



Relatório do Desafio LabSec

Aluno: Pedro Henrique Scheidt Prazeres

Data: 28/10/2024

Sumário:

[Introdução](#)

[Questão 1](#)

[Resposta](#)

[Processo](#)

[Questão 2](#)

[Etapa 1](#)

[Solução](#)

[Processo](#)

[Links Utilizados](#)

[Etapa 2](#)

[Etapa 3](#)

[Questão 3](#)

[Status da Assinatura](#)

[Status do Certificado](#)

[Status do Carimbo de Tempo](#)

[Questão 4](#)

[Questão 5](#)

[Etapa 1](#)

[Etapa 2](#)

[Conclusões](#)

Introdução

No início do desafio eu sabia pouco sobre os assuntos, contudo essas dúvidas geraram aprendizado. Aprendi bastante e rapidamente sobre o assunto, há coisas que não fiz, mas as que fiz representam uma boa evolução. Devo deixar claro, no arquivo helper está escrito que “O objetivo do desafio não é avaliar o seu grau de conhecimento, mas sim como você se comporta ao se deparar com um desafio.”. Então acho importante deixar claro, o normal quando me deparo com uma dúvida é pesquisar por conta própria por um tempo, mas se isso não der resultado, meu comportamento costuma ser perguntar à alguém (normalmente aquele que me passou a atividade) como fazer; contudo, como perguntar ao pessoal do labsec e aos professores estragaria a ideia de um desafio, escolhi me abster desse caminho.

Questão 1

Resposta

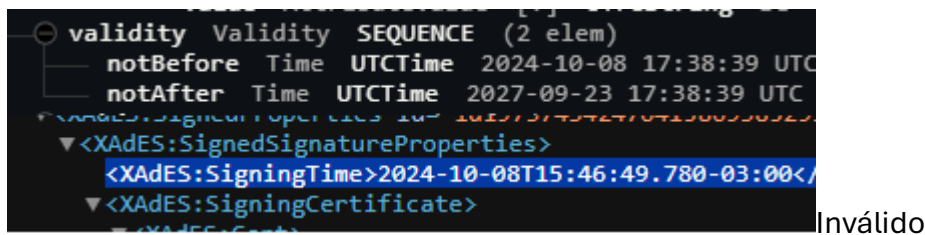
Essa é a resposta mais longa no relatório, pois é [impossível provar inexistência](#) (é possível em matérias exatas, mas eu não tenho o conhecimento de todos os testes possíveis). Toda vez que um teste não retornava uma falha, eu acreditava que é porque não havia encontrado o erro ainda, então levei muito tempo buscando um erro que pode ou não existir. Colocarei os erros que achei aqui, mas é possível ver todo meu processo de pensamento no processo.

cms_1.p7s está fora da validade:

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 21 (0x15)
    Signature Algorithm: sha512WithRSAEncryption
    Issuer: CN=THE_FINAL_COUNTDOWN/serialNumber=3, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
    Validity
      Not Before: Oct  8 17:38:53 2024 GMT
      Not After : Oct  8 17:39:03 2024 GMT
    Subject: CN=WHATS_MY_AGE_AGAIN/serialNumber=21, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
```

inválido

xades_b_t foi assinado antes de seu prazo começar a valer:

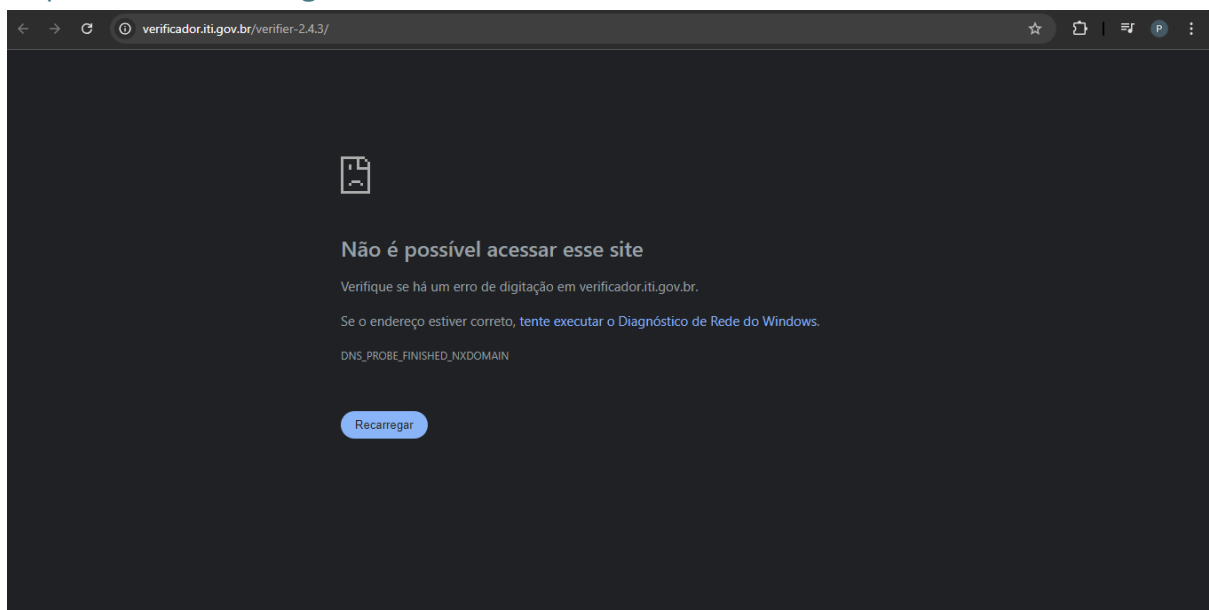


Processo

O processo foi longo e levou multiplas tentativas e processos diferentes. Não é necessário ler tudo nessa sessão, é apenas para mostrar minhas tentativas.

Primeiro tentei utilizar o verificador de assinaturas ICP do gov.br, mas ele não estava disponível:

<https://verificador.iti.gov.br/verifier-2.4.3/>



Converti os arquivos .xml em .p7b pegando o conteúdo da assinatura, colocando em um arquivo .pem e convertendo em .p7b pelo openssl

Verifiquei os arquivos .p7s e .p7b por meio do comando do openssl:

```
> openssl pkcs7 -in [NOME DO ARQUIVO].p7s -inform DER -print_certs -text
```

Isso me permitiu ler as características das assinaturas;

Embora não sejam emitidos por uma entidade certificadora oficial do ICP, sendo emitidos pela UFSC, escolhi ignorar esse detalhe e considerar os certificados que apresentam somente esse problema válidos apesar disso, uma vez que se trata de um exercício.

Observei que o arquivo “cms_1” estava fora da validade, o tornando inválido.

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 21 (0x15)
  Signature Algorithm: sha512WithRSAEncryption
  Issuer: CN=THE_FINAL_COUNTDOWN/serialNumber=3, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
  Validity
    Not Before: Oct  8 17:38:53 2024 GMT
    Not After : Oct  8 17:39:03 2024 GMT
  Subject: CN=WHATS_MY_AGE_AGAIN/serialNumber=21, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
```

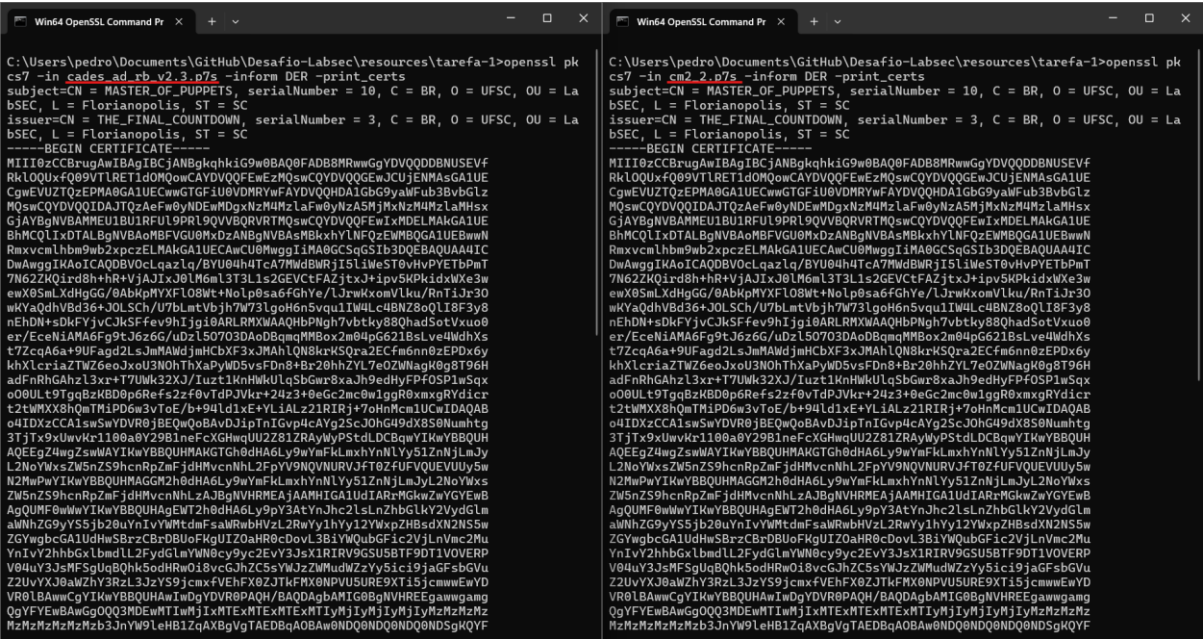
Salvo esse, o conteúdo das assinaturas .p7s visualmente não me levou a nenhuma conclusão, e o outro verificador online de assinaturas do Gov.br não foi útil:

Submeti um documento ao VALIDAR, mas o serviço não reconhece uma ou mais assinaturas. O que está acontecendo?

O VALIDAR entende como "assinatura desconhecida" aquelas assinaturas que não atendem aos critérios definidos na Portaria ITI Nº 22, de 28 de setembro de 2023. É possível que se trate de um modelo de assinatura que possua outras âncoras de confiança não válidas no Brasil. O serviço também não reconhece a assinatura feita através da prática de "envelopamento" de assinaturas, o que geralmente ocorre quando uma pessoa jurídica assina por cima das demais assinaturas (não permitindo que o VALIDAR verifique tais assinaturas diretamente) . Em suma, nesse caso, o serviço VALIDAR vai apresentar apenas a assinatura da pessoa jurídica, que tecnicamente é quem assinou o documento eletrônico.

(Provavelmente pois o órgão emissor é a UFSC e o site não reconhece o emissor)

Depois de mais de um dia investigando as assinaturas, eu tinha certeza que haviam arquivos repetidos:



Até escrevi um código em python que compara o conteúdo das duas assinaturas:

```
temp.py > ...
1 chave1 = 'comparar/cades_ad_rb_v2.3.txt'
2 chave2 = 'comparar/cm2_2.txt'
3
4 with open(chave1, 'r') as arq1, open(chave2, 'r') as arq2:
5     texto1 = arq1.read()
6     texto2 = arq2.read()
7
8 if texto1 == texto2:
9     print("Iguais")
10 else:
11     print("Diferentes")
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\pedro\Documents\GitHub\Desafio-Labsec> python -u "c:\Users\pedro\Iguais

Contudo, notei que os dois arquivos originais possuem tamanhos diferentes:

📄	cades_ad_rb_v2.3	18/10/2024 14...	Assinatura...	79 KB
📄	cm2_2	18/10/2024 14...	Assinatura...	35 KB

Decidi então que a chave da questão estava em encontrar a diferença entre os dois

Comparei os dois arquivos por meio do comando do openssl:

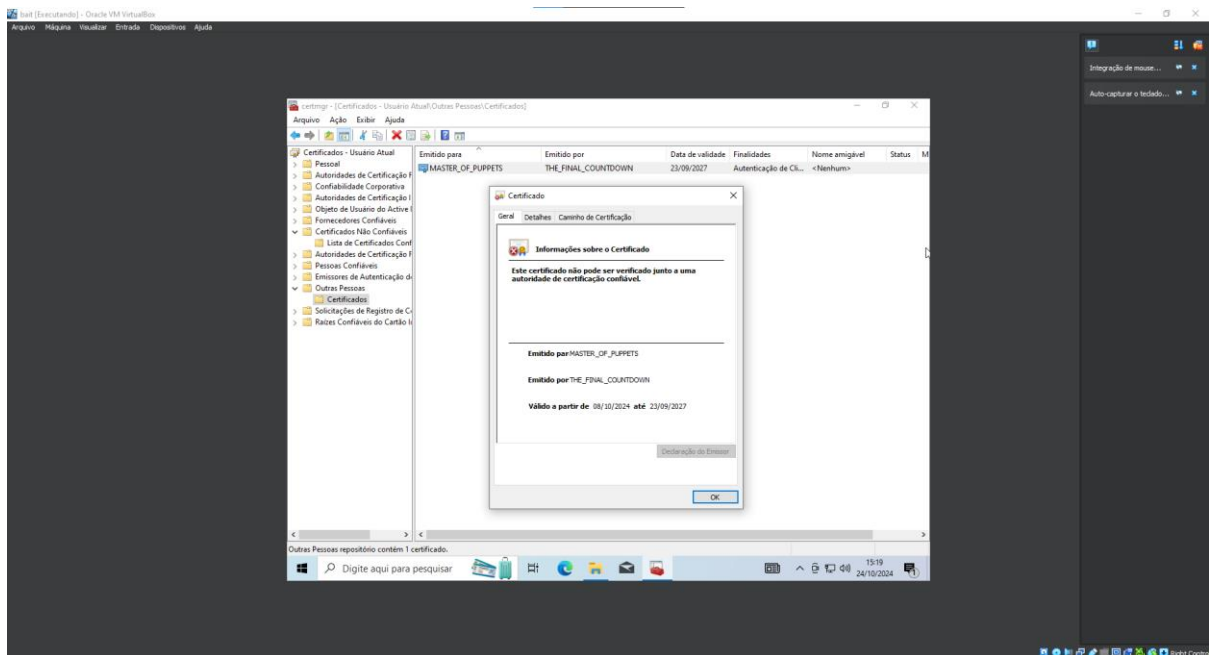
```
> openssl pkcs7 -in [NOME DO ARQUIVO].p7s -inform DER -print_certs -text
```

Novamente os dois arquivos são iguais.

Finalmente, decidi abrir os dois arquivos com o VIM, e encontrei uma diferença, no arquivo cm2_2.p7s há um link embutido.



Assumo que isso foi feito para simular algum tipo de ataque malicioso, embutindo um link desconhecido em uma assinatura. Considerando que acreditei que simula um ataque, eu decidi me proteger e abrir uma VM do Windows antes de tentar instalar.



Para minha surpresa, nada aconteceu, então voltei à estaca zero.

Após isso, tentei verificar o certificado com seu emissor por meio do seguinte comando:

```
> openssl ocsp -issuer [CERTIFICADO EMISSOR] -cert [CERTIFICADO EMITIDO] -url [LINK OCSP]
```

Mas para isso eu deveria conseguir pegar o link OCSP do certificado por meio do comando:

```
openssl x509 -in [NOME DO CERTIFICADO] -noout -ocsp_uri
```

Mas o resultado de todos é semelhante a este:

```
Could not find certificate from cades_ad_rb_v2.3.p7s
685A0000:error:1608010C:STORE routines:ossl_store_handle_load_result:unsupported:crypto\store\store_result.c:151:
```

Então decidi verificar os certificados contra a lista de revogação de seu emissor. Baixei a lista de revogação .crl e comparei com os certificados:


Resultado do cades_ad_rb_V2.3:

Id	terça-feira, 8 de outubr...	-noout -serial serial=0A	Válido
0e	terça-feira, 8 de outubr...		
0f	terça-feira, 8 de outubr...		

Resultado do cms_1:

Id	terça-feira, 8 de outubr...	ial serial=15	Válido
0e	terça-feira, 8 de outubr...		
0f	terça-feira, 8 de outubr...		

Resultado do xades_ad_rt_v1.1:

0d	terça-feira, 8 de outubr...	
0e	terça-feira, 8 de outubr...	
0f	terça-feira, 8 de outubr...	

Considerando que todos deram válido, esse não é o caminho correto.

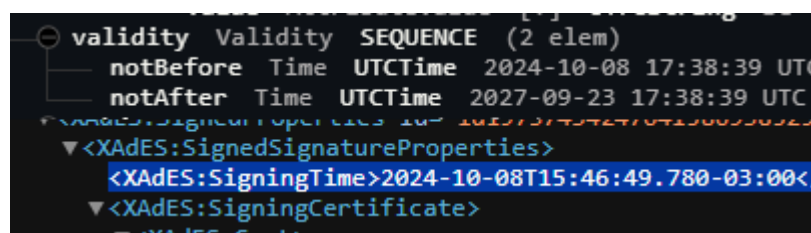
A próxima tentativa envolveu o comando:

```
openssl smime -verify -in [NOME DO ARQUIVO].p7s -noverify -inform DER
```

Esse comando resultou em muitas linhas de nada:

[illegible]

Ao verificar o arquivo xades_b_t notei que ele foi assinado antes de seu prazo iniciar:



Inválido

Questão 2

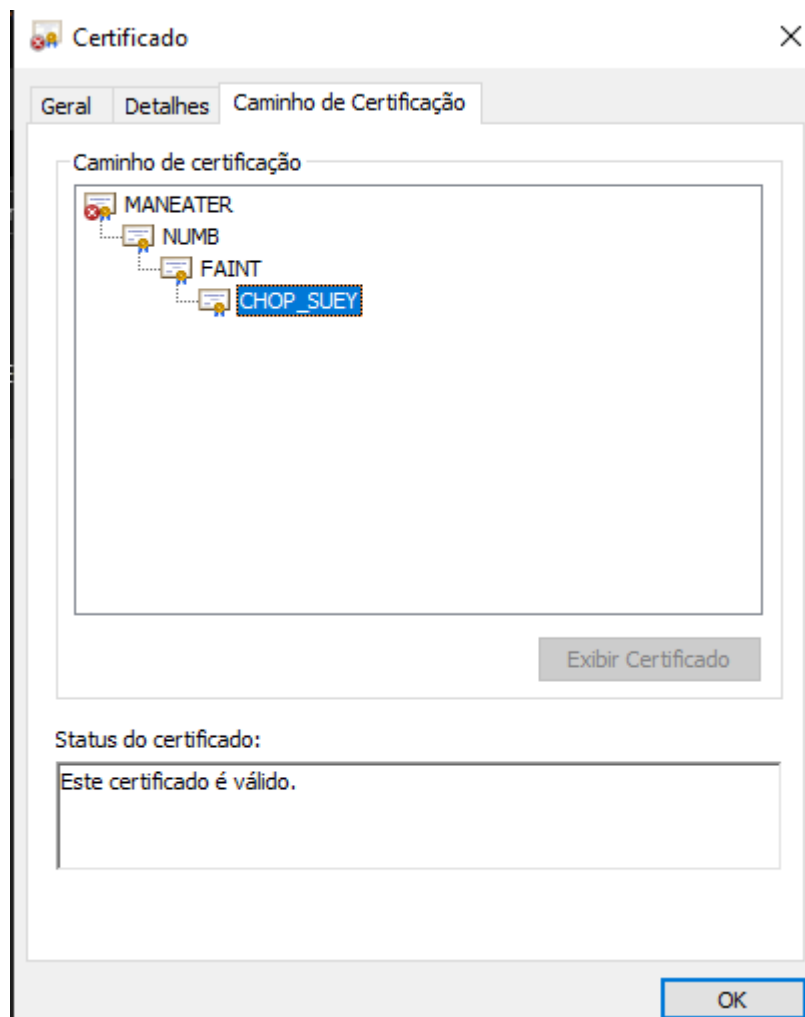
Etapa 1

Solução

Para ver facilmente o caminho do certificado, o converti para o formato .p7s com o openssl, com o comando:

```
> openssl crl2pkcs7 -nocrl -certfile moodle-ufsc-br.pem -out moodle.p7s
```


E verifiquei com o certmgr do windows:



Tenho a lista de certificação do certificado, e posso verificar se meu código está correto comparando com a corrente acima.

Criei as funções recomendadas no *template* fornecido do projeto, individualmente por cada arquivo. Comecei com o arquivo “CertChainFromAiA”, depois fui para “CertStoreCreator” e por fim “CertPathCreator”. Depois utilizei o main para chamar as funções criadas. No código incluí onde peguei alguns snippets de código e expliquei como funcionam algumas partes. Na minha solução o conjunto de certificados é baixado duas vezes, isso se deve pois mantive os argumentos iguais ao template fornecido. Não entendi também o argumento TrustChain ser um Set, pois somente a raiz é TrustAnchor (pelo meu entendimento [deste link](#)¹). Então meu código cria a CertStore invocando o método “downloadCertificateChain” e, portanto, não utiliza as TrustChains fornecidas, mas mantive o argumento para conservar o template fornecido. De todo modo, o código funciona, retornando um CertPath com todos os certificados “acima” do certificado analisado.

This method returns a List of zero or more `java.security.cert.Certificate` objects. The returned List and the Certificates contained within it are immutable, in order to protect the contents of the `CertPath` object. The ordering of the certificates returned depends on the type. By convention, the certificates in a `CertPath` object of type `X.509` are ordered starting with the target certificate and ending with a certificate issued by the trust anchor. That is, the issuer of one certificate is the subject of the following one. The certificate representing the `TrustAnchor` should not be included in the certification path. Unvalidated `X.509` `CertPaths` may not follow this convention. PKIX `CertPathValidators` will detect any departure from these conventions that cause the certification path to be invalid and throw a `CertPathValidatorException`.

Processo

O processo foi longo e com várias mudanças ao longo da criação do código. Aqui estão algumas anotações que fiz ao longo do processo. Não é necessário ler, mas como no documento adicional dizia que o mais importante é o que o candidato fez quando surgiram dúvidas, resolvi colocar meus pensamentos. Estão somente erros de pensamento, não considerando bugs e semelhantes.

Inicialmente fiquei perdido na criação do código, não sabia por onde começar, então descobri o template fornecido no desafio e decidi segui-lo. Comecei a preencher as funções, e ter o problema dividido em várias partes foi muito útil. Não mais eu tinha um problema grande, mas vários problemas pequenos. As funções individuais foram relativamente simples de fazer (embora tenha enfrentado múltiplos bugs durante o desenvolvimento), mas tive alguns erros de projeto.

Problema inicial:

Inicialmente achei que precisaria fazer um get no certificado que emite, e fazer get no emite deste, até chegar no CA, então o código inicial era bem complexo, e não muito escalável. O código pegava o link <https://pbad.labsec.ufsc.br/challenge/artifacts/rsa/> do certificado, adicionava “cert_” a ele, fazia um REGEX para pegar o nome do próximo certificado, adicionava ao link, e por fim adicionava um .pem. Achei a solução estranha e somente aplicável a este caso, não podendo escalar muito. Então pesquisei mais sobre AiA, especialmente o que é um arquivo .p7s, e entendi o problema. Em seguida modifiquei o código para pegar os certificados conforme o arquivo .p7s do certificado. Aqui estão minhas anotações de quando descobri:

Consegui fazer com que a requisição com get fosse feita, mas descobri que precisaria incluir o nome do certificado gerador na requisição. Isso funcionou com o primeiro certificado (CHOP_SUEY), mas pegar o AiA do segundo certificado (FAINT) falhou, então precisei modificar a lógica do código. Para fazer isso, baixei todos os certificados da corrente e analisei todos eles, verificando onde estava a link para o próximo, de forma que eu consiga tratar todos os certificados. Baixei os certificados por [este link](#) (que eu havia encontrado nas tentativas anteriores) e verifiquei seus conteúdos por meio [desta ferramenta](#).

Foi então que entendi que o problema estava no meu entendimento da lógica do AiA. Eu imaginava que seria necessário buscar o certificado que gerou o certificado atual e continuar isso até ter todos os certificados da cadeia, assim

“reconstruindo” a cadeia de certificação. Depois de ver isso entendi que seria pegar o AiA do certificado.

O segundo grande problema foi que achei que o resultado da função “downloadCertificateChain” fosse já o resultado da questão. Achei que devido ao nome, e o resultado ser uma lista ordenada dos certificados, a lista de certificados era a cadeia de certificação. Um dia depois, quando fui remover partes não utilizadas do código, percebi que havia importações não utilizadas (nominalmente CertPathCreator e CertStoreCreator) que já estavam por padrão. Observando esses arquivos, notei que uma das funções retornava um tipo CertPath (o mesmo que foi pedido no main), então modifiquei o código, criando e utilizando as funções desses arquivos.

Percebi que somente um dos certificados (CHOP_SUEY) estava no CertPath final, então procurei o erro. Percebi que era porque o certificado final estava incluso no conjunto de TrustAnchors. Para manter os argumentos originais fornecidos, tive que baixar os certificados duas vezes, isso é ineficiente, o método mais eficiente (na minha opinião) então seria enviar a lista de certificados do main para a criação da lista de certificados, ou baixar a lista de certificados dentro da função que cria a CertStore. Dessa forma, os certificados somente seriam baixados uma vez. Naturalmente, isso vem da minha solução, a solução de vocês é diferente e deve ser mais eficiente que a minha.

Links utilizados:

Aqui estão alguns dos links utilizados, tentei deixar no código as origens de snippets de código que peguei, mas aqui estão outros:

<https://docs.oracle.com/javase/8/docs/technotes/guides/lang/resources.html>

<https://stackoverflow.com/questions/63020771/how-to-use-bouncycastle-to-get-the-certification-path-between-a-root-ca-and-an-e>

<https://www.baeldung.com/java-read-pem-file-keys>

<https://www.baeldung.com/java-bouncy-castle>

<https://www.youtube.com/watch?v=1I9lQ42SnLA>

<https://docs.oracle.com/javase/8/docs/technotes/guides/security/certpath/CertPathPrologGuide.html>

<https://stackoverflow.com/questions/26097214/validate-certificate-chain-with-java-bouncing-castle>

<https://stackoverflow.com/questions/15974516/certificate-path-discovery-in-java>

https://www.youtube.com/watch?v=1925zmDP_BY

<https://stackoverflow.com/questions/44846091/how-to-parse-authorityinformation-from-x509certificate-object>

<https://docs.oracle.com/javase/8/docs/technotes/guides/security/certpath/CertPathProviderGuide.html>

<https://stackoverflow.com/questions/13671487/generate-x509certificate-certpath-in-java>

<https://stackoverflow.com/questions/16058889/java-bouncy-castle-ocsp-url>

<https://superuser.com/questions/126121/how-to-create-my-own-certificate-chain>

<https://stackoverflow.com/questions/7360602/openssl-and-error-in-reading-openssl-conf-file>

<https://stackoverflow.com/questions/65028654/how-to-retrieve-the-cn-name-of-an-x509-certificate-with-bouncycastle>

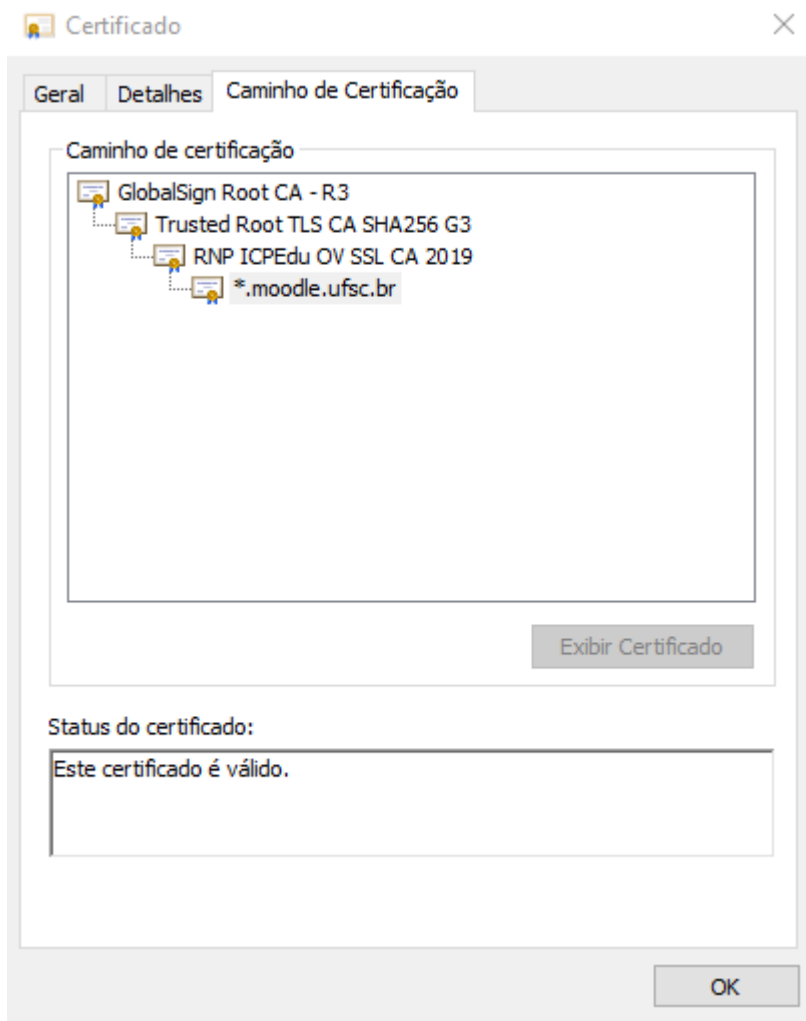
<https://akamath32.medium.com/certificate-authority-information-access-aia-7bc56f7257fc>

Etapa 2

Converti o certificado para .p7s por meio do comando e o abri com o certmgr:

```
> openssl crl2 pkcs7 -nocrl -certfile moodle-ufsc-br.pem -out moodle.p7s
```

Resultado:



Após isso, eu abri o certificado por meio do comando:




```
> openssl x509 -in moodle-ufsc-br.pem -noout -text
```

Verifiquei o link do próximo certificado:

```
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-2>openssl x509 -in moodle-ufsc-br.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6d:5e:0e:03:64:0f:c1:ef:50:c8:35:fa
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = BR, O = Rede Nacional de Ensino e Pesquisa - RNP, CN = RNP ICPEdu OV SSL CA 2019
        Validity
            Not Before: Jun 20 18:56:02 2024 GMT
            Not After : Jul 22 18:56:01 2025 GMT
        Subject: C = BR, ST = Santa Catarina, L = Florianópolis, O = UNIVERSIDADE FEDERAL DE SANTA CATARINA, CN = *.moodle.ufsc.br
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:cb:93:46:e4:3a:f2:31:f0:ee:b9:db:34:9f:4a:
                    f2:4b:97:fa:8e:b4:f0:14:cf:1d:45:f1:d5:98:84:
                    2d:94:31:6f:64:38:c8:0b:4a:a8:3d:f4:80:a4:
                    a5:e1:7a:67:f3:23:a1:c9:de:b5:d9:e4:90:44:d0:
                    f8:46:a6:dd:c0:2b:b1:9e:d9:95:a4:b2:80:b6:51:
                    59:25:fa:21:c7:b8:ed:89:d6:ce:ad:ec:ed:cb:49:
                    42:5f:a3:70:0f:b5:2a:df:41:6f:52:f9:33:fd:70:
                    13:13:e2:b4:a2:0d:e7:65:5b:0e:f6:68:16:8a:e6:
                    8f:c8:0b:23:e0:2b:01:eb:d2:b0:41:15:42:b4:07:
                    18:02:5a:ab:a2:79:35:c6:42:c2:3b:09:3e:c5:7f:
                    42:c7:60:94:7e:c7:41:49:f2:d7:6a:c2:db:d6:15:
                    99:f3:f5:2e:53:ca:5e:6d:f1:5c:05:96:3b:c7:b6:
                    b7:ac:90:e7:fd:5b:d1:4e:92:ba:07:83:4d:47:84:
                    bd:29:15:40:54:7d:27:fc:93:c0:bb:ed:5a:ec:a3:
                    0d:ca:bb:52:c5:aa:a9:89:97:c8:7e:5a:45:61:bc:
                    f3:c7:29:99:ea:b1:76:cf:86:ec:4d:ed:ff:f8:66:
                    97:0f:e9:71:55:a8:4d:45:05:d1:4b:1a:44:03:76:
                    a3:a9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Basic Constraints: critical
                CA:FALSE
            Authority Information Access:
                CA Issuers - URI: http://secure.globalsign.com/cacert/rnpicpeduovsslca2019.crt
                OCSP - URI: http://ocsp.globalsign.com/rnpicpeduovsslca2019
            X509v3 Certificate Policies:
```

Baixei o certificado com esse link, e fiz o mesmo com os próximos certificados.

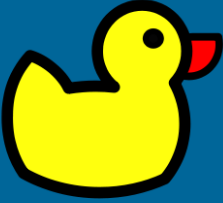
Assim peguei o nome correto dos certificados.

Nome	Data de modificação	Tipo	Tamanho
 rnpicpeduovsslca2019	26/10/2024 19:18	Certificado de Seg...	2 KB
 trustrootlssha2g3	26/10/2024 19:21	Certificado de Seg...	2 KB
 root-r3	26/10/2024 19:25	Certificado de Seg...	1 KB

Etapa 3

Ao pesquisar sobre certificados Let's Encrypt, percebi que eles são para certificação de domínio, infelizmente essa é uma área de pouco domínio meu, então foi uma etapa difícil.

Baixei o Caddy pela linha de comando Linux pelo WSL, mas tive problemas ao tentar gerar o certificado, então passei a tentar o duckDNS, mas ele não aceitou o domínio "labsec.local", devido a incluir um ponto no nome:



Duck DNS

account pedrohenriqueprazeres@hotmail.com
type free
token a0debd42-a5df-488b-ab2a-d136b07cb37b
token generated 20 minutes ago
created date 30 Oct 2024, 00:03:59

error: invalid domain entered, valid characters are : A-Z, 0-9, -

domains 0/5

http://

sub domain

.duckdns.org

add domain

domain	current ip	ipv6	changed
--------	------------	------	---------

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

O prazo de entrega já estava se aproximando, então decidi fazer um certificado auto assinado. Utilizei o seguinte comando openssl:
> openssl genpkey -algorithm RSA -out privatekey.pem -aes256

E fiz o certificado com:

> openssl req -new -x509 -key privatekey.pem -out certificado.pem -days 365

Questão 3

Status da Assinatura:

Tentei seguir os links disponíveis nos documentos (veja timestamp para mais informações).

Tentei múltiplas formas de verificar a assinatura com o certificado. Mas no fim não fui capaz.

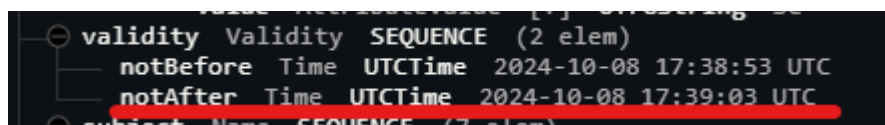
Tive pouco tempo para essa etapa específica, pois levei muito tempo nas outras.

Como não pude provar invalidade, coloquei todas as assinaturas como válidas.

Status do Certificado:

Separei os certificados em arquivos no formato .pem, e utilizei uma [ferramenta online](#) para verificar seus conteúdos:

cms_with_ts está fora do prazo de validade. Tendo de data limite 08/10/2024:



Xades_with_ts parece válido, não encontrei nenhum campo estranho, e possui informações sobre emissor, lista de revogação, OCSP está disponível e está dentro da validade.

Xades_with_ts_2 também está com informações válidas, como o certificado acima

Verifiquei as validades dos certificados.

Carimbo de tempo:

O arquivo cms_with_ts eu tentei inicialmente o comando do **WSL**:

```
> openssl pkcs7 -in cms_with_ts.p7s -inform DER -print | grep -A 100 'id-smime-aa-timeStampToken' > timestamp_token.der
```

No arquivo resultante eu usei o seguinte comando:

```
> openssl asn1parse -in timestamp_token.der -inform DER
```

Contudo, isso gerou um erro:


```

0:d=0 hl=2 l= 32 cons: EOC
Error in encoding
40076D02027F0000:error:0680009B:asn1 encoding routines:ASN1_get_object:too long:../crypto/asn1/asn1_lib.c:95:

```

Então tive de selecionar os bytes manualmente, primeiro usei o comando:

> openssl asn1parse -in cms_with_ts.p7s -inform DER

```

258751:d=5 hl=4 l=3371 cons: cont [ 1 ]
258755:d=6 hl=4 l=3367 cons: SEQUENCE
258759:d=7 hl=2 l= 11 prim: OBJECT          :id-smime-aa-timeStampToken
258772:d=7 hl=4 l=3350 cons: SET
258776:d=8 hl=4 l=3346 cons: SEQUENCE

```

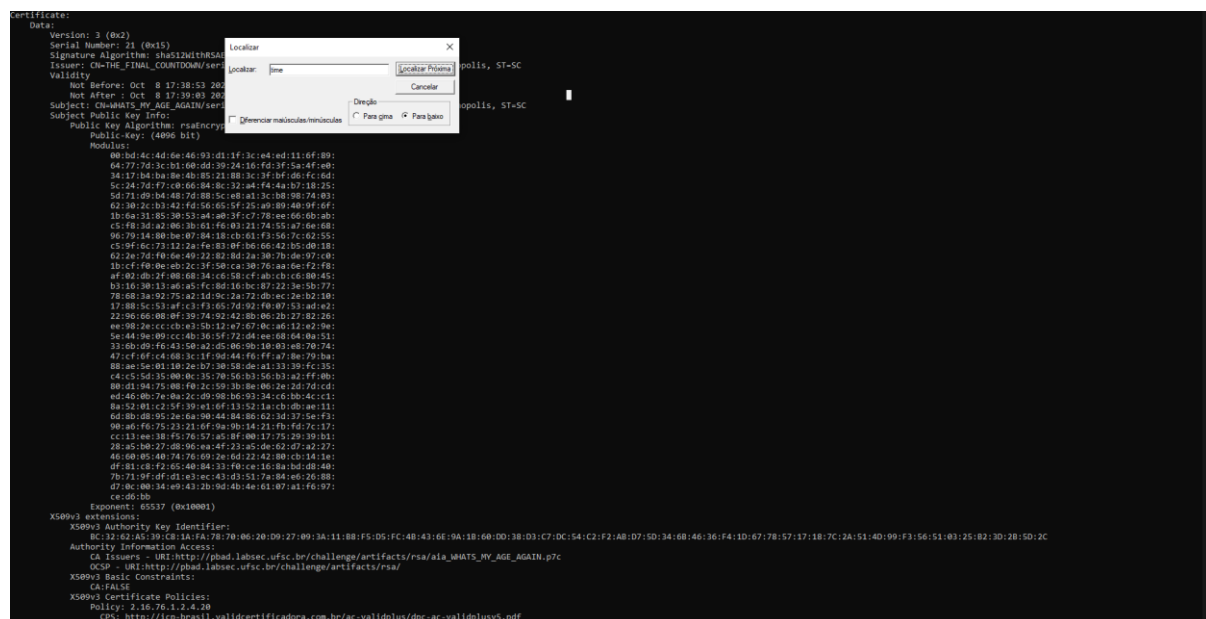
O que gera o comando:

> dd if=cms_with_ts.p7s of=timestamp_token.der bs=1 skip=258751 count=13

Após verificar o conteúdo, percebi que esses bytes indicam somente uma string “:id-smime-aa-timeStampToken”, o que faz sentido, com somente 13 bytes o conteúdo deve ser simples.

Então tentei abrir o certificado com o comando:

> openssl pkcs7 -in cms_with_ts.p7s -inform DER -print_certs -text



```

Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 21 (0x15)
  Signature Algorithm: sha512withRSA
  Issuer: CN=THE_FINAL_COUNTDOWN/serial
  Validity
    Not Before: Oct  8 17:38:53 2020
    Not After : Oct  8 17:39:03 2020
  Subject: CN=WHAT5_MY_AGE_AGAIN/serial
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (4096 bit)
    Modulus:
      00:b6:4d:0e:46:93:d1:1f:3c:e6:cd:11:0f:89:
      04:77:7d:3c:01:00:dd:99:24:16:f6:3f:5a:4f:08:
      34:17:b4:ba:0e:4b:85:21:88:3c:3f:bf:06:fc:6d:
      5c:24:7d:f7:c0:66:04:0c:32:04:f4:0a:07:10:25:
      5d:71:d9:b4:4b:7d:88:5c:e8:e1:3c:b0:98:74:03:
      62:30:2c:b3:42:fd:56:05:5f:25:a9:09:00:9f:0f:
      1b:6a:31:05:30:53:a4:00:5f:c7:70:ce:06:0b:0b:
      c5:f8:3d:a2:06:3b:61:f6:03:21:74:55:a7:0e:08:
      96:79:14:08:0a:07:04:18:cb:01:f3:50:7c:62:55:
      c5:0f:6c:73:12:2a:fa:03:0f:06:06:02:05:0b:18:
      62:7e:7d:f0:0e:49:22:82:8d:2a:30:7b:0e:97:c0:
      1bc:f9:0e:0b:2c:3f:50:ca:30:7b:0e:0e:f2:f8:
      af:02:db:2f:00:08:34:c6:58:cfa:b0:c6:00:45:
      b3:16:30:13:a6:a5:fc:0d:16:bc:07:22:3e:5b:77:
      7a:0b:3a:92:75:a2:1d:0c:2a:f2:0b:ce:2e:b2:10:
      17:08:5c:53:af:c3:f3:65:7d:92:f0:07:53:ad:e2:
      22:96:06:08:0f:39:74:92:42:0b:06:2b:27:82:26:
      ee:98:2a:cc:0a:5b:12:07:07:0e:a6:12:a2:9a:
      5e:44:9e:09:cc:db:30:5f:72:04:ee:08:04:0a:51:
      33:0b:0f:6c:43:50:a2:05:00:0a:00:03:0e:70:74:
      47:cf:0f:c4:0b:3c:1f:0d:44:f0:ff:a7:0e:79:b4:
      88:ae:5e:01:10:2e:b7:30:58:de:a1:33:39:fc:35:
      c4:c5:5d:35:00:0c:35:70:56:03:06:33:a2:ff:0a:
      00:d1:94:75:00:f0:2c:59:3b:0e:0e:2e:2d:7d:cd:
      ed:40:0b:7e:0a:2c:09:08:06:93:34:c6:b0:4c:c1:
      8a:51:01:c2:5f:39:e1:0f:13:52:1a:cb:00:ae:13:
      6d:8b:0b:95:2e:6a:90:44:84:86:02:3d:27:5e:f3:
      96:ed:f6:75:23:21:0f:0a:0b:14:21:f0:fd:7c:17:
      cc:13:ee:18:f5:76:57:a5:bf:00:17:75:29:39:b1:
      28:a5:b0:27:08:96:ea:4f:23:a5:de:02:c7:a2:27:
      4e:0b:05:40:74:76:09:2a:6d:22:c2:0b:cb:16:1a:
      df:81:c8:f2:65:40:84:33:f0:ce:16:0a:b0:db:00:
      7b:71:9f:df:d1:e3:ec:43:d3:51:7a:04:a6:26:80:
      d7:0c:00:34:e9:43:2b:9d:4b:de:c3:07:a6:f6:97:
      ce:05:bb
    Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Authority Key Identifier:
    EC:12:a5:39:0a:07:00:06:20:09:27:09:3a:11:88:f5:d5:fc:4b:43:6e:0a:18:60:0d:38:03:c7:dc:54:c2:f2:ab:07:5d:34:68:46:36:f4:10:67:78:57:17:18:7c:2a:51:4d:99:f3:56:51:03:25:b2:30:28:5d:2c
  Authority Information Access:
    CA Issuers - URI:http://pbad.labsec.ufsc.br/challenge/artifacts/rsa/ai4_WHATS_MY_AGE_AGAIN.p7c
    OCSP - URI:http://pbad.labsec.ufsc.br/challenge/artifacts/rsa/
X509v3 Basic Constraints:
  CA:FALSE
X509v3 Certificate Policies:
  Policy: 2.16.76.1.2.4.20
  CPS: http://ica-brasil.validaertificadora.com.br/ac-validplus/dpc-ac-validplusv5.pdf

```

Percebi que não consegui encontrar o timestamp no arquivo por meio desse comando, então assumi que o timestamp não estava presente corretamente. Por isso coloquei esse timestamp como INVALIDO.

Para os arquivos XML:

Peguei o conteúdo dos timestamps, converti de base64 para binário, e coloquei em um arquivo .cms. Mas ao tentar fazer parsing das informações por meio do comando abaixo, obtive um erro:

> openssl cms -inform DER -in xades_encaps_ts.cms -verify -noout -text -inform DER

```
Error reading SMIME Content Info
2C200000:error:06800098:asn1 encoding routines:ASN1_get_object:too long:crypto\asn1\asn1_lib.c:95:
2C200000:error:06800066:asn1 encoding routines:asn1_check_tlen:bad object header:crypto\asn1\tasn_dec.c:1184:
2C200000:error:0688010A:asn1 encoding routines:asn1_d2i_ex_primitive:nested asn1 error:crypto\asn1\tasn_dec.c:752:
2C200000:error:0688010A:asn1 encoding routines:asn1_template_noexp_d2i:nested asn1 error:crypto\asn1\tasn_dec.c:685:Field=contentType, Type=CMS_ContentInfo
```

Em seguida tentei os passos [deste link](#).

Mas não foi possível decodificar o base64:

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

☐ Decode each line separately (useful for when you have multiple entries).

☒ Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE > Decodes your data into the area below.

```
00 *H
00000001
000 *He00000000 *H
000s00000000 *0/000 *He000000 !04b050h0e00d0Lt80!0Wu860ah00wj0 0Xj0020241008173902
Z000000000000000000
0 *H
00
000z100000U000I_RAN_SO_FAR_AWAY1
0000U00004100 00U00000BR1
0000U0
0UFSC100
00U0000LabSEC100000U00
```

Em seguida, tentei seguir os links disponíveis nos arquivos XML:

```
</infAdic>
</infNFe>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="id216582138300816967960018639351075510311">
  <ds:SignedInfo>
```

Assumi que o campo Id deve concatenar com o início do link, mas não sabia de qual forma, então tentei várias combinações:

<http://www.w3.org/2000/09/xmldsig#>

Id=id216582138300816967960018639351075510311

<http://www.w3.org/2000/09/xmldsig#id21658213830081696796001863935107551031>

1 > Não funcionou

<https://www.w3.org/2000/09/xmldsig/id21658213830081696796001863935107551031>

1 > Não funcionou

<https://www.w3.org/2000/09/xmldsigid21658213830081696796001863935107551031>

1 > Não funcionou

<https://www.w3.org/2000/09/xmldsigId=id216582138300816967960018639351075510311> > Não funcionou

<https://www.w3.org/2000/09/xmldsig#Id=id216582138300816967960018639351075510311> > Não funcionou

<https://www.w3.org/2000/09/xmldsig/Id=id216582138300816967960018639351075510311> > Não funcionou

<http://www.w3.org/2000/09/xmldsig=id216582138300816967960018639351075510311> > Não funcionou

<http://www.w3.org/2000/09/xmldsig#=id216582138300816967960018639351075510311> > Não funcionou

Tentei a mesma coisa com o link do timestamp:

<http://uri.etsi.org/01903/v1.3.2/id216582138300816967960018639351075510311> > Não funcionou

<https://uri.etsi.org/01903/v1.3.2/id216582138300816967960018639351075510311> > Não funcionou

Passei a tentar verificar o conteúdo do timestamp com uma [ferramenta online](#). Tentei usar as tags BEGIN TIMESTAMP e END TIMESTAMP e, surpreendentemente, isso funcionou:

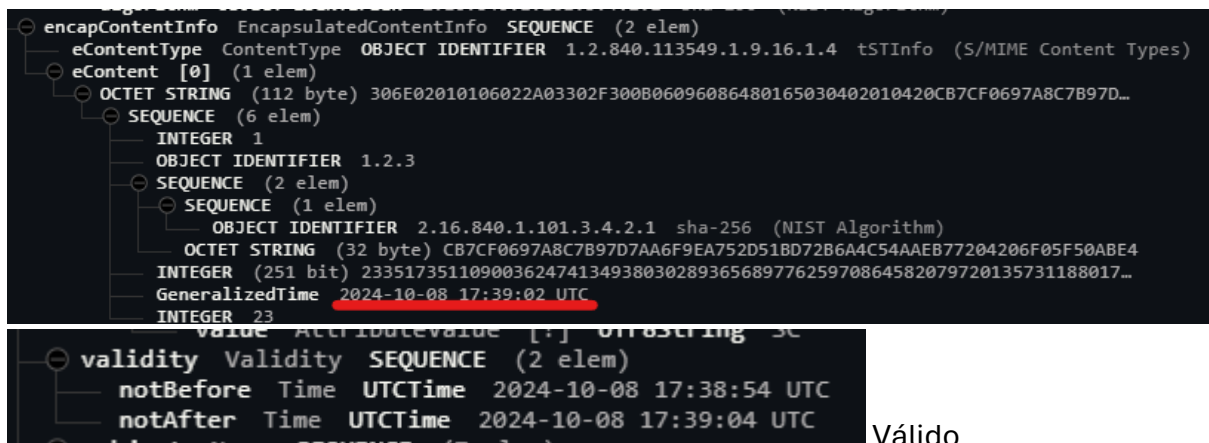
Xades_with_ts:

```
encapContentInfo EncapsulatedContentInfo SEQUENCE (2 elem)
├── eContentType ContentType OBJECT IDENTIFIER 1.2.840.113549.1.9.16.1.4 tSTInfo (S/MIME Content Types)
└── eContent [0] (1 elem)
    ├── OCTET STRING (113 byte) 306F02010106022A03302F300B06096086480165030402010420210FE13462C9C2EA1...
    └── SEQUENCE (6 elem)
        ├── INTEGER 1
        ├── OBJECT IDENTIFIER 1.2.3
        ├── SEQUENCE (2 elem)
        │   ├── SEQUENCE (1 elem)
        │   │   ├── OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256 (NIST Algorithm)
        │   │   └── OCTET STRING (32 byte) 210FE13462C9C2EA1894DBD48E188F68491965CD19FE10E0A564A61EF44C7438
        │   └── INTEGER (256 bit) 7902332091338318282933696295465856215438902634674869525652361241171609...
        └── GeneralizedTime 2024-10-08 17:39:02 UTC

value AttributeValue [?] UTF8String SC
validity Validity SEQUENCE (2 elem)
├── notBefore Time UTCTime 2024-10-08 17:38:54 UTC
└── notAfter Time UTCTime 2024-10-08 17:39:04 UTC
subject Name SEQUENCE (7 elem)
```

Válido

Xades_with_ts_2:



Infelizmente, devido ao prazo de entrega, não pude investigar mais que isso.

Questão 4

Extraí os certificados dos arquivos .p7s os abrindo com o comando:

```
> openssl pkcs7 -in [NOME DO ARQUIVO].p7s -inform DER -print_certs -text
```

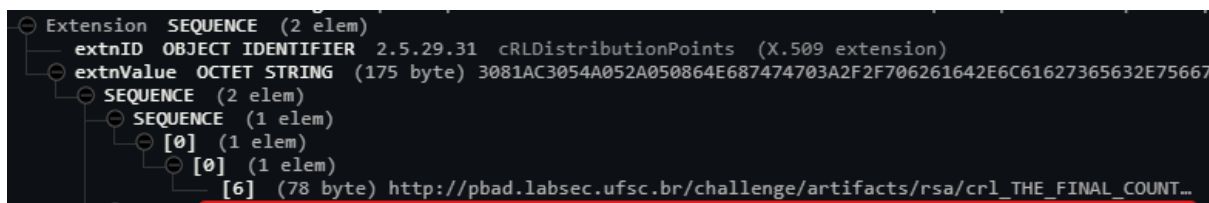
Os certificados dos arquivos .xml foram extraídos da tag de certificado

E analisei seu conteúdo com esta [ferramenta online](#). Dele peguei a lista de revogação do certificado gerador e comparei com o certificado original, usando o seguinte comando:

```
> openssl x509 -in [NOME DO ARQUIVO] -noout -serial
```

Por exemplo:

Link da lista de revogação do certificado cades_b_b (MANIAC):



Ao baixar a lista de revogação, temos:

Certificados revogados:

Número de série	Data de revogação
0d	terça-feira, 8 de outubr...
0e	terça-feira, 8 de outubr...
0f	terça-feira, 8 de outubr...

Resultado do comando do openssl:

Número de série	Data de revogação
0d	terça-feira, 8 de outubr...
0e	terça-feira, 8 de outubr...
0f	terça-feira, 8 de outubr...

D:\Pastas
ial
serial=0F
Revogado

Como todos tem como CA o THE_FINAL_COUNTDOWN, podemos comparar todos com esta lista:

Certificados revogados:

Número de série	Data de revogação
0d	terça-feira, 8 de outubr...
0e	terça-feira, 8 de outubr...
0f	terça-feira, 8 de outubr...

Seriais:

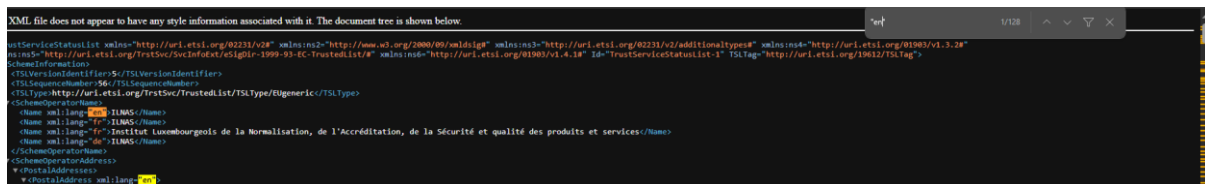
```
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-4>openssl x509 -in cades_b_b.pem -noout -serial serial=0F
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-4>openssl x509 -in cms.pem -noout -serial serial=0D
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-4>openssl x509 -in signature.pem -noout -serial serial=0E
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-4>openssl x509 -in xades_ad_rb.pem -noout -serial serial=0F
C:\Users\pedro\Documents\GitHub\Desafio-Labsec\resources\tarefa-4>openssl x509 -in xades_b_b.pem -noout -serial serial=0D
```

Somente xades_b_b é válido

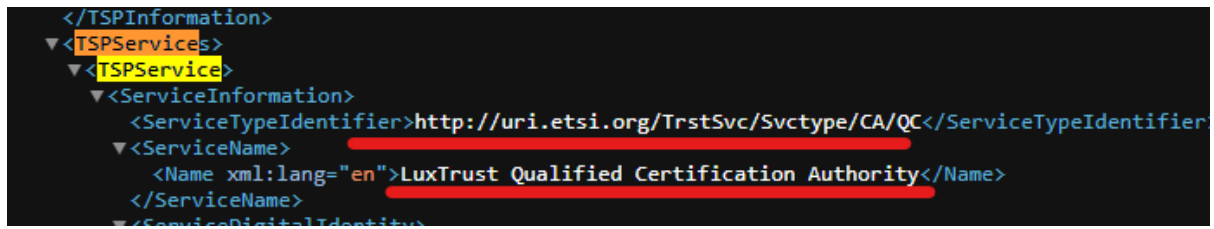
Questão 5

Etapas 1:

Filtrei o documento com “en” e comecei a ler os resultados:



Notei a existência de uma tag “TSPService” então filtrei por essa tag. Peguei o nome e o identificador:



```

</TSPInformation>
<TSPServices>
  <TSPService>
    <ServiceInformation>
      <ServiceTypeIdentifier>http://uri.etsi.org/TrstSvc/Svctype/CA/QC</ServiceTypeIdentifier>
      <ServiceName>
        <Name xml:lang="en">LuxTrust Qualified Certification Authority</Name>
      </ServiceName>
    </ServiceInformation>
  </TSPService>
</TSPServices>

```

Esse não foi um processo muito demorado, decidi fazer assim pois achei que seria mais rápido que escrever um código que pegue estas informações para mim. Contudo, para poder ser mais completo e escalável. Decidi também fazer um código em Java que pega e printa estas informações:

```

public static void main(String[] args) {
    //Feito com base em:
https://stackoverflow.com/questions/4076910/how-to-retrieve-element-value-of-xml-using-java
    try {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        Document document =
builder.parse("src/main/resources/LU.xml");
        document.getDocumentElement().normalize();

        NodeList serviceList =
document.getElementsByTagName("TSPService");

        for (int i = 0; i < serviceList.getLength(); i++) {
            Node serviceNode = serviceList.item(i);

            if (serviceNode.getNodeType() == Node.ELEMENT_NODE) {
                Element serviceElement = (Element) serviceNode;

                String serviceTypeIdentifier = serviceElement
.getElementsByTagName("ServiceTypeIdentifier")
                .item(0)
                .getTextContent();

                String serviceName = serviceElement
                .getElementsByTagName("ServiceName")
                .item(0)
                .getTextContent();
            }
        }
    }
}

```

```

        System.out.println(serviceTypeIdentifier + ": " +
serviceName);
    }
}

} catch (Exception e) {
    System.out.println("ERRO: " + e);
}
}

```

Resultado:

```

http://uri.etsi.org/TrstSvc/Svctype/CA/QC:
    LuxTrust Qualified Certification Authority

http://uri.etsi.org/TrstSvc/Svctype/TSA/TSS-QC:
    LuxTrust Time Stamping Authority

http://uri.etsi.org/TrstSvc/Svctype/CA/QC:
    LuxTrust Global Qualified Certification Authority

http://uri.etsi.org/TrstSvc/Svctype/TSA/TSS-QC:
    LuxTrust Global Time Stamping Authority

http://uri.etsi.org/TrstSvc/Svctype/CA/QC:
    LuxTrust Global Qualified Certification Authority 2

http://uri.etsi.org/TrstSvc/Svctype/CA/QC:
    LuxTrust Global Qualified Certification Authority 3

http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:
    LuxTrust Global Time Stamping Authority

http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:
    LuxTrust Qualified Time Stamping

http://uri.etsi.org/TrstSvc/Svctype/CA/QC:
    LuxTrust SSL Certification Authority 5

http://uri.etsi.org/TrstSvc/Svctype/QESValidation/Q:
    LuxTrust Seal and Signature Validation Service

http://uri.etsi.org/TrstSvc/Svctype/QESValidation/Q:
    LuxTrust Seal and Signature Validation Service

http://uri.etsi.org/TrstSvc/Svctype/EDS/Q:
    LuxTrust Qualified Electronic Registered Delivery Service

http://uri.etsi.org/TrstSvc/Svctype/QESValidation/Q:
    LuxTrust Seal and Signature Validation Service

```



```
http://uri.etsi.org/TrstSvc/Svctype/QESValidation/Q:  
LuxTrust Seal and Signature Validation Service  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
LuxTrust Qualified Time Stamping - LTQTSCA-RSA  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
LuxTrust Qualified Time Stamping - LTQTSCA-EC  
  
http://uri.etsi.org/TrstSvc/Svctype/CA/QC:  
SeMarket Qualified Certification Authority  
  
http://uri.etsi.org/TrstSvc/Svctype/CA/QC:  
BE-YS SIGNATURE AND AUTHENTICATION CA NC  
  
http://uri.etsi.org/TrstSvc/Svctype/CA/QC:  
BE-YS SIGNATURE AND AUTHENTICATION CA NC 2  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
BE INVEST TIMESTAMP UNIT 9  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
BE INVEST TIMESTAMP UNIT 10  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
BE INVEST TIMESTAMP UNIT Q 1  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
BE INVEST TIMESTAMP UNIT Q 2  
  
http://uri.etsi.org/TrstSvc/Svctype/CA/QC:  
IgniSign Global Qualified CA 2  
  
http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST:  
IgniSign TSA Qualified CA 2
```

Alguns desses resultados são iguais uns aos outros, fiquei em dúvida em como contar, se colocava o mesmo serviço duas vezes. Decidi que toda a combinação única de identificador – nome seria considerado um serviço diferente. Desta forma “<http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST> BE INVEST TIMESTAMP UNIT Q 1” e “<http://uri.etsi.org/TrstSvc/Svctype/TSA/QTST> BE INVEST TIMESTAMP UNIT Q 2” são serviços separados

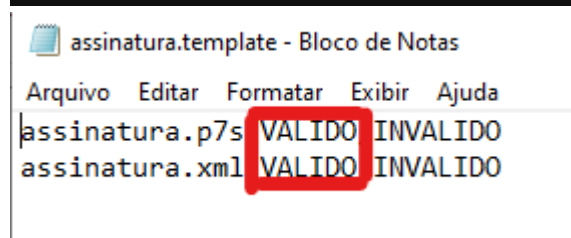
Etapa 2:

Inicialmente fiquei confuso com o enunciado da questão, o enunciado em si diz que as assinaturas são inválidas:

- Etapa 2: você deverá explicar de o motivo da invalidade da assinatura cms anexada, considerando tl.xml como a lista de confiança vigente.

Mas as instruções de resposta e o template dão a entender que as assinaturas podem ser válidas ou inválidas:

- Etapa 2: o status da assinatura, como VALIDO ou INVALIDO em um arquivo nomeado assinatura-lista.txt.



Como o número do segundo caso é maior, considerei a possibilidade de as assinaturas serem obrigatoriamente inválidas como um erro de digitação, e que as assinaturas podem ser válidas ou inválidas.

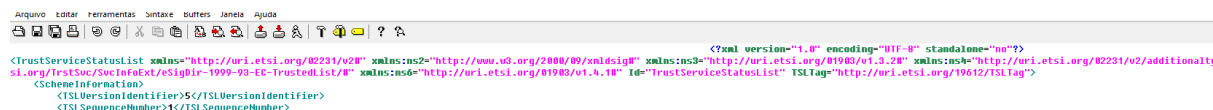
O arquivo “tl.xml” veio inicialmente danificado:

This page contains the following errors:

error on line 1 at column 126: XML declaration allowed only at the start of the document

Below is a rendering of the page up to the first error.

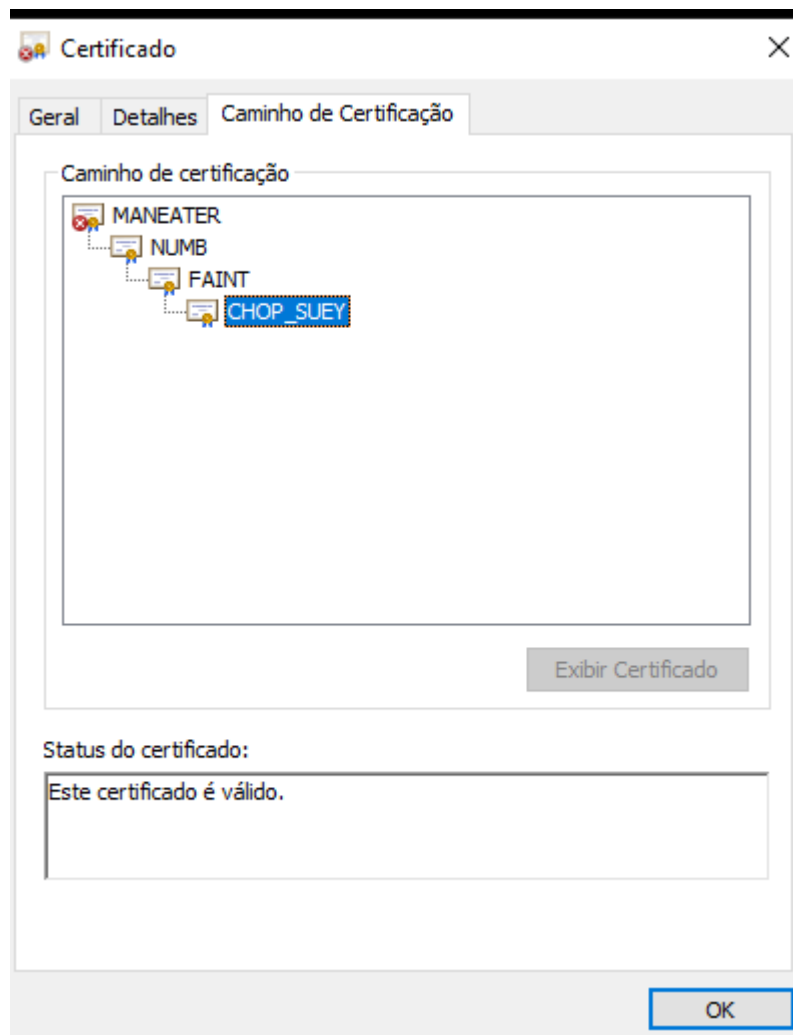
Assumi que isso é parte da questão e investiguei, o problema se dá pois o arquivo xml inicia com espaço em branco, e não com uma tag xml:



Removi esse espaço em branco e abri o arquivo novamente, o arquivo abriu com sucesso.

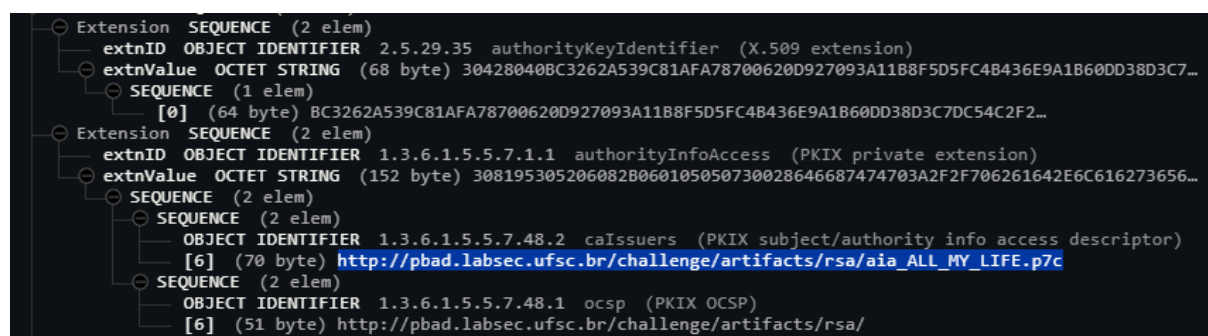
Analisando o arquivo, verifiquei haver três CAs, não somente o nome está listado, mas há também três certificados autoassinados, o que é uma característica de um CA. Dado que “tl.xml” é a lista de confiança vigente, todos os outros certificados cuja cadeia de confiança não leve a um dos três certificados “Pretty Fly”, “Hungry Like a Wolf” e “Maneater” não é válido.

O certificado .p7s foi verificado o abrindo com o certmgr do windows, aqui está o resultado:



Válido

O certificado XML foi verificado observando o conteúdo de seu certificado por meio de uma [ferramenta online](#), na qual busquei seu AiA com base no link:



E observei seu conteúdo por meio do comando do openssl:

```
> openssl pkcs7 -in aia_ALL_MY_LIFE.p7c -inform DER -print_certs -text
```

Observei que este certificado foi emitido pelo certificado “Pretty Fly”:

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 2 (0x2)
    Signature Algorithm: sha512WithRSAEncryption
    Issuer: CN=PRETTY_FLY/serialNumber=1, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
    Validity
      Not Before: Oct  8 17:38:39 2024 GMT
      Not After : Sep 23 17:38:39 2027 GMT
    Subject: CN=HIGHER/serialNumber=2, C=BR, O=UFSC, OU=LabSEC, L=Florianopolis, ST=SC
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (4096 bit)

```

Válido

Em seguida verifiquei a lista de revogação dos CAs:

Maneater não revogou nenhum certificado:

Lista de Certificados Revogados

Geral Lista de Certificados Revogados

Certificados revogados:

Número de série	Data de revogação
-----------------	-------------------

Entrada de revogação

Campo	Valor
-------	-------

Valor:

OK

Os certificados intermediários Numb e Faint, respectivamente:

Lista de Certificados Revogados

Geral Lista de Certificados Revogados

Certificados revogados:

Número de série	Data de revogação
-----------------	-------------------

Entrada de revogação

Campo	Valor
-------	-------

Valor:

OK

Lista de Certificados Revogados

Geral Lista de Certificados Revogados

Certificados revogados:

Número de série	Data de revogação
-----------------	-------------------

Entrada de revogação

Campo	Valor
-------	-------

Valor:

OK

Válido

Verificando CA do arquivo XML:

Lista de Certificados Revogados

Gerar Lista de Certificados Revogados

Certificados revogados:

Número de série	Data de revogação
-----------------	-------------------

Entrada de revogação

Campo	Valor
-------	-------

Valor:

OK

Válido

Conclusões:

Não foi possível fazer todas as questões, contudo, espero ter deixado claro por meio deste relatório que me esforcei no processo de resolução e que esse esforço resultou em muito aprendizado.

Estou ainda cursando a sexta fase e por isso ainda não tive a matéria de Segurança de Dados, que muito me interessa e só será semestre que vem. Tenho certeza

No início, eu possuía pouco conhecimento de certificações e assinaturas digitais, porém este desafio me trouxe novas ferramentas para meu entendimento, desta forma, agradeço humildemente pela oportunidade. Gostaria muito de ser escolhido, pois seria um grande passo para a carreira profissional que desejo na área de segurança digital.