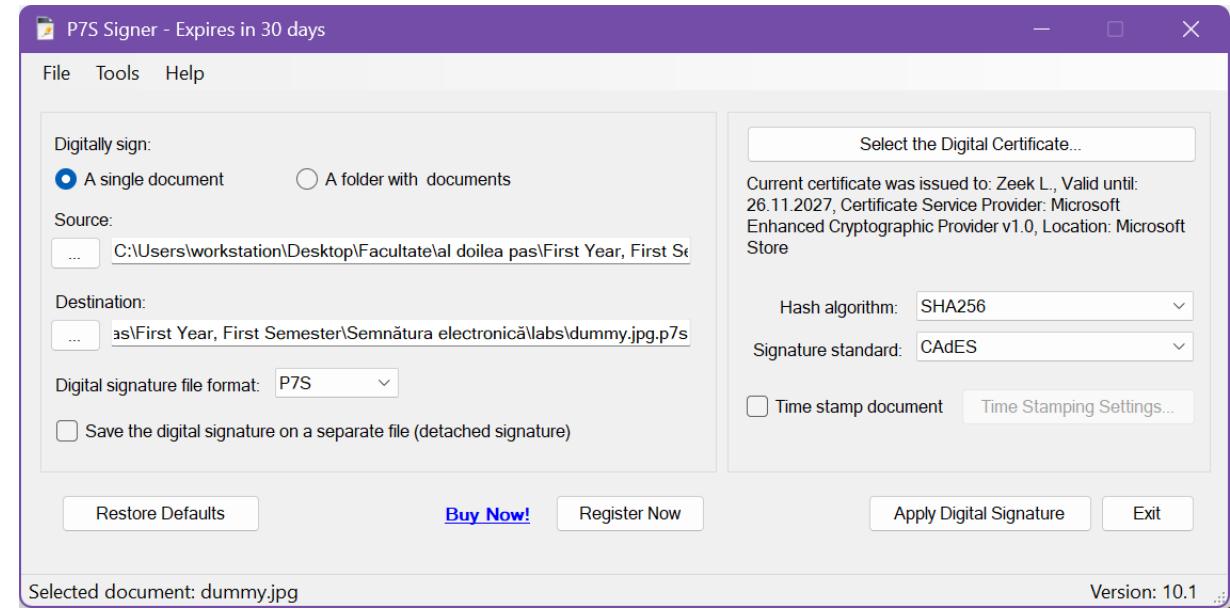
I will edit the price of a book. Let's say **Midnight Rain by Kim Ralls**. The new price will be **5,96**. Only the last digit will be modified! The signature is now invalid because hash of the document has changed.

```
[root@kali] [/home/kali/Desktop/shared]
# xmlsec1 --verify --trusted-pem full_chain.pem dummy\[signed\].xml
func=xmlSecOpenSSLEvpDigestVerify:file=digests.c:line=355:obj=sha256:subj=unknown:error=12:invalid data:data and digest do not match
FAIL
SignedInfo References (ok/all): 0/1
Manifests References (ok/all): 0/0
Error: failed to verify file "dummy[signed].xml"
```

PKCS#7 Signatures

- 1 Use a media document - a picture, a text file, or other file type.
- 2 Obtain an attached signature

To obtain an attached signature of a picture, we will use P7S with the following settings:



③ Validate it

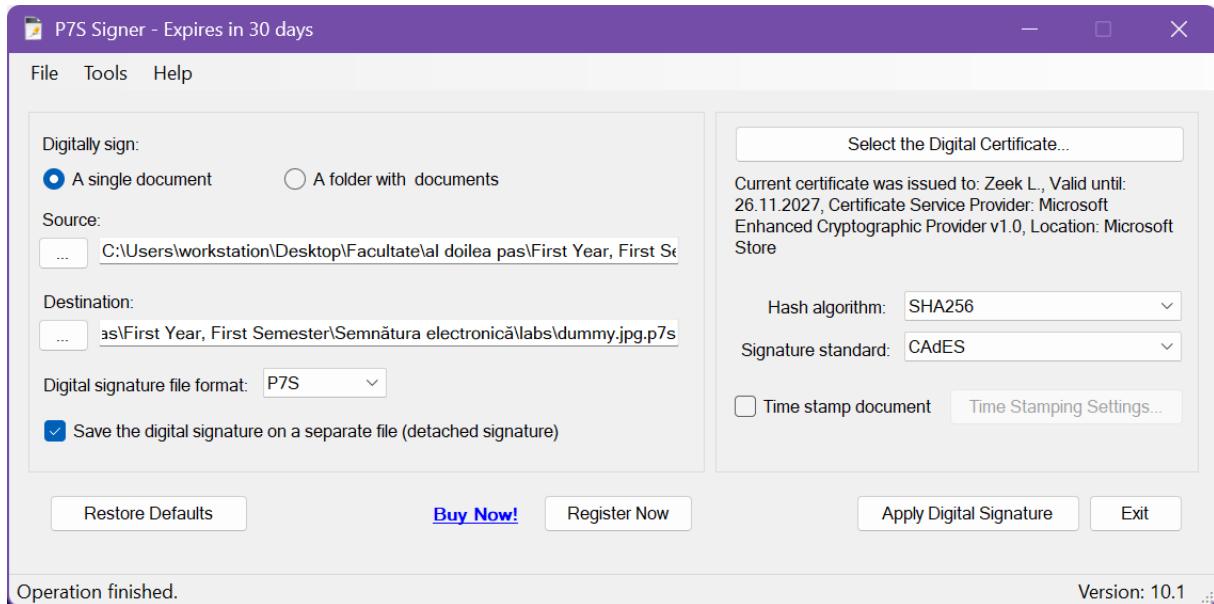
To validate the signature we use this command `openssl smime -verify -inform DER -in dummy.jpg.p7s -content dummy.jpg -CAfile full_chain.pem > /dev/null.` The result:

```
root@kali: /home/kali/Desktop/shared
File Actions Edit View Help
root@kali: /home/kali/Desktop/ejbca x root@kali: /home/kali/Desktop/shared x
root@kali: ~
└── (root㉿kali)-[~/Desktop/shared]
    # openssl smime -verify -inform DER -in dummy.jpg.p7s -content dummy.jpg -CAfile full_chain.pem > /dev/null
    Verification successful
```

④ Obtain a detached signature

⑤ Validate it

To obtain a detached signature we just check the box to save the signature on a separate file from the P7S Signer.



In order to validate the detached signature, we use the command `openssl smime -verify -inform DER -in dummy_detached.jpg.p7s -content dummy.jpg -CAfile full_chain.pem > /dev/null`. The result:

```
(root㉿kali)-[~/Desktop/shared] # openssl smime -verify -inform DER -in dummy_detached.jpg.p7s -content dummy.jpg -CAfile full_chain.pem > /dev/null
Verification successful
```

⑥ Check the obtained file with ASN.1 viewer.

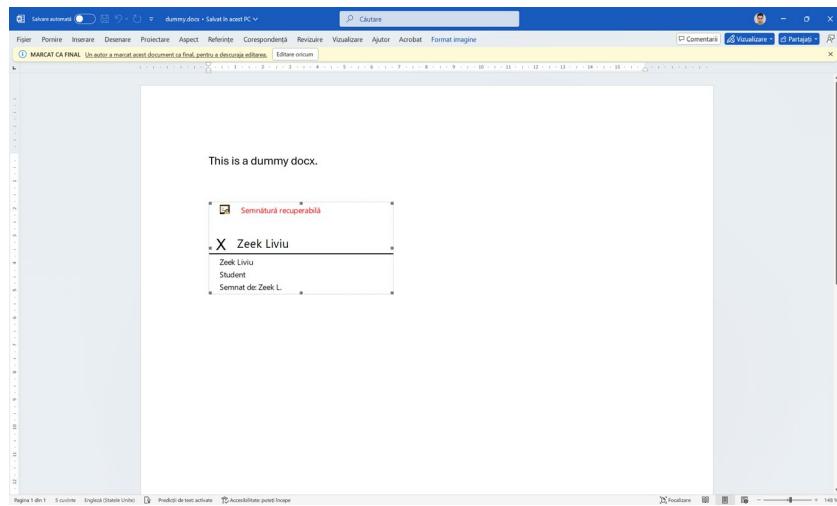
I will use the standard command from OpenSSL: `openssl asn1parse -inform DER -in dummy_detached.jpg.p7s`. The output (a snapshot of it) will be:

```
C:\Windows\System32\cmd.e x + v
91:d=7 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
102:d=7 hl=2 l= 0 prim: NULL
104:d=6 hl=2 l= 19 cons: SEQUENCE
106:d=7 hl=2 l= 17 cons: SET
108:d=8 hl=2 l= 15 cons: SEQUENCE
110:d=9 hl=2 l= 3 prim: OBJECT :commonName
115:d=9 hl=2 l= 8 prim: UTF8STRING :MasterCA
125:d=6 hl=2 l= 30 cons: SEQUENCE
127:d=7 hl=2 l= 13 prim: UTCTIME :241126175448Z
142:d=7 hl=2 l= 13 prim: UTCTIME :271126175447Z
157:d=6 hl=2 l= 73 cons: SEQUENCE
159:d=7 hl=2 l= 16 cons: SET
161:d=8 hl=2 l= 14 cons: SEQUENCE
163:d=9 hl=2 l= 3 prim: OBJECT :commonName
168:d=9 hl=2 l= 7 prim: UTF8STRING :Zeek L.
177:d=7 hl=2 l= 13 cons: SET
179:d=8 hl=2 l= 11 cons: SEQUENCE
181:d=9 hl=2 l= 3 prim: OBJECT :givenName
186:d=9 hl=2 l= 4 prim: UTF8STRING :Zeek
192:d=7 hl=2 l= 11 cons: SET
194:d=8 hl=2 l= 9 cons: SEQUENCE
196:d=9 hl=2 l= 3 prim: OBJECT :surname
201:d=9 hl=2 l= 2 prim: UTF8STRING :L.
205:d=7 hl=2 l= 12 cons: SET
207:d=8 hl=2 l= 10 cons: SEQUENCE
```

Microsoft Word Signatures

1 Sign a document using Microsoft Word.

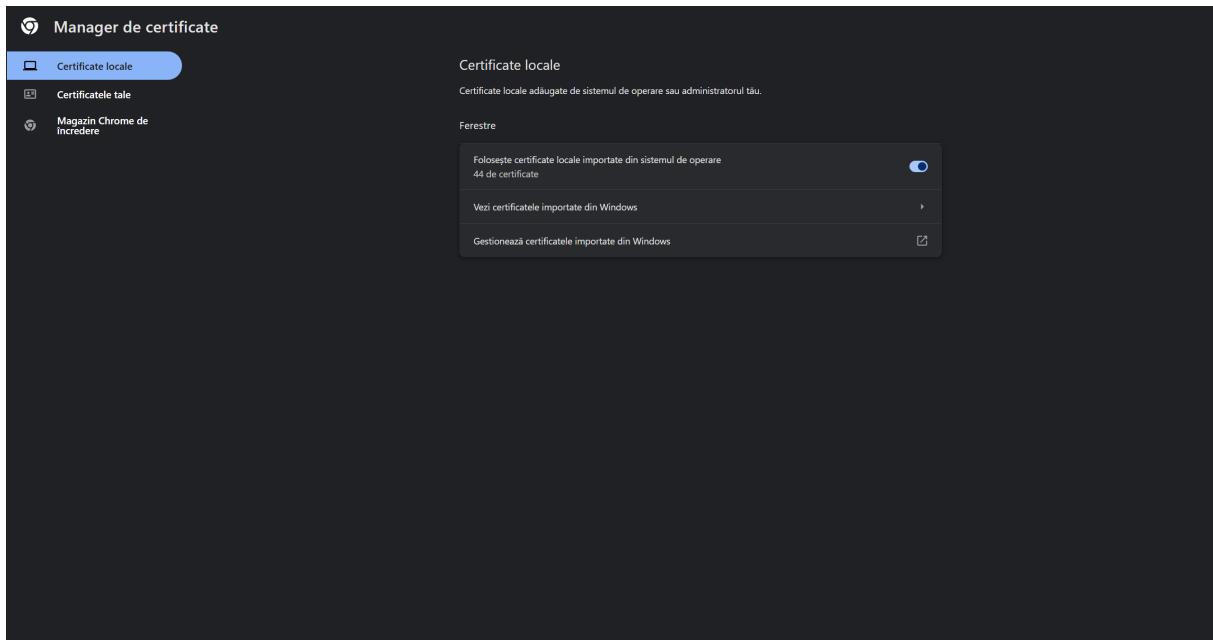
I did follow the steps from [here](#). The result:



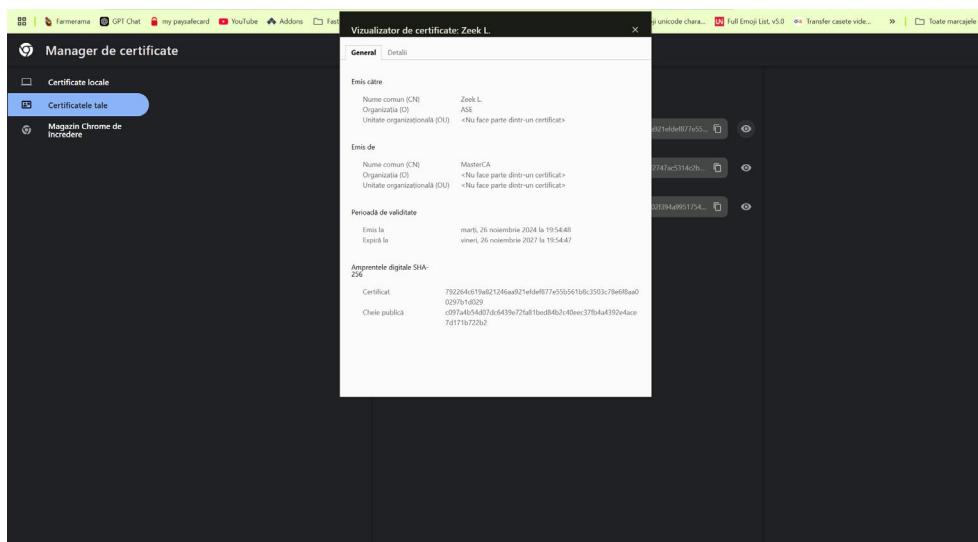
Stores

- ➊ Prove that we are using different stores to validate the certificate of the web server.
- ➋ We can use <https://testssl.certsign.ro/>
- ➌ Hint: Should use different browsers (Mozilla+Chrome, for example)

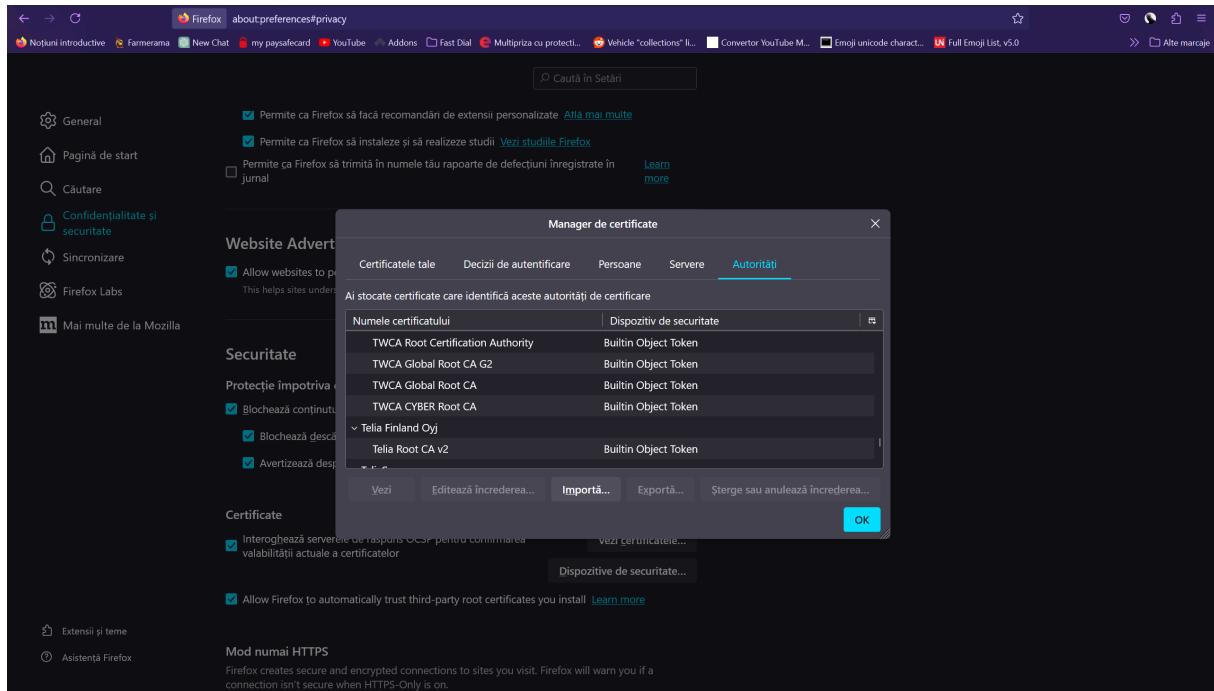
We will start with Chrome. Chrome uses the Windows Trust store. We can see that in the settings:



Here we can see our certificate:



Now let's head to Firefox.



Here we can see Mozilla's own trusted authorities. We can also see our certificate on the **Own certificates** tab, but MasterCA authority (which is the authority that emitted our certificate) is not on the list of trusted authorities here.

Certificate Requests

1 Generate a certificate request using openssl.

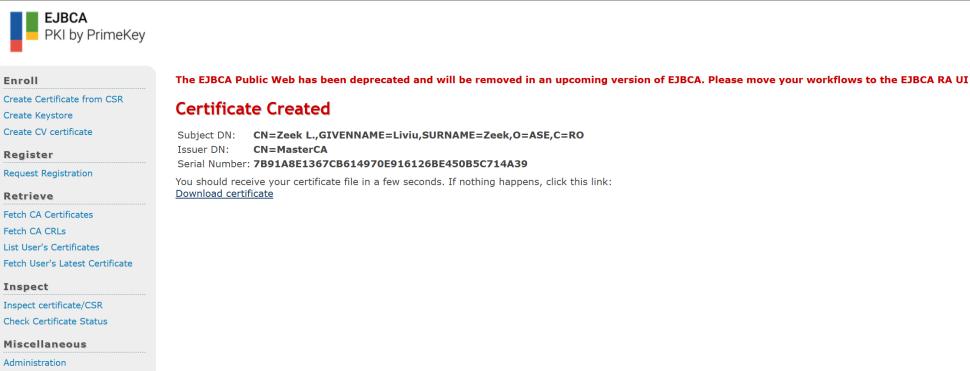
We use the provided command (adapted to our use case) **openssl req -newkey rsa:2048 -keyout private.key -out openssl.csr** and fill in the required fields. Here is the result:

```
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> openssl req -newkey rsa:2048 -keyout private.key -out openssl.csr
-----+
Enter PEM pass phrase:
-----+
Verifying - Enter PEM pass phrase:
-----+
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----+
Country Name (2 letter code) [AU]:RO
State or Province Name (full name) [Some-State]:Romania
Locality Name (eg, city) []:Bucharest
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Academia de Studii Economice Bucuresti
Organizational Unit Name (eg, section) []:CSIE
Common Name (e.g. server FQDN or YOUR name) []:Zeek L.
Email Address []:zecheruliviu21@stud.ase.ro

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:kali
An optional company name []:ASE
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs>
```

② Use the generated CSR to issue a new certificate, from EJBCA

We first must create an end entity from the admin interface. Then we navigate to the user interface and fill in the fields for enrollment via CSR. We will get a .pem file which will be our certificate.

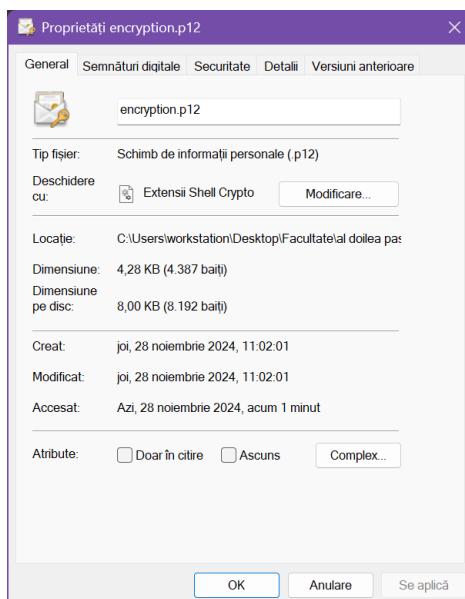


-
- ③ Use openssl to create a pkcs12 file from certificate and private key

Again we use the given command (adapted to our use case) **openssl pkcs12 -export -in encryption.crt -inkey enc_private.key -out encryption.p12**. Here is the result:

```
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> openssl pkcs12 -export -in .\ZeekL.pem -inkey .\private.key -out encryption.p12
Enter pass phrase for .\private.key:
Enter Export Password:
Verifying - Enter Export Password:
PS C:\Users\workstation\Desktop\Facultate\al doilea pas\First Year, First Semester\Semnătura electronică\labs> |
```

And the PKCS#12 output file:

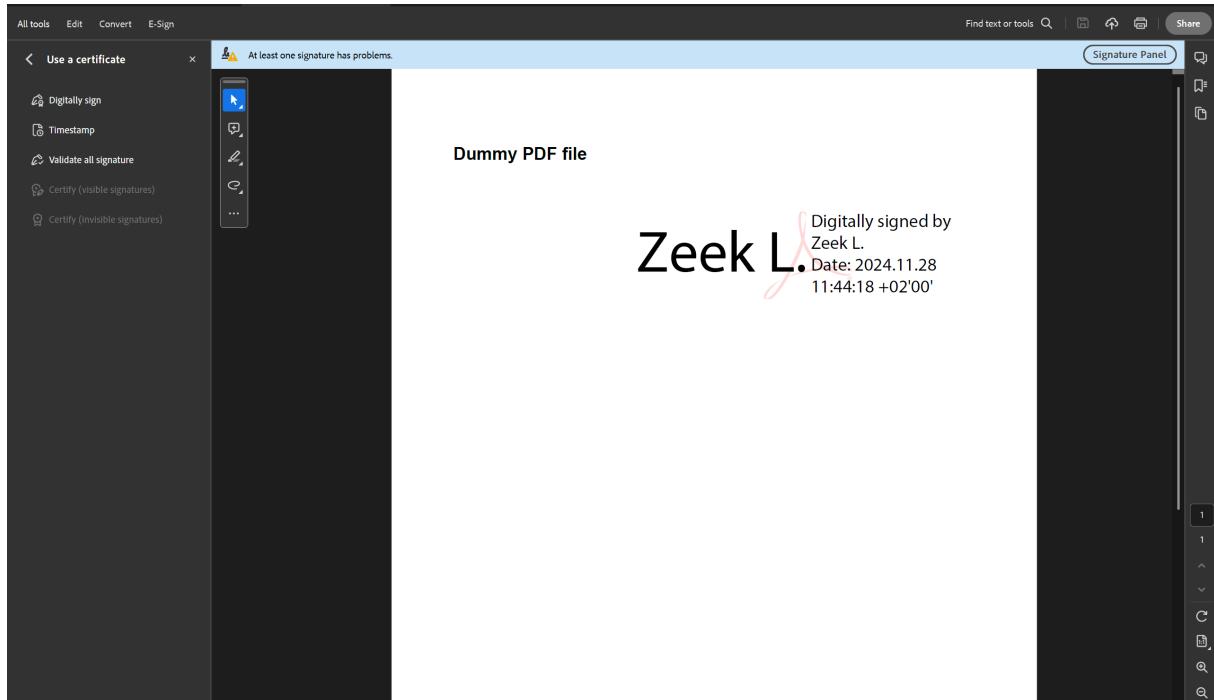
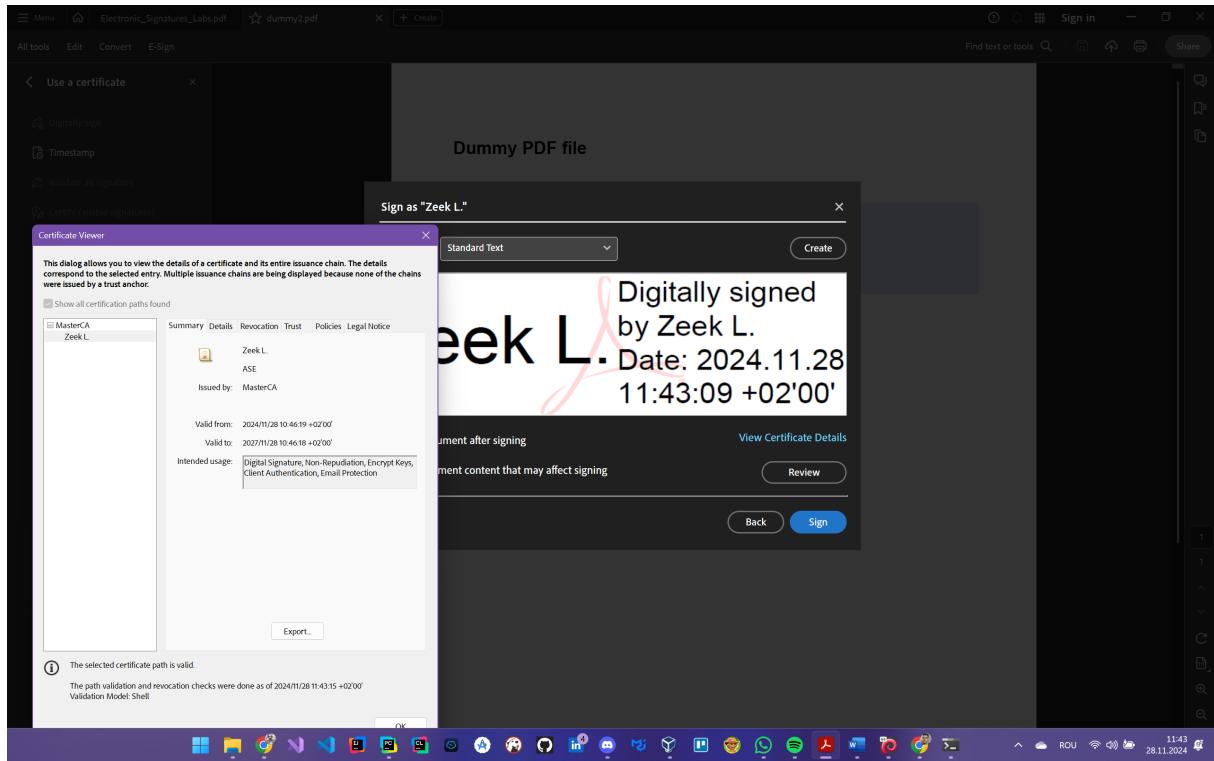


-
- ④ Import the certificate to Windows store

- ⑤ Try to sign a PDF file

We import the certificate as the previous one, by following the wizard steps we followed in the live lab.

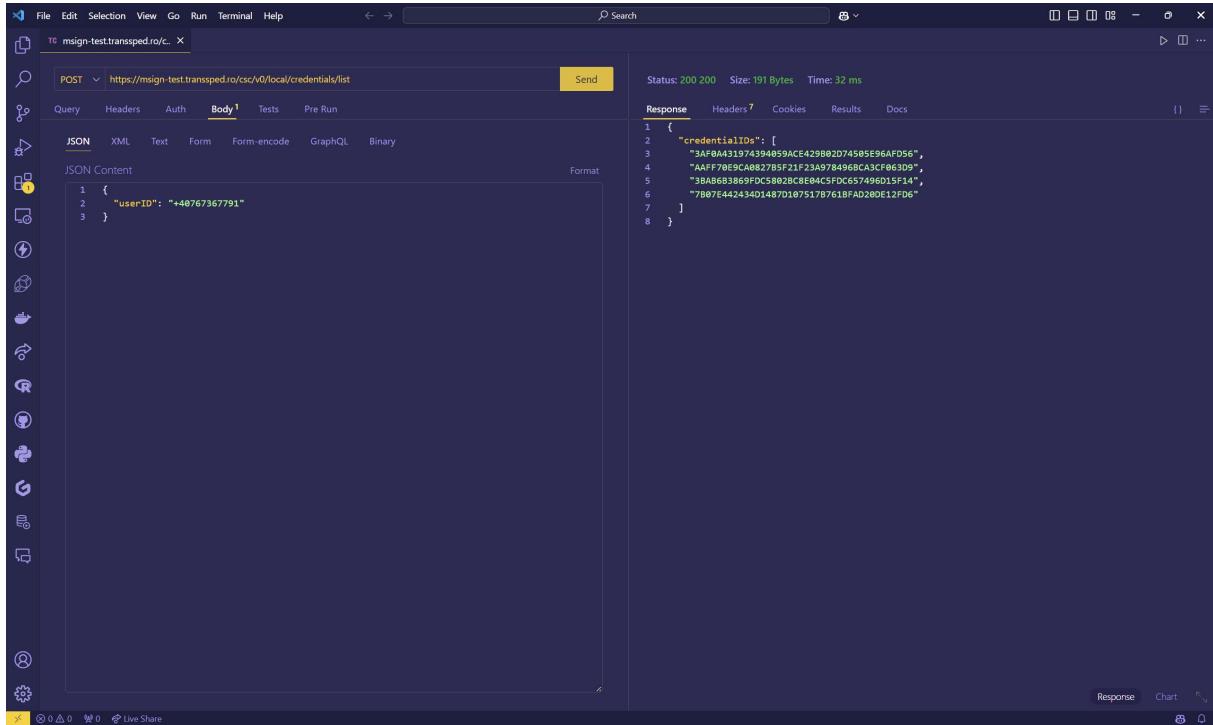
To sign, we use the steps from the live lab. I will use a copy of the **dummy.pdf** I used earlier. Here is the result:



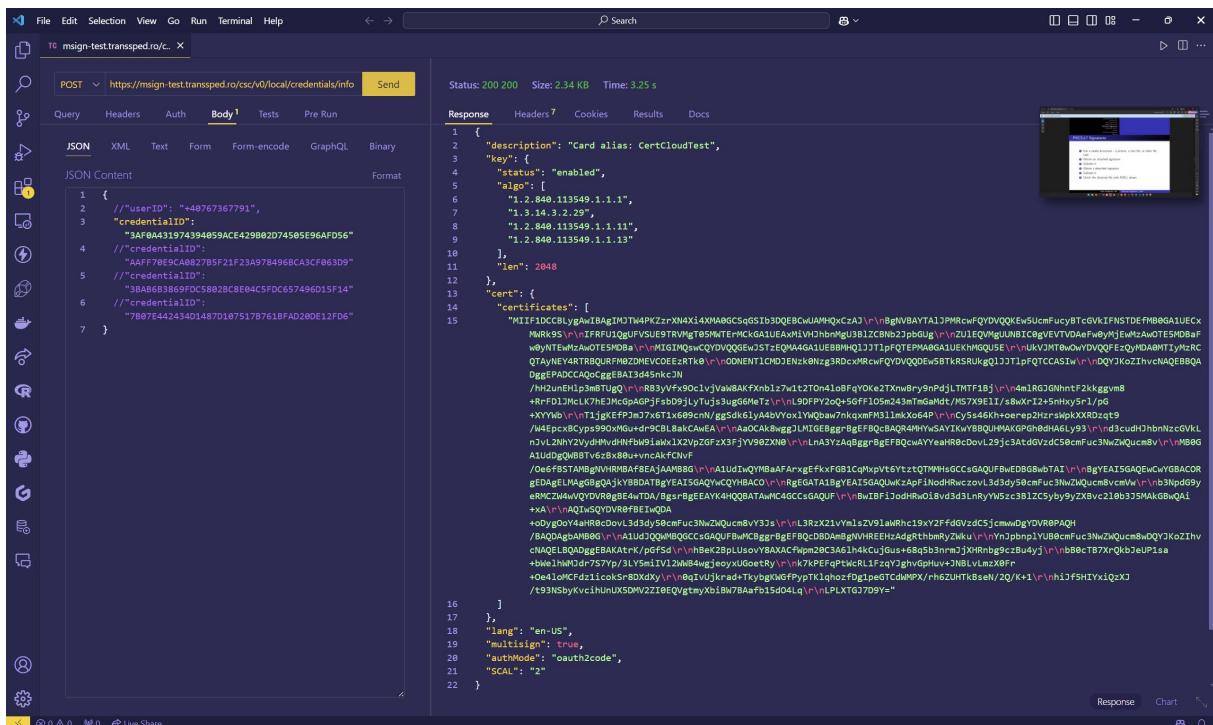
CSC Protocol – ‘Hello world’ level

Obtain one of the cloud certificates of the user with phone number +40767367791, using the CSC protocol.

Let's head to a REST API Client like **Thunder Client** extension from VS Code. Do the following POST to retrieve credentials IDs.



Now, to get the cloud certificate we have to do a POST like this:



And here we have our cloud certificate.

Bonus - Python Certificate Requests

The Python source code:

```
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.x509.oid import NameOID
import cryptography.x509 as x509

# Generate an RSA private key
def generate_private_key():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    return private_key

# Generate a Certificate Signing Request (CSR)
def generate_csr(private_key, country, state, locality, org, org_unit,
common_name, email):
    # Subject details
    subject = x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, country),
        x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, state),
        x509.NameAttribute(NameOID.LOCALITY_NAME, locality),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, org),
        x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, org_unit),
        x509.NameAttribute(NameOID.COMMON_NAME, common_name),
        x509.NameAttribute(NameOID.EMAIL_ADDRESS, email),
    ])

    # Create CSR
    csr = x509.CertificateSigningRequestBuilder().subject_name(subject).sign(
        private_key, hashes.SHA256()
    )
    return csr

# Save private key to file
def save_private_key(private_key, filename):
    with open(filename, "wb") as f:
        f.write(
            private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.TraditionalOpenSSL,
                encryption_algorithm=serialization.NoEncryption(),
            )
        )
```

```

# Save CSR to file
def save_csr(csr, filename):
    with open(filename, "wb") as f:
        f.write(csr.public_bytes(serialization.Encoding.PEM))

# Main function
if __name__ == "__main__":
    # CSR details
    country = "US"
    state = "California"
    locality = "San Francisco"
    organization = "Example Corp"
    organizational_unit = "IT Department"
    common_name = "example.com"
    email_address = "admin@example.com"

    # Generate private key
    private_key = generate_private_key()
    save_private_key(private_key, "private_key.pem")

    # Generate CSR
    csr = generate_csr(
        private_key, country, state, locality, organization,
        organizational_unit, common_name, email_address
    )
    save_csr(csr, "certificate_request.csr")

print("Private key and CSR have been generated.")

```

Next, the same steps as generating the CSR with **OpenSSL** must be followed to sign the PDF. I will not reproduce them again because I see it as a redundancy.

Bonus – Coding

- Write a python script that can create a PKCS#7 or an XML signature.

```

from M2Crypto import SMIME, X509, BIO

# Paths to necessary files
PRIVATE_KEY_FILE = "private.key"  # Path to your private key
CERTIFICATE_FILE = "certificate.pem"  # Path to your certificate
DATA_FILE = "dummy.txt"  # Path to the file to be signed
OUTPUT_SIGNATURE = "signed_dummy.p7s"  # Output PKCS#7 signature file

```

```
def create_pkcs7_signature(data_file, private_key_file, certificate_file,
output_signature):
    # Create an SMIME object
    smime = SMIME.SMIME()

    # Load the signer's certificate
    smime.load_key(private_key_file, certificate_file)

    # Read the data to be signed
    with open(data_file, "rb") as f:
        data_bio = BIO.MemoryBuffer(f.read())

    # Create a PKCS#7 signature
    pkcs7 = smime.sign(data_bio, SMIME.PKCS7_DETACHED)

    # Write the signature to a file
    with open(output_signature, "wb") as f:
        out_bio = BIO.File(f)
        smime.write(out_bio, pkcs7, data_bio)

    print(f"PKCS#7 signature created and saved to {output_signature}")

if __name__ == "__main__":
    create_pkcs7_signature(DATA_FILE, PRIVATE_KEY_FILE, CERTIFICATE_FILE,
OUTPUT_SIGNATURE)
```

The verification went smoothly with the same approach of **OpenSSL**.