Test: validating a file with a renewed certificate

The purpose of this page is to present the results of the tests performed to find if a renewed certificate (a new certificate issued using the same private key as its ancestor) would still validate a signature made by the old certificate. This test was requested in the SMP/SML meeting of September 13th. As such, 2 different tests were performed: based on the *openssl* tool and on Java code take from the Signature Manager component of the OpenNCP.

1. Command-line validation

For this test, we did:

1. Generate 3 different certificates out of the same private key;
2. Sign a file with the private key;
3. Verify the signature with the private key and output the recovered data;
4. Export the 3 public keys;
5. Verify the signature against the public keys and output the recovered data.

Command-line validation

```
$ openssl genrsa -out key.pem
Generating RSA private key, 2048 bit long modulus
........+++
.....................................................................
..+++
e is 65537 (0x10001)


$ openssl req -new -x509 -key key.pem -out cert.pem
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:


$ openssl x509 -in cert.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 16814434890736589193 (0xe958df724949d989)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd
        Validity
            Not Before: Sep 15 15:37:54 2016 GMT
            Not After : Oct 15 15:37:54 2016 GMT
```

```
        Subject: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b4:00:06:50:05:f8:bf:3a:f2:53:13:df:89:46:
                    7d:cf:bb:99:28:48:23:5c:22:d7:2d:25:ff:10:0c:
                    f4:44:6d:91:c7:50:bb:b7:40:02:1f:72:9b:c4:66:
                    a0:80:8d:e1:10:ef:26:a3:38:37:fb:70:6a:7a:95:
                    33:25:44:97:cd:c6:df:f4:e6:f0:d6:f8:08:66:b5:
                    61:6c:e3:6e:48:b4:6e:f7:14:f5:51:ab:7c:9d:e8:
                    83:3b:8c:e5:58:dd:c9:e0:51:87:da:1e:ff:fe:a1:
                    3c:68:37:e4:31:36:5a:84:f9:2a:68:38:ea:e5:66:
                    d8:4c:65:6b:d7:9f:88:ff:57:c8:ab:75:a8:2e:a1:
                    7a:3c:7c:8e:9c:70:b5:2e:4c:95:63:b6:85:1c:30:
                    bf:69:c7:c2:ed:5b:b7:6e:ad:72:81:56:0b:e5:00:
                    24:c8:91:38:2b:5b:8b:c4:df:82:95:20:ff:9f:83:
                    0e:3b:fe:ca:b9:93:ef:1e:3f:cb:ec:ef:6f:0d:fa:
                    10:12:1e:2a:a6:b5:61:5a:d4:21:5c:b9:f2:e0:a8:
                    5b:2d:0e:ea:3c:31:52:69:22:39:50:61:79:33:02:
                    5a:56:92:8d:32:e5:d7:d3:80:2e:23:a9:e4:f7:4f:
                    6b:45:68:3f:c5:6b:1a:a5:d3:a7:d6:26:41:12:08:
                    95:b1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                8B:D5:8B:ED:5C:E8:EC:12:BC:17:3E:E7:23:8F:57:1F:5F:0E:5A:EC
            X509v3 Authority Key Identifier:

keyid:8B:D5:8B:ED:5C:E8:EC:12:BC:17:3E:E7:23:8F:57:1F:5F:0E:5A:EC

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         65:26:c7:5d:9f:c4:2d:8d:df:46:1d:52:bf:c7:7a:dc:56:11:
         c9:df:bc:0a:b7:b0:17:71:89:65:0a:a3:f8:f4:dc:3e:04:79:
         b2:1a:0c:85:36:f7:d0:b5:42:10:48:f3:d6:2e:a2:54:04:60:
         e3:13:1a:aa:b9:01:ad:40:57:97:d2:10:6e:2b:64:28:0e:eb:
         3b:97:e8:51:b1:4c:55:82:b2:ff:f7:9e:e1:5f:80:46:11:b6:
         b9:00:f1:6d:d2:fa:00:fb:96:e8:4f:12:39:bd:be:a7:b0:05:
         ce:61:46:a7:d4:07:a4:4d:32:a4:00:74:38:ca:d7:c1:61:86:
         6b:54:ee:1d:18:19:03:ef:f1:20:73:3b:46:d5:88:04:41:9e:
         43:68:44:a8:a2:5e:09:6e:27:06:19:00:23:8c:a7:ce:93:bd:
         12:81:df:fb:84:ce:cb:45:b8:a1:93:ce:50:40:22:8d:b9:af:
         51:1a:be:de:cc:8a:d8:52:1b:c8:c7:69:4c:50:e2:6b:e3:d4:
         6e:b6:73:c1:64:4c:ab:e9:fd:ff:27:29:da:88:54:74:7a:e0:
         a6:67:95:75:4f:6e:ca:d7:16:4f:73:f5:d1:18:b2:49:a7:75:
         7c:71:16:9f:0f:97:47:4a:bb:18:86:ce:00:0d:25:95:f2:45:
         6a:0b:b2:c1


$ openssl req -new -x509 -key key.pem -out cert2.pem
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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:


$ diff cert.pem cert2.pem
2c2
< MIIDXTCCAkWgAwIBAgIJAOlY33JJSdmJMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
---
> MIIDXTCCAkWgAwIBAgIJAJA5iTbYYSI7MA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
4c4
< aWRnaXRzIFB0eSBMdGQwHhcNMTYwOTE1MTUzNzU0WhcNMTYxMDE1MTUzNzU0WjBF
---
> aWRnaXRzIFB0eSBMdGQwHhcNMTYwOTE1MTU0MDUxWhcNMTYxMDE1MTU0MDUxWjBF
14,20c14,20
< DAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAZSbHXZ/ELY3fRh1Sv8d6
< 3FYRyd+8CrewF3GJZQqj+PTcPgR5shoMhTb30LVCEEjz1i6iVARg4xMaqrkBrUBX
< l9IQbitkKA7rO5foUbFMVYKy//ee4V+ARhG2uQDxbdL6APuW6E8SOb2+p7AFzmFG
< p9QHpE0ypAB0OMrXwWGGa1TuHRgZA+/xIHM7RtWIBEGeQ2hEqKJeCW4nBhkAI4yn
< zpO9EoHf+4TOy0W4oZPOUEAijbmvURq+3syK2FIbyMdpTFDia+PUbrZzwWRMq+n9
< /ycp2ohUdHrgpmeVdU9uytcWT3P10RiySad1fHEWnw+XR0q7GIbOAA0llfJFaguy
< wQ==
---
> DAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAO664zrdjypgk2nTS34ck
> mYIuiI61Uf/mNzoM5r9ay+Kxdz9ZgaGwwlwWqpmSqmvEGCv2KkmZf5gWfexlTOlp
> wlSoONWXqB6uwn5RQfHZfm7qBI8GHzLK+kTtwx7SxK27Ccgy1sQ4ck8lozCg+dwB
> iS+qlLqQ1k/LWAGmnrTmvLzzFpIlBlF0K98yHgMllz6Q9oo5C9hHGoMpz9yM30sc
> djLhvzWc4MyGeAN1Y/F3EdkpkCZDUE35pMEcOX1iukXYVm/hhyCJWVGqu4YFowqy
> z3ahRaEDFtdTS+czVXQZmodWpP8wp8LA+/JlRPvMn7z09WXdNoDD1qw4jrHv5+z3
> 5A==


$ openssl req -new -x509 -key key.pem -out cert3.pem
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]:PT
State or Province Name (full name) [Some-State]:Porto
Locality Name (eg, city) []:Porto
```

```
Organization Name (eg, company) [Internet Widgits Pty Ltd]:openncp
Organizational Unit Name (eg, section) []:ehealth
Common Name (e.g. server FQDN or YOUR name) []:joao
Email Address []:joao


$ echo 'OpenNCP testing signatures' > file

$ openssl rsautl -sign -in file -inkey key.pem -out sig

$ openssl rsautl -verify -in sig -inkey key.pem
OpenNCP testing signatures


$ openssl x509 -in cert.pem -pubkey -noout
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtAAGUAX4vzryUxPfiUZ9
z7uZKEgjXCLXLSX/EAz0RG2Rx1C7t0ACH3KbxGaggI3hEO8mozg3+3BqepUzJUSX
zcbf9Obw1vgIZrVhbONuSLRu9xT1Uat8neiDO4zlWN3J4FGH2h7//qE8aDfkMTZa
hPkqaDjq5WbYTGVr15+I/1fIq3WoLqF6PHyOnHC1LkyVY7aFHDC/acfC7Vu3bq1y
gVYL5QAkyJE4K1uLxN+ClSD/n4MOO/7KuZPvHj/L7O9vDfoQEh4qprVhWtQhXLny
4KhbLQ7qPDFSaSI5UGF5MwJaVpKNMuXX04AuI6nk909rRWg/xWsapdOn1iZBEgiV
sQIDAQAB
-----END PUBLIC KEY-----

$ openssl x509 -in cert2.pem -pubkey -noout
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtAAGUAX4vzryUxPfiUZ9
z7uZKEgjXCLXLSX/EAz0RG2Rx1C7t0ACH3KbxGaggI3hEO8mozg3+3BqepUzJUSX
zcbf9Obw1vgIZrVhbONuSLRu9xT1Uat8neiDO4zlWN3J4FGH2h7//qE8aDfkMTZa
hPkqaDjq5WbYTGVr15+I/1fIq3WoLqF6PHyOnHC1LkyVY7aFHDC/acfC7Vu3bq1y
gVYL5QAkyJE4K1uLxN+ClSD/n4MOO/7KuZPvHj/L7O9vDfoQEh4qprVhWtQhXLny
4KhbLQ7qPDFSaSI5UGF5MwJaVpKNMuXX04AuI6nk909rRWg/xWsapdOn1iZBEgiV
sQIDAQAB
-----END PUBLIC KEY-----


$ openssl x509 -in cert3.pem -pubkey -noout
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtAAGUAX4vzryUxPfiUZ9
z7uZKEgjXCLXLSX/EAz0RG2Rx1C7t0ACH3KbxGaggI3hEO8mozg3+3BqepUzJUSX
zcbf9Obw1vgIZrVhbONuSLRu9xT1Uat8neiDO4zlWN3J4FGH2h7//qE8aDfkMTZa
hPkqaDjq5WbYTGVr15+I/1fIq3WoLqF6PHyOnHC1LkyVY7aFHDC/acfC7Vu3bq1y
gVYL5QAkyJE4K1uLxN+ClSD/n4MOO/7KuZPvHj/L7O9vDfoQEh4qprVhWtQhXLny
4KhbLQ7qPDFSaSI5UGF5MwJaVpKNMuXX04AuI6nk909rRWg/xWsapdOn1iZBEgiV
sQIDAQAB
-----END PUBLIC KEY-----


$ openssl x509 -in cert.pem -pubkey -noout > pubkey1.pem
$ openssl x509 -in cert2.pem -pubkey -noout > pubkey2.pem
$ openssl x509 -in cert3.pem -pubkey -noout > pubkey3.pem
```

```
$ openssl rsautl -in sig -verify -inkey pubkey1.pem -pubin
OpenNCP testing signatures

$ openssl rsautl -in sig -verify -inkey pubkey2.pem -pubin
OpenNCP testing signatures
```

```
$ openssl rsautl -in sig -verify -inkey pubkey3.pem -pubin
OpenNCP testing signatures
```

Conclusion: using the *openssl* command-line tool we are able to validate a signed file using any of the public keys matching the same private key used for signing. OK

2. Java validation (Security Manager)

We also performed tests using the Security Manager classes. For that purpose we did:

1. Use the scripts from the OpenNCP installation manual to generate a self-signed CA and issue two signature certificates from it, storing them (the whole certificate chain) in a keystore and a truststore;
2. Use the code already developed for the SMP add-on of the TSL-Editor to sign a SMP file using the certificate from the keystore;
3. Verify the trust on the signed SMP file using the certificate in the truststore.

cacert.sh

```
mkdir -p ROOT
country="pt"
openssl genrsa -des3 -out ROOT/$country-ca.key 4096
openssl req -new -x509 -days 3650 -key ROOT/$country-ca.key -out
ROOT/$country-ca.pem
```

The selfcert.sh script was tweaked to only generate 2 signing certificates and store them (as well as the root CA certificate) in 2 different keystores.

selfcert.sh

```
# ********************
# Variable Definition
# ********************

# Country
country="pt"
# Organization
organization="spms"
# Password of generated keystores
passwordKS="changeit"
# Password of root CA key (defined during the execution of root CA
certificate creation script - cacert.sh)
passwordCA="changeit"
# Password of truststore
passwordTS="changeit"
# Initial truststore to extend (if empty, a new one will be created under
/keystore with the CA certificate)
#initialTrustStore=/usr/java/jdk1.6.0_45/jre/lib/security/cacerts
initialTrustStore=

# Certificates alias
CAcertAlias=ppt.ca.epsos.$organization.$country
NcpSignatureAlias=ppt.ncp-signature.epsos.$organization.$country
```

```
# ******************* END OF CONFIGUATION SECTION ********************

echo "Creating Certificate Requests for: "$country
# WARNING: existing keystores will be deleted
rm -rf pem private test_requests keystore ROOT/$country-ca.srl
mkdir -p pem private test_requests keystore

# For each epSOS compliant certificate:
# 1) Generate a new certificate request (.csr) in text form and a new
private key
# 2) Generate a new certificate (.pem) from the previous certificate
request, valid for 365 days, signed with the generated CA certificate
# 3) Export the certificate and its private key to a PKCS#12 file
# 4) Import the entries from the PKCS#12 keystore into a JKS keystore
# 5) Change the default entry alias ('1') in the keystore to a more clear
one
# Note: The last 3 steps are not performed for VPN certificates.

# NCP Signature
# Generate private key + csr, then certificate and store in keystore
openssl req -new -sha512 -nodes -newkey rsa:2048 -keyout
private/$country-ncp-sign.key -out test_requests/$country-ncp-sign.csr
-config conf/config-NCPsignature-csr -text -utf8
openssl x509 -req -days 365 -in test_requests/$country-ncp-sign.csr
-CAcreateserial -CAserial ROOT/$country-ca.srl -CA ROOT/$country-ca.pem
-CAkey ROOT/$country-ca.key -out pem/$country-ncp-sig-self-sign.pem
-extfile conf/config-NCPsignature-crt -passin pass:$passwordCA -sha512

openssl pkcs12 -export -in pem/$country-ncp-sig-self-sign.pem -inkey
private/$country-ncp-sign.key -password pass:$passwordKS >
keystore/ncp-$country-sign.p12
keytool -importkeystore -srckeystore keystore/ncp-$country-sign.p12
-destkeystore keystore/$country-signature-keystore.jks -srcstoretype pkcs12
-deststorepass $passwordKS -srcstorepass $passwordKS
keytool -changealias -v -alias 1 -destalias $NcpSignatureAlias -keystore
keystore/$country-signature-keystore.jks -storepass $passwordKS


# Generate 2nd certificate from the previous csr
openssl x509 -req -days 365 -in test_requests/$country-ncp-sign.csr
-CAcreateserial -CAserial ROOT/$country-ca.srl -CA ROOT/$country-ca.pem
-CAkey ROOT/$country-ca.key -out pem/$country-ncp-sig-self-sign-2.pem
-extfile conf/config-NCPsignature-crt -passin pass:$passwordCA -sha512


# Importing a New Trusted Certificate
#
# Before adding the certificate to the keystore, keytool tries to verify it
by attempting to construct a chain of trust from that certificate to a
self-signed certificate (belonging to a root CA), using trusted
certificates that are already available in the keystore.
#
```

```
# If the -trustcacerts option has been specified, additional certificates
are considered for the chain of trust, namely the certificates in a file
named "cacerts".
#
# If keytool fails to establish a trust path from the certificate to be
imported up to a self-signed certificate (either from the keystore or the
"cacerts" file), the certificate information is printed out, and the user
is prompted to verify it

# Importing CA certificate into the keystores
keytool -importcert -alias $CAcertAlias -file ROOT/$country-ca.pem
-keystore keystore/$country-signature-keystore.jks -storepass $passwordKS

# Importing the certificates into the truststore
keytool -importcert -alias $CAcertAlias -file ROOT/$country-ca.pem
-keystore keystore/$country-truststore.jks -storepass $passwordTS
keytool -importcert -alias $NcpSignatureAlias -file
pem/$country-ncp-sig-self-sign-2.pem -keystore
keystore/$country-truststore.jks -storepass $passwordTS
```

```
# Clean .p12 files
rm -f keystore/*.p12
```

So, in the end we have the following:

Artifacts structure

```
/cert
 /keystore
  |-- pt-signature-keystore.jks
   * pt-ncp-sign.key + pt-ncp-sig-self-sign.pem
   * pt-ca.pem
  |-- pt-truststore.jks
   * pt-ncp-sig-self-sign-2.pem
   * pt-ca.pem
    /pem
  |-- pt-ncp-sig-self-sign.pem
  |-- pt-ncp-sig-self-sign-2.pem // will act as the renewed certificate
 /private
  |-- pt-ncp-sign.key
 /ROOT
  |-- pt-ca.key
  |-- pt-ca.pem
  |-- pt-ca.srl
 /test_requests
  |-- pt-ncp-sign.csr
```

These are the artifacts generated:

pt-ncp-sign.key

```
$ openssl rsa -in pt-ncp-sign.key -check
RSA key ok
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEA0JSJWVCyaLeq5AcI5j8cd82pI8hRTPIbfPLmQijWkMxHjOie
KVsEet1ATOFU9/QNcluILD+1+2gBAngsP6dqv1ctV9nhxrTzTU5ms9jtlu+ex5KR
FtK7Z4h9cuKKRVdvZouFZCxgxhESKUY8IRzUSkfOAe5Df3qQgvmxJeFreRk5n+5Y
N+0d4aBf+Vt9E9WzYrqQESojVZ5LkE1dMOFIEwnc7M7iRpPkZD7Cb41/dZ0PuX4D
jpQkaAASzlcNzuF73lr+1tPHb6OiyunUQYY8IHj06Nechnlrrqexb BYqhp1UhL4I
V2/xUx8uyq2BFiZhiebCxUDtK+YKyTCWD6l3gQIDAQABAoIBAQCqvEvJzqKb0WyT
h6ajQpn8/14qqjjSsYQKvZH6Kg+pK26OS4Pb+mMqZ53Q0MqKxT2UlJRdFDHU/4eE
C/a8jEB3K9c0CGgmDczzRPAQ28twN+lTGAioFAvPbgyyU/o3VKEd4atpGmKE/0hM
fO+Pc1fbDff14KFj4mIUtGtrxKAC2UZFpDpK1LYqSPsxlnWMveLrvvyqUtsTEa9f
VMGJd094+GXbXI0HDlfGYS1WphlzxORbrdUUY+eH0u0QjlLLFZdLY5qdNU+yWg92
Eu9/44UiKqRKrjCehrqSmZum9gi/b6O1apO/WA/ZNWUJGAGbay973HTqZYmV5uj9
6c9GrHiBAoGBAPSTTSQUt4Yqg0ng5PIKFkaLr063+fNBiRzqa3kQW0QbtYMvKxUF
luQHtK1mEqspDtSA5Pvh1w9kG0zjAAM57Y4OqP5nnfhM06AJ/tJ9+41ATmbdUsTY
FfRunGLHyT4XEc6hEVBkuh6Djgz6h8A67aZ/WLPhag19gN8jNqlN5Fj5AoGBANpS
y496g3UwkNHmjxNP0qYCt8zWg28RzE+xtPQfLuvsX6CuznPHwNTuv+l1sWHxYTE+
rSoPl1X0RwPrfuGcsPTg9zv07qT2Yts7JY/I7eN741NQbLV9sqqoWacwUpz76sHx
7C5QUbXUOmAiJRxM9Ek0Uehp8z0ewUG6ti9fP3zJAoGBAJ2LZV2gmreqGvg0DkZ7
iO3YfQzQJgo0ZQZjWueZSbENyFbAw5e/CfvJmvE3lz76K7tnbBl02PvpGEzmzNNH
NWcUundmd7PCiW/GAIfG74uFqPtKBk8Wgs35knNvDosYgTgBOQ34VIvjYlWp7Fg/
nWZrEdCNm6sk9SbHoCZfLoDBAoGBALNgj2uFR3kDvBkZi7hcP0Cyvv0Hytl5EBry
cTCaZlzkyvUkau/p0V+iAf/4+RL11ds2GeSTBs5Sc/6egJMxwJqSM9AJQAI9hZ+G
iygF+J6NbYtIauj7K87Xn0GkjFv7BzjuYhTzXQ8+HfBaXY9REu0KlbmVQ4qWOEae
940YfvsxAoGBANMDWtHTGfABDXiL9Ke2cWeqU3QWkmYVfqu7pSsaYDifC63rH+eE
fzIrZr/G6N53NwTmSoZ3UXJE5F0YDleKrF71LeHPd0VgZjXYg11Z8mAeR6ViENkN
sEl9uQbfUEglFZJ6S1gQSVTAYblOnpU+DO1rUDL+qPm9G+OQrjzm3gSs
-----END RSA PRIVATE KEY-----
```

pt-ncp-sig-self-sign.pem

```
$ openssl x509 -in pt-ncp-sig-self-sign.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 16408962163240547286 (0xe3b8582628c733d6)
    Signature Algorithm: sha512WithRSAEncryption
        Issuer: C=PT, ST=Porto, L=Porto, O=SPMS, OU=IOP,
CN=joao.cunha/emailAddress=joao.cunha@spms.min-saude.pt
        Validity
            Not Before: Sep 13 15:44:02 2016 GMT
            Not After : Sep 13 15:44:02 2017 GMT
        Subject: C=PT, O=SPMS, GN=Joao, SN=Cunha, CN=qaepsos.min-saude.pt
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
```

```
                    00:d0:94:89:59:50:b2:68:b7:aa:e4:07:08:e6:3f:
                    1c:77:cd:a9:23:c8:51:4c:f2:1b:7c:f2:e6:42:28:
                    d6:90:cc:47:8c:e8:9e:29:5b:04:7a:dd:40:4c:e1:
                    54:f7:f4:0d:72:5b:88:2c:3f:b5:fb:68:01:02:78:
                    2c:3f:a7:6a:bf:57:2d:57:d9:e1:c6:b4:f3:4d:4e:
                    66:b3:d8:ed:96:ef:9e:c7:92:91:16:d2:bb:67:88:
                    7d:72:e2:8a:45:57:6f:66:8b:85:64:2c:60:c6:11:
                    12:29:46:3c:21:1c:d4:4a:47:ce:01:ee:43:7f:7a:
                    90:82:f9:b1:25:e1:6b:79:19:39:9f:ee:58:37:ed:
                    1d:e1:a0:5f:f9:5b:7d:13:d5:b3:62:ba:90:11:2a:
                    23:55:9e:4b:90:4d:5d:30:e1:48:13:09:dc:ec:ce:
                    e2:46:93:e4:64:3e:c2:6f:8d:7f:75:9d:0f:b9:7e:
                    03:8e:94:24:68:00:12:ce:57:0d:ce:e1:7b:de:5a:
                    fe:d6:d3:c7:6f:a3:a2:ca:e9:d4:41:86:3c:20:78:
                    f4:e8:d7:9c:86:79:6b:ae:a7:b1:6c:16:2a:86:9d:
                    54:84:be:08:57:6f:f1:53:1f:2e:ca:ad:81:16:26:
                    61:89:e6:c2:c5:40:ed:2b:e6:0a:c9:30:96:0f:a9:
                    77:81
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:joao.cunha@spms.min-saude.pt
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation
            X509v3 Authority Key Identifier:

keyid:75:B3:E9:C2:7D:EB:50:A0:BF:0E:77:40:3E:7C:2C:DE:D7:CF:16:5C

            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                C4:22:21:A7:EB:87:D6:EB:15:BF:23:5D:48:7E:F5:24:E6:80:C5:29
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://your.cert.authority/CRL/openncp-crl.pem

    Signature Algorithm: sha512WithRSAEncryption
         2c:9c:98:cc:2a:12:a2:f6:fc:f3:0a:9d:cd:4c:cb:45:9a:6e:
         56:f3:53:2e:b0:8c:96:38:f6:fb:e6:31:17:57:ea:4f:f8:51:
         20:51:fd:8f:28:64:49:af:99:7c:44:e8:06:1e:65:19:70:02:
         a4:4c:3e:91:f8:cd:79:34:cc:ee:f8:45:bb:95:72:11:cc:e4:
         bd:7b:dc:5d:8f:f7:5a:85:80:b8:32:14:40:c3:3a:32:08:d7:
         bc:d0:78:41:18:60:be:fc:78:c2:ec:c2:7a:a4:e1:72:d8:ad:
         c8:4e:1b:f4:25:f9:c5:e1:30:96:9f:10:b1:14:e1:52:e2:26:
         71:56:72:76:f5:0f:eb:29:31:7a:1b:3b:9a:da:98:e7:59:c1:
         53:2f:6d:02:54:a9:15:ea:3e:fa:09:e1:96:21:92:da:16:fa:
         9d:a7:3d:06:a1:6c:3b:af:e7:91:72:9a:95:44:cc:52:41:3c:
         c5:2f:18:86:a7:2c:c6:ae:4c:a7:0c:31:9c:c4:95:47:21:dd:
         5c:81:a9:7c:83:ce:3d:92:a9:a8:ca:bc:05:15:a1:4d:fc:94:
         d6:ae:0f:07:bd:3e:e8:dc:9a:f6:96:d0:9f:fa:44:9e:d2:6e:
         45:36:d2:3d:58:fe:ea:d2:9f:a4:01:30:97:ec:1d:6d:8b:c1:
         7e:5c:b6:8f:ae:95:a7:17:50:78:dc:59:98:62:99:a2:49:f3:
```

```
a6:ad:8c:07:c3:c1:61:3a:cb:47:ff:10:d4:b3:8b:c7:99:eb:
dd:e1:cb:09:91:81:4e:ad:7b:b1:f8:94:ec:d4:cd:dd:de:c4:
c6:e2:14:dc:55:4d:f9:8c:1c:89:a8:bb:c6:41:1d:46:fb:c2:
25:54:35:28:5c:fd:0f:45:d6:0a:69:4e:95:30:77:c8:15:ef:
86:f7:65:ec:94:09:9e:a8:ec:40:d7:fc:e4:9d:64:d6:cf:c5:
19:68:f9:c3:6b:43:41:a9:f0:9a:16:10:9d:d1:ea:42:73:32:
10:0c:b3:f0:d8:c1:40:1f:2b:27:e8:71:e9:21:69:41:08:c4:
50:6e:e0:93:f9:ad:1d:37:6b:ad:2b:12:39:77:bf:99:87:5a:
eb:5b:9f:ce:a0:d9:bd:93:ac:6b:2c:97:f6:6c:2c:9c:72:20:
13:35:bc:34:89:ef:98:06:60:8c:33:8d:ef:d0:98:a6:92:19:
c9:eb:d5:12:68:e5:9e:83:1b:6e:53:7f:49:1b:59:45:89:c4:
cc:86:8f:c7:52:9e:c7:c7:92:88:93:53:89:9f:72:86:5c:6a:
```

```
          67:dc:9d:71:9e:1e:6d:2f:2e:f4:d2:de:3d:c6:df:57:e8:d6:
          e6:4c:20:9e:77:57:9a:f9
```

pt-ncp-sig-self-sign-2.pem

```
$ openssl x509 -in ../pem/pt-ncp-sig-self-sign-2.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 16408962163240547287 (0xe3b8582628c733d7)
    Signature Algorithm: sha512WithRSAEncryption
        Issuer: C=PT, ST=Porto, L=Porto, O=SPMS, OU=IOP,
CN=joao.cunha/emailAddress=joao.cunha@spms.min-saude.pt
        Validity
            Not Before: Sep 13 15:44:03 2016 GMT
            Not After : Sep 13 15:44:03 2017 GMT
        Subject: C=PT, O=SPMS, GN=Joao, SN=Cunha, CN=qaepsos.min-saude.pt
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d0:94:89:59:50:b2:68:b7:aa:e4:07:08:e6:3f:
                    1c:77:cd:a9:23:c8:51:4c:f2:1b:7c:f2:e6:42:28:
                    d6:90:cc:47:8c:e8:9e:29:5b:04:7a:dd:40:4c:e1:
                    54:f7:f4:0d:72:5b:88:2c:3f:b5:fb:68:01:02:78:
                    2c:3f:a7:6a:bf:57:2d:57:d9:e1:c6:b4:f3:4d:4e:
                    66:b3:d8:ed:96:ef:9e:c7:92:91:16:d2:bb:67:88:
                    7d:72:e2:8a:45:57:6f:66:8b:85:64:2c:60:c6:11:
                    12:29:46:3c:21:1c:d4:4a:47:ce:01:ee:43:7f:7a:
                    90:82:f9:b1:25:e1:6b:79:19:39:9f:ee:58:37:ed:
                    1d:e1:a0:5f:f9:5b:7d:13:d5:b3:62:ba:90:11:2a:
                    23:55:9e:4b:90:4d:5d:30:e1:48:13:09:dc:ec:ce:
                    e2:46:93:e4:64:3e:c2:6f:8d:7f:75:9d:0f:b9:7e:
                    03:8e:94:24:68:00:12:ce:57:0d:ce:e1:7b:de:5a:
                    fe:d6:d3:c7:6f:a3:a2:ca:e9:d4:41:86:3c:20:78:
                    f4:e8:d7:9c:86:79:6b:ae:a7:b1:6c:16:2a:86:9d:
                    54:84:be:08:57:6f:f1:53:1f:2e:ca:ad:81:16:26:
                    61:89:e6:c2:c5:40:ed:2b:e6:0a:c9:30:96:0f:a9:
                    77:81
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:joao.cunha@spms.min-saude.pt
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation
            X509v3 Authority Key Identifier:

keyid:75:B3:E9:C2:7D:EB:50:A0:BF:0E:77:40:3E:7C:2C:DE:D7:CF:16:5C

            X509v3 Basic Constraints: critical
                CA:FALSE
```

```
            X509v3 Subject Key Identifier:
                C4:22:21:A7:EB:87:D6:EB:15:BF:23:5D:48:7E:F5:24:E6:80:C5:29
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://your.cert.authority/CRL/openncp-crl.pem

    Signature Algorithm: sha512WithRSAEncryption
         47:32:bf:73:58:7f:d5:c2:5d:ee:e5:47:d5:e2:86:9c:3a:09:
         84:d0:c1:f0:32:82:13:2f:42:a9:11:6f:b5:6a:0a:c6:56:99:
         0b:f0:75:e8:3d:79:e4:e8:b2:af:08:8c:d0:fe:60:b1:49:2f:
         72:b0:31:49:6c:a9:2e:73:f0:c9:6d:23:55:89:c4:72:5d:0d:
         29:20:5b:48:d1:48:4c:a4:cf:f9:78:0e:0a:50:37:a3:b6:70:
         f3:3e:b8:cc:b1:e2:0b:5f:f4:fa:39:4a:d5:7a:46:86:f9:39:
         8f:00:ae:74:47:b5:63:24:cf:6d:9b:d5:4a:bf:ca:21:26:a3:
         26:37:7c:d4:8a:f6:2f:75:74:04:88:0a:35:b6:c0:b8:0a:d1:
         a0:4c:b9:74:6b:58:ed:fb:1c:4d:b6:55:6e:fd:27:3c:57:8f:
         bb:19:c1:e2:7d:13:51:54:3f:f1:4a:ba:88:99:76:32:83:1d:
         0b:f2:71:66:f4:d9:ee:13:9e:ee:96:99:6b:b4:dc:7f:e3:56:
         27:09:ca:35:02:94:7d:be:b7:d6:40:1e:88:3e:c6:60:4c:7c:
         6b:50:f1:df:95:5a:fe:82:55:61:bb:0f:eb:63:6b:b0:b2:40:
         b9:66:0c:3e:ab:da:e3:5f:84:e5:5f:1c:f2:ac:66:cd:0a:96:
         83:7c:f6:3c:5a:a4:a9:6d:d5:8f:2b:ea:81:62:0f:9e:e9:6d:
         97:ed:bc:c4:a1:f8:03:df:f6:0f:cb:8c:ca:60:96:42:61:bb:
         67:b1:37:bd:ae:7a:8f:b2:f0:54:2a:07:17:c6:2a:62:73:b4:
         eb:4f:4c:23:c9:95:01:0d:cd:31:d2:d9:5d:3b:4f:61:14:6b:
         c1:f9:1a:a3:8d:68:51:3f:ac:87:7e:8c:12:cd:e2:11:24:d4:
         2a:30:44:4e:db:fb:27:45:8a:c3:7d:06:5c:f7:c0:41:d8:23:
         fe:e7:72:d7:25:d5:20:85:c0:8f:a1:65:03:08:c9:51:ed:ec:
         8c:12:99:aa:55:89:eb:89:14:8d:e0:89:f1:1a:64:65:6e:21:
         1a:fe:2d:28:b7:02:da:bd:5f:2b:c0:a5:3a:88:bf:b2:2c:0f:
         94:3c:2f:43:02:9e:29:ff:c8:3b:bd:b3:43:bf:5c:71:87:65:
         40:13:7b:8d:51:62:78:ed:f6:b8:ba:b4:f7:92:53:56:29:9a:
         be:61:05:c8:6d:d8:06:ea:79:3e:1d:ec:b2:16:42:86:a5:4a:
```

```
          ca:12:c5:86:8e:d1:50:ab:0a:fd:9e:83:f7:22:cf:ea:a7:b4:
          28:83:3e:bc:e6:31:6b:79:36:5f:91:34:18:08:ec:d5:de:09:
          32:52:43:83:85:53:67:76
```

pt-ncp-sign.csr

```
$ openssl req -text -noout -verify -in pt-ncp-sign.csr
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=PT, O=SPMS, GN=Joao, SN=Cunha, CN=qaepsos.min-saude.pt
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d0:94:89:59:50:b2:68:b7:aa:e4:07:08:e6:3f:
                    1c:77:cd:a9:23:c8:51:4c:f2:1b:7c:f2:e6:42:28:
                    d6:90:cc:47:8c:e8:9e:29:5b:04:7a:dd:40:4c:e1:
                    54:f7:f4:0d:72:5b:88:2c:3f:b5:fb:68:01:02:78:
                    2c:3f:a7:6a:bf:57:2d:57:d9:e1:c6:b4:f3:4d:4e:
                    66:b3:d8:ed:96:ef:9e:c7:92:91:16:d2:bb:67:88:
                    7d:72:e2:8a:45:57:6f:66:8b:85:64:2c:60:c6:11:
                    12:29:46:3c:21:1c:d4:4a:47:ce:01:ee:43:7f:7a:
                    90:82:f9:b1:25:e1:6b:79:19:39:9f:ee:58:37:ed:
                    1d:e1:a0:5f:f9:5b:7d:13:d5:b3:62:ba:90:11:2a:
                    23:55:9e:4b:90:4d:5d:30:e1:48:13:09:dc:ec:ce:
                    e2:46:93:e4:64:3e:c2:6f:8d:7f:75:9d:0f:b9:7e:
                    03:8e:94:24:68:00:12:ce:57:0d:ce:e1:7b:de:5a:
                    fe:d6:d3:c7:6f:a3:a2:ca:e9:d4:41:86:3c:20:78:
                    f4:e8:d7:9c:86:79:6b:ae:a7:b1:6c:16:2a:86:9d:
                    54:84:be:08:57:6f:f1:53:1f:2e:ca:ad:81:16:26:
                    61:89:e6:c2:c5:40:ed:2b:e6:0a:c9:30:96:0f:a9:
                    77:81
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                C4:22:21:A7:EB:87:D6:EB:15:BF:23:5D:48:7E:F5:24:E6:80:C5:29
    Signature Algorithm: sha512WithRSAEncryption
         94:6b:76:c5:ca:f8:08:e3:42:c3:34:95:c3:e4:1b:72:d0:48:
         81:6d:f3:6c:fc:d3:09:b4:a3:c7:19:eb:a2:68:6e:4c:bf:47:
         59:69:91:9d:bc:42:c8:7a:08:42:00:c0:cb:30:99:c5:31:34:
         2d:e6:9a:90:06:c3:d7:6d:5d:78:31:75:d7:ac:30:09:89:d1:
         9d:d4:37:35:6f:45:4e:52:b5:7c:27:08:0b:3b:c2:13:a9:05:
         fa:8e:a8:21:40:e1:9b:c9:aa:47:7f:de:eb:cd:28:3f:c1:90:
         62:fc:78:43:f5:65:58:d8:e1:bc:8b:68:0b:ba:dd:6f:f9:3c:
```

```
c7:8c:b0:e9:d9:67:c0:da:37:57:b1:a7:83:19:87:e3:af:b8:
fa:57:67:04:b1:21:9e:e6:58:9e:4e:68:e8:a2:c7:c3:3d:0c:
05:bb:cf:92:f7:3c:86:63:99:a8:4c:25:04:ae:18:62:0f:31:
77:0e:9d:1f:52:e3:32:bd:84:3e:8c:ec:fa:4c:6e:56:9c:5c:
ed:de:c6:3c:5d:f0:c9:b1:4a:39:f9:dc:bd:a9:7d:80:3e:94:
```

```
          3a:f4:1a:e4:fc:8e:0a:dc:33:bb:21:b0:eb:5a:b9:b7:89:e2:
          9a:6f:3f:b0:b0:a1:ec:7c:83:a7:57:72:2e:fc:97:01:f9:6d:
          2f:f6:b1:f6
```

pt-signature-keystore.jks

```
$ keytool -v -list -keystore pt-signature-keystore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: ppt.ncp-signature.epsos.spms.pt
Creation date: Sep 13, 2016
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=qaepsos.min-saude.pt, SURNAME=Cunha, GIVENNAME=Joao, O=SPMS, C=PT
Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Serial number: e3b8582628c733d6
Valid from: Tue Sep 13 16:44:02 WEST 2016 until: Wed Sep 13 16:44:02 WEST
2017
Certificate fingerprints:
     MD5:  98:0A:54:B6:DE:7F:A2:4A:9D:83:65:49:16:3D:39:03
     SHA1: EB:DD:5E:CA:48:E0:A3:22:63:FC:5A:80:FC:C4:6C:3C:AF:B4:7C:B4
     SHA256:
FA:35:77:F2:28:B6:AD:7A:F6:C7:FF:24:C0:96:F3:22:78:36:FF:65:75:9F:4D:1B:FC
:4B:EC:8C:A4:27:B4:61
     Signature algorithm name: SHA512withRSA
     Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:false
  PathLen: undefined
]

#3: ObjectId: 2.5.29.31 Criticality=false
```

```
CRLDistributionPoints [
  [DistributionPoint:
     [URIName: http://your.cert.authority/CRL/openncp-crl.pem]
]]

#4: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  DigitalSignature
  Non_repudiation
]

#5: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  RFC822Name: joao.cunha@spms.min-saude.pt
]

#6: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C4 22 21 A7 EB 87 D6 EB   15 BF 23 5D 48 7E F5 24  ."!.......#]H..$
0010: E6 80 C5 29                                        ...)
]
]



*******************************************
*******************************************


Alias name: ppt.ca.epsos.spms.pt
Creation date: Sep 13, 2016
Entry type: trustedCertEntry

Owner: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Serial number: fbb9a3d56e7d84a3
Valid from: Tue Sep 13 15:55:38 WEST 2016 until: Fri Sep 11 15:55:38 WEST
2026
Certificate fingerprints:
     MD5:  AD:63:EF:F9:F5:EC:DB:75:43:7A:23:2D:E2:71:91:02
     SHA1: F6:29:73:DC:76:54:4F:6D:90:48:FC:A8:29:92:C5:8E:2A:48:D4:F9
     SHA256:
23:B1:F6:52:CC:0D:F3:F1:49:F0:91:A9:A3:A7:E2:09:7C:F6:CE:FA:88:17:4D:8D:55
:E4:97:F1:06:47:52:A5
     Signature algorithm name: SHA256withRSA
     Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
```

```
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]
```

```
****************************************
****************************************
```

pt-truststore.jks

```
$ keytool -v -list -keystore pt-truststore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: ppt.ncp-signature.epsos.spms.pt
Creation date: Sep 16, 2016
Entry type: trustedCertEntry

Owner: CN=qaepsos.min-saude.pt, SURNAME=Cunha, GIVENNAME=Joao, O=SPMS, C=PT
Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Serial number: e3b8582628c733d7
Valid from: Tue Sep 13 16:44:03 WEST 2016 until: Wed Sep 13 16:44:03 WEST
2017
Certificate fingerprints:
     MD5:  18:E5:0A:8C:F9:B3:49:BC:6D:E6:FD:53:78:FF:36:CF
     SHA1: F2:41:71:C3:2B:30:C0:F7:93:E5:32:80:23:EF:61:AD:16:B5:FF:23
     SHA256:
86:90:78:DA:63:6D:C1:F9:B1:98:C7:04:C2:EA:36:62:F1:C3:79:3C:3E:71:A1:03:6B
:7C:DA:A7:32:3F:6A:52
     Signature algorithm name: SHA512withRSA
     Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:false
  PathLen: undefined
]

#3: ObjectId: 2.5.29.31 Criticality=false
CRLDistributionPoints [
```

```
  [DistributionPoint:
     [URIName: http://your.cert.authority/CRL/openncp-crl.pem]
]]

#4: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  DigitalSignature
  Non_repudiation
]

#5: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  RFC822Name: joao.cunha@spms.min-saude.pt
]

#6: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C4 22 21 A7 EB 87 D6 EB   15 BF 23 5D 48 7E F5 24  ."!.......#]H..$
0010: E6 80 C5 29                                        ...)
]
]



*******************************************
*******************************************


Alias name: ppt.ca.epsos-spms.pt
Creation date: Sep 16, 2016
Entry type: trustedCertEntry

Owner: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
Serial number: fbb9a3d56e7d84a3
Valid from: Tue Sep 13 15:55:38 WEST 2016 until: Fri Sep 11 15:55:38 WEST
2026
Certificate fingerprints:
     MD5:  AD:63:EF:F9:F5:EC:DB:75:43:7A:23:2D:E2:71:91:02
     SHA1: F6:29:73:DC:76:54:4F:6D:90:48:FC:A8:29:92:C5:8E:2A:48:D4:F9
     SHA256:
23:B1:F6:52:CC:0D:F3:F1:49:F0:91:A9:A3:A7:E2:09:7C:F6:CE:FA:88:17:4D:8D:55
:E4:97:F1:06:47:52:A5
     Signature algorithm name: SHA256withRSA
     Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
```

```
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]
```

```
****************************************
****************************************
```

The signed SMP file:

Consent_Service_Discard_PT.xml

```
<?xml version="1.0" encoding="UTF-8"?><ServiceMetadata
xmlns="http://busdox.org/serviceMetadata/publishing/1.0/"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:ids="http://busdox.org/transport/identifiers/1.0/"
xmlns:ns="urn:esens:smp" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
ity-utility-1.0.xsd" xmlns:xades="http://uri.etsi.org/01903/v1.3.2#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ServiceInformation>
<ids:ParticipantIdentifier
scheme="ehealth-actorid-qns">urn:ehealth:pt:ncpb-idp</ids:ParticipantIdent
ifier><ids:DocumentIdentifier
scheme="ehealth-resid-qns">urn::epsos##services:extended:epsos::52</ids:Do
cumentIdentifier><ProcessList><Process><ids:ProcessIdentifier
scheme="ehealth-procid-qns">urn:epsosConsentService::Discard</ids:ProcessI
dentifier><ServiceEndpointList><Endpoint
transportProfile="urn:ihe:iti:2013:xdr"><wsa:EndpointReference><wsa:Addres
s>https://qaepsos.min-saude.pt:8443/epsos-ws-server/services/XDR_Service</
wsa:Address></wsa:EndpointReference><RequireBusinessLevelSignature>false</
RequireBusinessLevelSignature><MinimumAuthenticationLevel>urn:epSOS:loa:1<
/MinimumAuthenticationLevel><ServiceActivationDate>2016-06-06T11:06:51Z</S
erviceActivationDate><ServiceExpirationDate>2026-06-06T11:06:51Z</ServiceE
xpirationDate><Certificate>MIID7jCCA1egAwIBAgICA+YwDQYJKoZIhvcNAQENBQAwOjE
LMAkGA1UEBhMCRlIxEzARBgNVBAoMCklIRSBFdXJvcGUxFjAUBgNVBAMMDUlIRSBFdXJvcGUgQ
0EwHhcNMTYwNjAxMTQzNTUzWhcNMjYwNjAxMTQzNTUzWjCBgzELMAkGA1UEBhMCUFQxDDAKBgN
VBAoMA01vSDENMAsGA1UECwwEU1BNUzENMAsGA1UEKgwESm9hbzEOMAwGA1UEBRMFQ3VuaGExH
TAbBgNVBAMMFHFhZXBzb3MubWluLXNhdWRlLnB0MRkwFwYDVQQMDBBTZXJ2aWNlIFByb3ZpZGV
yMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1eN4qPSSRZqjVFG9TlcPlxf2WiSim
QK9L1nf9Z/s0ezeGQjCukDeDq/Wzqd9fpHhaMMq+XSSOtyEtIr5K/As4kFrViONUUkG12J6Ull
SWogp0NYFwA4wIqKSFiTnQS5/nRTs05oONCCGILCyJNNeO53JzPlaq3/QbPLssuSAr6XucPE8w
BBGM8b/TsB2G/zjG8yuSTgGbhaZekq/Vnf9ftj1fr/vJDDAQgH6Yvzd88Z0DACJPHfW1p4F/OW
LI386Bq7g/bo1DUPAyEwlf+CkLgJWRKki3yJlOCIZ9enMA5O7rfeG3rXdgYGmWS7tNEgKXxgC+
heiYvi7ZWd7M+/SUwIDAQABo4IBMzCCAS8wPgYDVR0fBDcwNTAzoDGgL4YtaHR0cHM6Ly9nYXp
lbGxlLmloZS5uZXQvcGtpL2NybC82NDMvY2FjcmwuY3JsMDwGCWCGSAGG+EIBBAQvFi1odHRwc
zovL2dhemVsbGUuaWhlLm5ldC9wa2kvY3JsLzY0My9jYWNybC5jcmwwPAYJYIZIAYb4QgEDBC8
WLWh0dHBzOi8vZ2F6ZWxsZS5paGUubmV0L3BraS9jcmwvNjQzL2NhY3JsLmNybDAfBgNVHSMEG
DAWgBTsMw4TyCJeouFrr0N7el3Sd3MdfjAdBgNVHQ4EFgQU1GQ/K1ykIwWFgiONzWJLQzufF/8
wDAYDVR0TAQH/BAIwADAOBgNVHQ8BAf8EBAMCBSAwEwYDVR0lBAwwCgYIKwYBBQUHAwEwDQYJK
oZIhvcNAQENBQADgYEAZ7t1Qkr9wz3q6+WcF6p/YX7Jr0CzVe7w58FvJFk2AsHeYkSlOyO5hxN
pQbs1L1v6JrcqziNFrh2QKGT2v6iPdWtdCT8HBLjmuvVWxxnfzYjdQ0J+kdKMAEV6EtWU78OqL
60CCtUZKXE/NKJUq7TTUCFP2fwiARy/t1dTD2NZo8c=</Certificate><ServiceDescripti
on>This is the epSOS Consent Service Discard of the PT
NCP</ServiceDescription><TechnicalContactUrl>licinio.mano@spms.min-saude.p
t</TechnicalContactUrl><TechnicalInformationUrl>licinio.mano@spms.min-saud
e.pt</TechnicalInformationUrl><Extension><root xmlns=""><Signature
```

xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/><SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/><Reference
URI=""><Transforms><Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/></Transforms><DigestMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/><DigestValue>sok0hQY2jLtyLQYN960cvAnOyVERM7KjjXdnRRpIgE0=</DigestValue></Reference></SignedInfo><SignatureValue>sy3Lwx5psKjrctDqlKYDHSFyOjSnJmdS7TZEPnf/fq9caesct0Uwr7kmuRQckQB3orr5aFDuAklW
c+1uQs9ZzEMSW2WCCy//Y9r4CfeIw/Din+Z1AVMA5IQxOReFfGQ+r+B5M/J3t9+GnOyq6KXzJxe/
cVlviWFegi4OihiOfAQTLNttE1vu8BCSEIeHlBXUAjecuOaPf7Y0TFq5gWWB0JxlpEHA4nnq6wXn
NbSxtCnx3L8LXJyhFYimWwFa8c5ed142WYKY6G1Z/oDpEkQwDp4MPqw62H4ccOGwNykgspx7bLsP
eAKkvyX9f8tOtfDPIuzy6c7wThxEtDuXskRblA==</SignatureValue><KeyInfo><X509Data><X509SubjectName>CN=qaepsos.min-saude.pt,2.5.4.4=#130543756e6861,2.5.4.42=#13044a6f616f,O=SPMS,C=PT</X509SubjectName><X509Certificate>MIIFNjCCAx6gAwIBAgIJAOO4WCYoxzPWMA0GCSqGSIb3DQEBDQUAMIGMMQswCQYDVQQGEwJQVDEO
MAwGA1UECAwFUG9ydG8xDjAMBgNVBAcMBVBvcnRvMQ0wCwYDVQQKDARTUE1TMQwwCgYDVQQLDANJ
T1AxEzARBgNVBAMMCmpvYW8uY3VuaGExKzApBgkqhkiG9w0BCQEWHGpvYW8uY3VuaGFAc3Btcy5t
aW4tc2F1ZGUucHQwHhcNMTYwOTEzMTU0NDAyWhcNMTcwOTEzMTU0NDAyWjBaMQswCQYDVQQGEwJQ
VDENMAsGA1UEChMEU1BNUzENMAsGA1UEKhMESm9hbzEOMAwGA1UEBBMFQ3VuaGExHTAbBgNVBAMT
FHFhZXBzb3MubWluLXNhdWRlLnB0MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0JSJ
WVCyaLeq5AcI5j8cd82pI8hRTPIbfPLmQijWkMxHjOieKVsEet1ATOFU9/QNcluILD+1+2gBAngs
P6dqv1ctV9nhxrTzTU5ms9jtlu+ex5KRFtK7Z4h9cuKKRVdvZouFZCxgxhESKUY8IRzUSkfOAe5D
f3qQgvmxJeFreRk5n+5YN+0d4aBf+Vt9E9WzYrqQESojVZ5LkE1dMOFIEwnc7M7iRpPkZD7Cb41/
dZ0PuX4DjpQkaAASzlcNzuF73lr+1tPHb6OiyunUQYY8IHj06Nechnlrrqexb BYqhp1UhL4IV2/x
Ux8uyq2BFiZhiebCxUDtK+YKyTCWD6l3gQIDAQABo4HLMIHIMCcGA1UdEQQgMB6BHGpvYW8uY3VuaGFAc3Btcy5taW4tc2F1ZGUucHQwDgYDVR0PAQH/BAQDAgbAMB8GA1UdIwQYMBaAFHWz6cJ961Cg
vw53QD58LN7XzxZcMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFMQiIafrh9brFb8jXUh+9STmgMUp
MD8GA1UdHwQ4MDYwNKAyoDCGLmh0dHA6Ly95b3VyLmNlcnQuYXV0aG9yaXR5L0NSTC9vcGVubmNw
LWNybC5wZW0wDQYJKoZIhvcNAQENBQADggIBACycmMwqEqL2/PMKnc1My0WablbzUy6wjJY49vvm
MRdX6k/4USBR/Y8oZEmvmXxE6AYeZRlwAqRMPpH4zXk0zO74RbuVchHM5L173F2P91qFgLgyFEDD
OjII17zQeEEYYL78eMLswnqk4XLYrchOG/Ql+cXhMJafELEU4VLiJnFWcnb1D+spMXobO5ramO

dZ
wVMvbQJUqRXqPvoJ4ZYhktoW+p2nPQahbDuv55FympVEzFJBPMUvGIanLMauTKcMMZzElUch3V
yB
qXyDzj2SqajKvAUVoU38lNauDwe9PujcmvaW0J/6RJ7SbkU20j1Y/urSn6QBMJfsHW2LwX5cto
+u
lacXUHjcWZhimaJJ86atjAfDwWE6y0f/ENSzi8eZ693hywmRgU6te7H4lOzUzd3exMbiFNxVTf
mM
HImou8ZBHUb7wiVUNShc/Q9F1gppTpUwd8gV74b3ZeyUCZ6o7EDX/OSdZNbPxRlo+cNrQ0Gp8J
oW
EJ3R6kJzMhAMs/DYwUAfKyfocekhaUEIxFBu4JP5rR03a60rEjl3v5mHWutbn86g2b2TrGssl/
Zs
LJxyIBM1vDSJ75gGYIwzje/QmKaSGcnr1RJo5Z6DG25Tf0kbWUWJxMyGj8dSnsfHkoiTU4mfco
Zc

```
amfcnXGeHm0vLvTS3j3G31fo1uZMIJ53V5r5</X509Certificate></X509Data></KeyInfo
></Signature></root></Extension></Endpoint></ServiceEndpointList></Process
></ProcessList></ServiceInformation></ServiceMetadata>
```

Ok, so far so good. We have a signed SMP file, a truststore with the renewed certificate and its root CA, and we want to verify the SMP file against it using the source code of the Signature Manager. This component already provides a set of test classes that were copied into a new project developed specifically for this test. Plus, the unused source code was removed from the copied classes. Following is a description and paste of the code:

SMgrException: encapsulates exceptions thrown by this component.

SMgrException

```
/*
 *  Copyright 2010 Jerry Dimitriou <jerouris at netsmart.gr>.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */

package org.openncp.renewcert.test;

import javax.xml.crypto.dsig.XMLSignatureException;

/**
 * The Exception that is thrown by the NCP Signature Manager. Provides an
error
 * message.
 * @author Jerry Dimitriou <jerouris at netsmart.gr>
 */
public class SMgrException extends XMLSignatureException {

    /**
     *
     */
    private static final long serialVersionUID = 1L;


    /**
     * Constructor of the exception
     * @param exceptionMessage - the error message
     */
    public SMgrException(String exceptionMessage) {
        super(exceptionMessage);
    }


    public SMgrException(String exceptionMessage, Exception e) {
        super(exceptionMessage, e);
    }

}
```

KeyStoreManager: interface that defines the set of methods for managing keystores, truststores, their keypairs and certificates:

KeyStoreManager

```
/*
 *  Copyright 2010 Jerry Dimitriou.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */

package org.openncp.renewcert.test;

import java.security.KeyPair;
import java.security.KeyStore;
import java.security.cert.Certificate;

/** This is a helper Interface that manages the keystores, truststores
 * and their respective keys and certificates.
 *
 * @author Jerry Dimitriou
 */
public interface KeyStoreManager {

/**
 * Tries to fetch the default KeyPair from the default keystore, as it is
configured in the
 * Configuration manager.
 *
 * @return the KeyPair that contains the default Private Key as it is
configured in the
 * configuration manager
 * @throws SMgrException
 */
    public KeyPair getDefaultPrivateKey() throws SMgrException;

    /**
 * Tries to fetch the default Certificate from the default keystore, as it
is configured in the
 * Configuration manager.
 *
 * @return the Certificate that contains the default Public Key as it is
configured in the
 * configuration manager
```

```
 * @throws SMgrException
 */
    public Certificate getDefaultCertificate() throws SMgrException;

 /**
 * Tries to fetch the Private Key with alias <i>alias</i>, from the default
keystore as it is configured in the
 * Configuration manager.
 * @param alias the Key Alias
  * @param password the private key password.
 * @return the Certificate that matches the alias
 * @throws SMgrException
 */
    public abstract KeyPair getPrivateKey(String alias, char[] password)
throws SMgrException;
    public Certificate getCertificate(String alias) throws SMgrException;

    public KeyStore getKeyStore();
    public KeyStore getTrustStore();
```

```
}
```

NSTestKeyStoreManager: implements the previous interface, providing a way to access the artifacts we created earlier (keystore, truststore, private key and certificates).

NSTestKeyStoreManager

```
/*
 *  Copyright 2010 jerouris.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */

package org.openncp.renewcert.test;

import java.io.IOException;
import java.io.InputStream;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.util.Enumeration;
import java.util.logging.Level;

import org.apache.log4j.Logger;
import pt.minsaude.spms.iop.ncp.smp.xslt.fix.SmpXslt;


/**
 *
 * @author jerouris
 */
```

```
public class NSTestKeyStoreManager implements KeyStoreManager {
    private static final Logger logger =
Logger.getLogger(NSTestKeyStoreManager.class);


    private static final String TEST_KEYSTORE_LOCATION =
"keystores/pt-signature-keystore.jks";
    private static final String TEST_TRUSTSTORE_LOCATION =
"keystores/pt-signature-keystore-2.jks";
//    private static final String TEST_TRUSTSTORE_LOCATION =
"keystores/pt-truststore.jks";
    private static final String TEST_KEYSTORE_PASSWORD = "changeit";
    private static final String TEST_TRUSTSTORE_PASSWORD = "changeit";

    private static final String TEST_PRIVATEKEY_ALIAS =
"ppt.ncp-signature.epsos.spms.pt";
    private static final String TEST_PRIVATEKEY_PASSWORD = "changeit";

    private KeyStore keyStore;
    private KeyStore trustStore;

    public NSTestKeyStoreManager()
    {
        // For testing purposes...
            if (keyStore == null)
            {
                keyStore = getTestKeyStore();
                trustStore = getTestTrustStore();

                try {
                    Enumeration<String> aliases = trustStore.aliases();
                    while (aliases.hasMoreElements())
                        logger.info("Alias: " + aliases.nextElement());
                } catch (KeyStoreException ex) {

java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(Level.SEVERE, null, ex);
                }
            }
    }


    @Override
    public KeyPair getPrivateKey(String alias, char[] password) throws
SMgrException{


        try {

            // Get private key
            Key key = keyStore.getKey(alias, password);
            if (key instanceof PrivateKey) {
                // Get certificate of public key
```

```
                java.security.cert.Certificate cert =
keyStore.getCertificate(alias);

                // Get public key
                PublicKey publicKey = cert.getPublicKey();

                // Return a key pair
                return new KeyPair(publicKey, (PrivateKey) key);
            }
        } catch (UnrecoverableKeyException e) {
            Logger.getLogger(SignatureManager.class.getName()).error(null,
e);
            throw new SMgrException("Key with alias:"+alias+" is
unrecoverable", e);
        } catch (NoSuchAlgorithmException e) {
            Logger.getLogger(SignatureManager.class.getName()).error(null,
e);
            throw new SMgrException("Key with alias:"+alias+" uses an
incompatible algorithm", e);
        } catch (KeyStoreException e) {
            Logger.getLogger(SignatureManager.class.getName()).error(null,
e);
            throw new SMgrException("Key with alias:"+alias+" not found",
e);
        }
        return null;
    }

     private KeyStore getTestKeyStore() {
        try {
            keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            InputStream keystoreStream =
ClassLoader.getSystemResourceAsStream(TEST_KEYSTORE_LOCATION);
            keyStore.load(keystoreStream,
TEST_KEYSTORE_PASSWORD.toCharArray());

            return keyStore;

        } catch (IOException ex) {
            Logger.getLogger(SignatureManager.class.getName()).error(null,
ex);
        } catch (NoSuchAlgorithmException ex) {
          Logger.getLogger(SignatureManager.class.getName()).error(null,
ex);
        } catch (CertificateException ex) {
          Logger.getLogger(SignatureManager.class.getName()).error(null,
ex);
        } catch (KeyStoreException ex) {
          Logger.getLogger(SignatureManager.class.getName()).error(null,
ex);
        }
        return null;
    }
```

```
    @Override
    public KeyStore getKeyStore() {
       return keyStore;
    }

    @Override
    public KeyStore getTrustStore() {
        return trustStore;

    }

    @Override
    public Certificate getCertificate(String alias) throws SMgrException {
        try {
            java.security.cert.Certificate cert =
keyStore.getCertificate(alias);
            return cert;
        }
        catch (KeyStoreException ex) {

Logger.getLogger(NSTestKeyStoreManager.class.getName()).error(null, ex);
                        throw new SMgrException("Certificate with
alias:"+alias+" not found in keystore", ex);
        }

    }

    private KeyStore getTestTrustStore() {
        try {
            trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
            InputStream keystoreStream =
ClassLoader.getSystemResourceAsStream(TEST_TRUSTSTORE_LOCATION);
            trustStore.load(keystoreStream,
TEST_TRUSTSTORE_PASSWORD.toCharArray());
            return trustStore;
        }
        catch (IOException ex) {

java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(java.util.logging.Level.SEVERE, null, ex);
        }
        catch (NoSuchAlgorithmException ex) {

java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(java.util.logging.Level.SEVERE, null, ex);
        }
        catch (CertificateException ex) {

java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(java.util.logging.Level.SEVERE, null, ex);
        }        catch (KeyStoreException ex) {
```

```
java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(java.util.logging.Level.SEVERE, null, ex);
        }
        return null;
    }

    @Override
    public KeyPair getDefaultPrivateKey() throws SMgrException {
        return getPrivateKey(TEST_PRIVATEKEY_ALIAS,
TEST_PRIVATEKEY_PASSWORD.toCharArray());
    }

    @Override
     public Certificate getDefaultCertificate() throws SMgrException {
        return getCertificate(TEST_PRIVATEKEY_ALIAS);
```

```
    }

}
```

CertificateValidator: implements KeySelector, defining the customized procedures to follow for returning the public key used for validating the signature. Notice that this class is slightly different from the one that is being used currently in the OpenNCP: **I uncommented the check of the trust of the certificate chain** (in method validateCertificate(X509Certificate cert)). I also turned off the check of the CRL because I was using self-signed certificates, but it was just for testing purposes.

CertificateValidator

```
/*
 *  Copyright 2010 Jerry Dimitriou <jerouris at netsmart.gr>.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */
package org.openncp.renewcert.test;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.Key;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PublicKey;
import java.security.cert.CertPath;
import java.security.cert.CertPathBuilder;
import java.security.cert.CertPathBuilderException;
import java.security.cert.CertPathBuilderResult;
import java.security.cert.CertPathValidator;
import java.security.cert.CertPathValidatorException;
import java.security.cert.CertStore;
import java.security.cert.CertStoreParameters;
import java.security.cert.Certificate;
import java.security.cert.CertificateExpiredException;
import java.security.cert.CertificateNotYetValidException;
import java.security.cert.CollectionCertStoreParameters;
import java.security.cert.PKIXBuilderParameters;
import java.security.cert.PKIXCertPathValidatorResult;
import java.security.cert.PKIXParameters;
```

```
import java.security.cert.X509CertSelector;
import java.security.cert.X509Certificate;
import java.util.Collections;
import java.util.Date;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.crypto.AlgorithmMethod;
import javax.xml.crypto.KeySelector;
import javax.xml.crypto.KeySelectorException;
import javax.xml.crypto.KeySelectorResult;
import javax.xml.crypto.XMLCryptoContext;
import javax.xml.crypto.XMLStructure;
import javax.xml.crypto.dsig.keyinfo.KeyInfo;
import javax.xml.crypto.dsig.keyinfo.X509Data;

/**
 * The Certificate Validator is a component that is responsible for
validating
 * certificates against the NCP Trust Store. The certificate validation
consists
 * of checking if the certificate is trusted and if it is not revoked: All
 * certificates registered with the local NCP trust store are assumed as
 * trusted. For revocation check both CRL retrieval and OCSP are supported.
 *
 * @author Jerry Dimitriou <jerouris at netsmart.gr>
 */
public final class CertificateValidator extends KeySelector {
    private static final org.apache.log4j.Logger logger =
org.apache.log4j.Logger.getLogger(CertificateValidator.class);

    private Certificate cert = null;
    private final KeyStore trustStore;
    private boolean isRevocationEnabled = false;
    public static final String CRLDP_OID = "2.5.29.31";
    public static final String AIA_OID = "1.3.6.1.5.5.7.1.1";
    public static final String PROP_CHECK_FOR_KEYUSAGE =
"secman.cert.validator.checkforkeyusage";

    /**
     *
     */
    public boolean CHECK_FOR_KEYUSAGE;

    public CertificateValidator(KeyStore trustStore) throws
FileNotFoundException, IOException {
        CHECK_FOR_KEYUSAGE = false;
        this.trustStore = trustStore;
    }

    /**
     * This method verifies the validity of a certificate by a given
```

```
keyInfo
     * structure
     *
     * @param keyInfo The keyInfo structure that contains the certificate
     * (X509).
     * @throws SMgrException when the validation of the certificate fails
     */
    public void validateCertificate(KeyInfo keyInfo) throws SMgrException {

        for (Iterator<?> it = keyInfo.getContent().iterator();
it.hasNext();) {
            Object object = it.next();

            if (object instanceof X509Data) {
                X509Data data = (X509Data) object;
                Iterator<?> xi = data.getContent().iterator();
                while (xi.hasNext()) {
                    Object o = xi.next();
                    // check X509Certificate
                    if (o instanceof X509Certificate) {
                        X509Certificate xcert = (X509Certificate) o;
                        validateCertificate(xcert);
                        cert = xcert;
                        return;
                    } else {
                        // skip all other entries
                        continue;
                    }
                }

                throw new SMgrException("The KeyInfo Structure does not
contain X509Data element: No Certificate Present");
            }
        }
    }


    /**
     * This method verifies the validity of the given X509 certificate by
     * checking the truststore.
     *
     * @param cert the certificate that will be validated
     * @throws SMgrException when the validation of the certificate fails
     */
    public void validateCertificate(X509Certificate cert) throws
SMgrException {
        logger.info("Validating the following certificate: \n" +  cert);
        try {
            if (CHECK_FOR_KEYUSAGE) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.INFO, "Key
usage available in conf manager");
                boolean[] keyUsage = cert.getKeyUsage();
```

```
                if (keyUsage == null || keyUsage[0] == false) {
                    throw new SMgrException("Certificate Key Usage
Extension for Digital Signature Missing");
                }
            }

            try {
                cert.checkValidity(new Date());
            } catch (CertificateExpiredException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
                throw new SMgrException("Certificate Expired", ex);
            } catch (CertificateNotYetValidException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
                throw new SMgrException("Certificate Not Valid Yet", ex);
            }

            // Check if the cert supports the CRLDP
            if (cert.getExtensionValue(AIA_OID) != null) {
                setRevocationEnabled(true);
                setOCSPEnabled(true);

Logger.getLogger(CertificateValidator.class.getName()).log(Level.INFO,
"Found AIA Extension. Using OCSP");
            }

            if (cert.getExtensionValue(CRLDP_OID) != null) {
                setRevocationEnabled(true);
                setCRLDPEnabled(true);

Logger.getLogger(CertificateValidator.class.getName()).log(Level.INFO,
"Found CRLDP Extension. Using CRLDP");
            }

            CertStoreParameters intermediates
                    = new
CollectionCertStoreParameters(Collections.singletonList(cert));

            X509CertSelector target = new X509CertSelector();
            target.setCertificate(cert);

            PKIXBuilderParameters builderParams = new
PKIXBuilderParameters(trustStore, target);

            builderParams.addCertStore(CertStore.getInstance("Collection",
intermediates));

            builderParams.setRevocationEnabled(false); // Joao: I set it to
false to avoid check of CRL of my self-signed cert
```

```
            builderParams.setDate(new Date());

            CertPathBuilder builder = CertPathBuilder.getInstance("PKIX");
            CertPathBuilderResult build = builder.build(builderParams);
            CertPath cp = build.getCertPath();

            CertPathValidator cpv = CertPathValidator.getInstance("PKIX");

            PKIXParameters params = new PKIXParameters(trustStore);
            params.setRevocationEnabled(false); // Joao: I set it to false
to avoid check of CRL of my self-signed cert

            PKIXCertPathValidatorResult validationResult =
(PKIXCertPathValidatorResult) cpv.validate(cp, params);
            logger.info(validationResult);

        } catch (NoSuchAlgorithmException ex) {


Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
            throw new SMgrException("Certificate's Public key algorithm is
unknown", ex);

        } catch (KeyStoreException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
            throw new SMgrException("Error when tried to use the
TrustStore", ex);
        } catch (InvalidAlgorithmParameterException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
            throw new SMgrException("Invalid Algorith parameters for
building Certificate Path", ex);
        } catch (CertPathBuilderException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
            throw new SMgrException("Certification path building failed",
ex);
        } catch (CertPathValidatorException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
            throw new SMgrException("Certification path validation failed",
ex);
        }

    }

    public KeySelectorResult select(KeyInfo keyInfo,
```

```
            KeySelector.Purpose purpose,
            AlgorithmMethod method,
            XMLCryptoContext context)
            throws KeySelectorException {
        Iterator<?> ki = keyInfo.getContent().iterator();
        while (ki.hasNext()) {
            XMLStructure info = (XMLStructure) ki.next();
            if (!(info instanceof X509Data)) {
                continue;
            }
            X509Data x509Data = (X509Data) info;
            Iterator<?> xi = x509Data.getContent().iterator();
            while (xi.hasNext()) {
                Object o = xi.next();
                if (!(o instanceof X509Certificate)) {
                    continue;
                }
                try {
                    validateCertificate(keyInfo);
                } catch (SMgrException ex) {

Logger.getLogger(CertificateValidator.class.getName()).log(Level.SEVERE,
null, ex);
                    throw new KeySelectorException("Validation Failed: " +
ex.getMessage());
                }
                final PublicKey key = ((X509Certificate) o).getPublicKey();
                // Make sure the algorithm is compatible
                // with the method.
                if (algEquals(method.getAlgorithm(), key.getAlgorithm())) {
                    return new KeySelectorResult() {
                        public Key getKey() {
                            return key;
                        }
                    };
                }
            }
        }
        throw new KeySelectorException("No key found!");
    }

    static boolean algEquals(String algURI, String algName) {
        if ((algName.equalsIgnoreCase("DSA")
                && algURI.contains("#dsa"))
                || (algName.equalsIgnoreCase("RSA")
                && algURI.contains("#rsa"))) {
            return true;
        } else {
            return false;
        }
    }

    /**
```

```
     * @return the X509 Certificate that is validated from the #validate
     * function
     */
    public Certificate getValidatedCertificate() {
        return cert;
    }

    public void setOCSPEnabled(boolean flag) {
        java.security.Security.setProperty("ocsp.enable",
Boolean.toString(flag));
    }

    public void setCRLDPEnabled(boolean flag) {
        System.setProperty("com.sun.security.enableCRLDP",
Boolean.toString(flag));
    }

    public void setRevocationEnabled(boolean flag) {
        this.isRevocationEnabled = flag;
    }
```

```
}
```

X509KeySelector: alternative implementation of KeySelector.

X509KeySelector

```
/*
 *  Copyright 2010 Jerry Dimitriou <jerouris at netsmart.gr>.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */

package org.openncp.renewcert.test;


import java.io.IOException;
import java.security.Key;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.PublicKey;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.CertSelector;
import java.security.cert.CertificateEncodingException;
import java.security.cert.X509Certificate;
import java.security.cert.X509CertSelector;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.security.auth.x500.X500Principal;
import javax.xml.crypto.*;
import javax.xml.crypto.dsig.*;
import javax.xml.crypto.dsig.keyinfo.*;

/**
 * A <code>KeySelector</code> that returns {@link PublicKey}s of trusted
 * {@link X509Certificate}s stored in a {@link KeyStore}.
 *
```

```
 * <p>This <code>KeySelector</code> uses the specified
<code>KeyStore</code>
 * to find a trusted <code>X509Certificate</code> that matches information
 * specified in the {@link KeyInfo} passed to the {@link #select} method.
 * The public key from the first match is returned. If no match,
 * <code>null</code> is returned. See the <code>select</code> method for
more
 * information.
 *
 * @author Sean Mullan
 */
public class X509KeySelector extends KeySelector {
    private static final org.apache.log4j.Logger logger =
org.apache.log4j.Logger.getLogger(X509KeySelector.class);

    private KeyStore ks;

    /**
     * Creates an <code>X509KeySelector</code>.
     *
     * @param keyStore the keystore
     * @throws KeyStoreException if the keystore has not been initialized
     * @throws NullPointerException if <code>keyStore</code> is
     *    <code>null</code>
     */
    public X509KeySelector(KeyStore keyStore) throws KeyStoreException {
        if (keyStore == null) {
            throw new NullPointerException("keyStore is null");
        }
        this.ks = keyStore;
        // test to see if KeyStore has been initialized
        this.ks.size();

        try {
            Enumeration<String> aliases = ks.aliases();
            while (aliases.hasMoreElements())
                logger.info("Alias: " + aliases.nextElement());
        } catch (KeyStoreException ex) {

java.util.logging.Logger.getLogger(NSTestKeyStoreManager.class.getName()).
log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Finds a key from the keystore satisfying the specified constraints.
     *
     * <p>This method compares data contained in {@link KeyInfo} entries
     * with information stored in the <code>KeyStore</code>. The
implementation
     * iterates over the KeyInfo types and returns the first {@link
PublicKey}
     * of an X509Certificate in the keystore that is compatible with the
```

```
     * specified AlgorithmMethod according to the following rules for each
     * keyinfo type:
     *
     * X509Data X509Certificate: if it contains a <code>KeyUsage</code>
     *   extension that asserts the <code>digitalSignature</code> bit and
     *   matches an <code>X509Certificate</code> in the
<code>KeyStore</code>.
     * X509Data X509IssuerSerial: if the serial number and issuer DN match
an
     *    <code>X509Certificate</code> in the <code>KeyStore</code>.
     * X509Data X509SubjectName: if the subject DN matches an
     *    <code>X509Certificate</code> in the <code>KeyStore</code>.
     * X509Data X509SKI: if the subject key identifier matches an
     *    <code>X509Certificate</code> in the <code>KeyStore</code>.
     * KeyName: if the keyname matches an alias in the
<code>KeyStore</code>.
     * RetrievalMethod: supports rawX509Certificate and X509Data types. If
     *    rawX509Certificate type, it must match an
<code>X509Certificate</code>
     *    in the <code>KeyStore</code>.
     *
     * @param keyInfo a <code>KeyInfo</code> (may be <code>null</code>)
     * @param purpose the key's purpose
     * @param method the algorithm method that this key is to be used for.
     *    Only keys that are compatible with the algorithm and meet the
     *    constraints of the specified algorithm should be returned.
     * @param context an <code>XMLCryptoContext</code> that may contain
additional
     *    useful information for finding an appropriate key
     * @return a key selector result
     * @throws KeySelectorException if an exceptional condition occurs
while
     *    attempting to find a key. Note that an inability to find a key is
not
     *    considered an exception (<code>null</code> should be
     *    returned in that case). However, an error condition (ex: network
     *    communications failure) that prevented the
<code>KeySelector</code>
     *    from finding a potential key should be considered an exception.
     * @throws ClassCastException if the data type of <code>method</code>
     *    is not supported by this key selector
     */
    public KeySelectorResult select(KeyInfo keyInfo,
    KeySelector.Purpose purpose, AlgorithmMethod method,
    XMLCryptoContext context) throws KeySelectorException {

        SignatureMethod sm = (SignatureMethod) method;

        try {
            // return null if keyinfo is null or keystore is empty
            if (keyInfo == null || ks.size() == 0) {
                return new SimpleKeySelectorResult(null);
            }
```

```
    // Iterate through KeyInfo types
    Iterator<?> i = keyInfo.getContent().iterator();
    while (i.hasNext()) {
        XMLStructure kiType = (XMLStructure) i.next();
// check X509Data
        if (kiType instanceof X509Data) {
            logger.info("Found X509Data!");
            X509Data xd = (X509Data) kiType;
    KeySelectorResult ksr = x509DataSelect(xd, sm);
        if (ksr != null) {
        return ksr;
        }
// check KeyName
        } else if (kiType instanceof KeyName) {
    KeyName kn = (KeyName) kiType;
    Certificate cert = ks.getCertificate(kn.getName());
    if (cert != null && algEquals(sm.getAlgorithm(),
    cert.getPublicKey().getAlgorithm())) {
    return new SimpleKeySelectorResult(cert.getPublicKey());
    }
// check RetrievalMethod
        } else if (kiType instanceof RetrievalMethod) {
    RetrievalMethod rm = (RetrievalMethod) kiType;
            try {
    KeySelectorResult ksr = null;
        if (rm.getType().equals
        (X509Data.RAW_X509_CERTIFICATE_TYPE)) {
        OctetStreamData data = (OctetStreamData)
        rm.dereference(context);
        CertificateFactory cf =
            CertificateFactory.getInstance("X.509");
        X509Certificate cert = (X509Certificate)
            cf.generateCertificate(data.getOctetStream());
            ksr = certSelect(cert, sm);
        } else if (rm.getType().equals(X509Data.TYPE)) {
        //NodeSetData nd = (NodeSetData)
        rm.dereference(context);
        // convert nd to X509Data
            // ksr = x509DataSelect(xd, sm);
        } else {
        // skip; keyinfo type is not supported
        continue;
    }
        if (ksr != null) {
            return ksr;
            }
            } catch (Exception e) {
        throw new KeySelectorException(e);
    }
}
    }
} catch (KeyStoreException kse) {
```

```
            // throw exception if keystore is uninitialized
            throw new KeySelectorException(kse);
        }

        // return null since no match could be found
        return new SimpleKeySelectorResult(null);
    }

    /**
     * Searches the specified keystore for a certificate that matches the
     * criteria specified in the CertSelector.
     *
     * @return a KeySelectorResult containing the cert's public key if
there
     *   is a match; otherwise null
     */
    private KeySelectorResult keyStoreSelect(CertSelector cs)
    throws KeyStoreException {
        logger.info("Searching keystore with the CertSelector");
        Enumeration<?> aliases = ks.aliases();
        while (aliases.hasMoreElements()) {
        String alias = (String) aliases.nextElement();
            logger.info("Trying to match cert with the one from truststore
with alias: " + alias);
        Certificate cert = ks.getCertificate(alias);
        if (cert != null && cs.match(cert)) {
                logger.info("Found matching certificate: " + cert);
            return new SimpleKeySelectorResult(cert.getPublicKey());
        }
    }
        logger.info("Didn't find matching certificate!");
    return null;
    }

    /**
     * Searches the specified keystore for a certificate that matches the
     * specified X509Certificate and contains a public key that is
compatible
     * with the specified SignatureMethod.
     *
     * @return a KeySelectorResult containing the cert's public key if
there
     *   is a match; otherwise null
     */
    private KeySelectorResult certSelect(X509Certificate xcert,
    SignatureMethod sm) throws KeyStoreException {
        // skip non-signer certs

        boolean[] keyUsage = xcert.getKeyUsage();

        if (keyUsage[0] == false) {
            return null;
        }
```

```
        String alias = ks.getCertificateAlias(xcert);
        if (alias != null) {
            logger.info("Cert Alias found: " + alias);
            PublicKey pk = ks.getCertificate(alias).getPublicKey();
            // make sure algorithm is compatible with method
            if (algEquals(sm.getAlgorithm(), pk.getAlgorithm())) {
                return new SimpleKeySelectorResult(pk);
            }
        }
        logger.info("Couldn't obtain certificate from truststore");
    return null;
    }

    /**
     * Returns an OID of a public-key algorithm compatible with the
specified
     * signature algorithm URI.
     */
    private String getPKAlgorithmOID(String algURI) {
    if (algURI.equalsIgnoreCase(SignatureMethod.DSA_SHA1)) {
        return "1.2.840.10040.4.1";
    } else if (algURI.equalsIgnoreCase(SignatureMethod.RSA_SHA1)) {
        return "1.2.840.113549.1.1";
    } else {
        return null;
    }
    }

    /**
     * A simple KeySelectorResult containing a public key.
     */
    private static class SimpleKeySelectorResult implements
KeySelectorResult {
    private final Key key;
    SimpleKeySelectorResult(Key key) { this.key = key; }
    public Key getKey() { return key; }
    }

    /**
     * Checks if a JCA/JCE public key algorithm name is compatible with
     * the specified signature algorithm URI.
     */
    //@@@FIXME: this should also work for key types other than DSA/RSA
    private boolean algEquals(String algURI, String algName) {
        if (algName.equalsIgnoreCase("DSA") &&
            algURI.equalsIgnoreCase(SignatureMethod.DSA_SHA1)) {
            return true;
        } else if (algName.equalsIgnoreCase("RSA") &&
            algURI.equalsIgnoreCase(SignatureMethod.RSA_SHA1)) {
            return true;
        } else {
            return false;
```

```
        }
    }

    /**
     * Searches the specified keystore for a certificate that matches an
     * entry of the specified X509Data and contains a public key that is
     * compatible with the specified SignatureMethod.
     *
     * @return a KeySelectorResult containing the cert's public key if
there
     *   is a match; otherwise null
     */
    private KeySelectorResult x509DataSelect(X509Data xd, SignatureMethod
sm)
    throws KeyStoreException, KeySelectorException {
        logger.info("Processing x509Data");
    // convert signature algorithm to compatible public-key alg OID
    String algOID = getPKAlgorithmOID(sm.getAlgorithm());

    KeySelectorResult ksr = null;
        Iterator<?> xi = xd.getContent().iterator();
        while (xi.hasNext()) {
        ksr = null;
            Object o = xi.next();
        // check X509Certificate
            if (o instanceof X509Certificate) {
                logger.info("Found X509Certificate!");
                X509Certificate xcert = (X509Certificate) o;
            ksr = certSelect(xcert, sm);
        // check X509IssuerSerial
        } else if (o instanceof X509IssuerSerial) {
            X509IssuerSerial xis = (X509IssuerSerial) o;
            X509CertSelector xcs = new X509CertSelector();
            try {
                xcs.setSubjectPublicKeyAlgID(algOID);
                xcs.setSerialNumber(xis.getSerialNumber());
            xcs.setIssuer(new X500Principal
                (xis.getIssuerName()).getName());
            } catch (IOException ioe) {
            throw new KeySelectorException(ioe);
        }
        ksr = keyStoreSelect(xcs);
        // check X509SubjectName
        } else if (o instanceof String) {
            String sn = (String) o;
            X509CertSelector xcs = new X509CertSelector();
            try {
                xcs.setSubjectPublicKeyAlgID(algOID);
            xcs.setSubject(new X500Principal(sn).getName());
                    logger.info("CertSelector info: \n AlgID: " +
xcs.getSubjectPublicKeyAlgID() + "\n SubjectName: " +
xcs.getSubjectAsString());
        } catch (IOException ioe) {
```

```
            throw new KeySelectorException(ioe);
        }
        ksr = keyStoreSelect(xcs);
        // check X509SKI
        } else if (o instanceof byte[]) {
            byte[] ski = (byte[]) o;
            X509CertSelector xcs = new X509CertSelector();
            try {
                xcs.setSubjectPublicKeyAlgID(algOID);
        } catch (IOException ioe) {
            throw new KeySelectorException(ioe);
        }
        // DER-encode ski - required by X509CertSelector
        byte[] encodedSki = new byte[ski.length+2];
        encodedSki[0] = 0x04; // OCTET STRING tag value
        encodedSki[1] = (byte) ski.length; // length
        System.arraycopy(ski, 0, encodedSki, 2, ski.length);
        xcs.setSubjectKeyIdentifier(encodedSki);
        ksr = keyStoreSelect(xcs);
        // check X509CRL
        // not supported: should use CertPath API
        } else {
            // skip all other entries
            continue;
        }
        if (ksr != null) {
        return ksr;
        }
    }
```

```
    return null;
    }
}
```

RenewCertTest: unit test class. Takes the code from Massi's PoC for validating the signature.

RenewCertTest

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package org.openncp.renewcert.test;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Iterator;
import java.util.logging.Level;
import javax.xml.crypto.dsig.Reference;
import javax.xml.crypto.dsig.XMLSignature;
import javax.xml.crypto.dsig.XMLSignatureFactory;
import javax.xml.crypto.dsig.dom.DOMValidateContext;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import sun.misc.IOUtils;

/**
 *
 * @author jgoncalves
 */
public class RenewCertTest {

    private static final Logger logger =
```

```
Logger.getLogger(RenewCertTest.class);

    public RenewCertTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

//    @Test
    public void testSuccessfulVerifyEnvelopedSignature() {
        try {
            System.out.println("verifyEnvelopedSignature");
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            dbf.setNamespaceAware(true);
            Document signedDoc;
            signedDoc =
dbf.newDocumentBuilder().parse(ClassLoader.getSystemResourceAsStream("Cons
ent_Service_Discard_PT.xml"));
            SignatureManager instance = new SignatureManager(new
NSTestKeyStoreManager());
            instance.verifyEnvelopedSignature(signedDoc);
        }
        catch (SMgrException ex) {

java.util.logging.Logger.getLogger(SignatureManager.class.getName()).log(L
evel.SEVERE, null, ex);
            fail(ex.getMessage());
        }        catch (ParserConfigurationException ex) {

java.util.logging.Logger.getLogger(SignatureManager.class.getName()).log(L
evel.SEVERE, null, ex);
            fail(ex.getMessage());
        }        catch (SAXException ex) {

java.util.logging.Logger.getLogger(SignatureManager.class.getName()).log(L
evel.SEVERE, null, ex);
            fail(ex.getMessage());
        }
        catch (IOException ex) {
```

```
java.util.logging.Logger.getLogger(SignatureManager.class.getName()).log(L
evel.SEVERE, null, ex);
                        fail(ex.getMessage());
        }
    }


     @Test
     public void test() throws SAXException, IOException,
ParserConfigurationException, Exception {
         String file = "Consent_Service_Discard_PT.xml";
         validateXml(file, "", "root");
     }

     private static void validateSignature(final Document smpRecord,
Element xtPointer) throws Exception {
        logger.info("Starting signature validation...");
        // Find Signature element.
        XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");

        // NodeList elements = smpRecord.getElementsByTagNameNS(ns,
        // "Extension");
        // Node n = elements.item(0);
        // Element xtPointer = (Element) n;


//      NodeList nl = xtPointer.getElementsByTagNameNS(XMLSignature.XMLNS,
"Signature");

        NodeList nl = xtPointer.getChildNodes();
        if (nl.getLength()==0) {
            throw new Exception("Unable to find child nodes on the
element");
        }


        int size = nl.getLength();
        Element signatureel = null;
        for (int i=0; i<size; i++) {
            Node n = nl.item(i);
            if (n.getNodeType() == Node.ELEMENT_NODE) {
                Element el = (Element)n;
                if (el.getLocalName().equals("Signature") &&
el.getNamespaceURI().equals("http://www.w3.org/2000/09/xmldsig#") ) {
                    signatureel = el;
                }
            }
        }

        if (signatureel == null) {
            throw new Exception("Unable to get the signature");
        }
```

```
        // Create a DOMValidateContext and specify a KeySelector
        // and document context.
//        CertificateValidator cv = new CertificateValidator(new
NSTestKeyStoreManager().getTrustStore());
//        DOMValidateContext valContext = new DOMValidateContext(cv,
signatureel);
        DOMValidateContext valContext = new DOMValidateContext(new
X509KeySelector(new NSTestKeyStoreManager().getTrustStore()), signatureel);

        logger.info("Created DOMValidateContext from KeySelector.");
        valContext.setProperty("javax.xml.crypto.dsig.cacheReference",
Boolean.TRUE);

        // Unmarshal the XMLSignature.
        XMLSignature signature = fac.unmarshalXMLSignature(valContext);
        // Validate the XMLSignature.
        logger.info("Going to validate signature now!");
        boolean coreValidity = signature.validate(valContext);

        Iterator i = signature.getSignedInfo().getReferences().iterator();
        for (int j = 0; i.hasNext(); j++) {
            logger.debug("Iterating: " + j);
            final Reference ref = (Reference) i.next();
            InputStream is = (ref).getDigestInputStream();
            // Display the data.
            byte[] a = IOUtils.readFully(is, 0, true);
            logger.debug("Reference: " + new String(a));
        }

        // // Unmarshal the XMLSignature.
        // XMLSignature signature = fac.unmarshalXMLSignature(valContext);
        //
        // // Validate the XMLSignature.
        // boolean coreValidity = signature.validate(valContext);

        // Check core validation status.
        if (coreValidity == false) {
            logger.error("Signature failed core validation");
            boolean sv =
signature.getSignatureValue().validate(valContext);
            logger.debug("signature validation status: " + sv);
            if (sv == false) {
                // Check the validation status of each Reference.
                Iterator i1 =
signature.getSignedInfo().getReferences().iterator();
                for (int j = 0; i1.hasNext(); j++) {
                    boolean refValid = ((Reference)
i1.next()).validate(valContext);
                    logger.debug("ref[" + j + "] validity status: " +
refValid);
                }
            }
```

```
        } else {
            logger.debug("+++ Signature passed core validation +++ ");
        }

    }

    private static void validateXml(String file, String namespace, String
element) throws FileNotFoundException, ParserConfigurationException,
SAXException, IOException, Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        try (InputStream fis = ClassLoader.getSystemResourceAsStream(file))
{
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document smpRecord = db.parse(fis);
            NodeList elements = smpRecord.getElementsByTagNameNS(namespace,
element);
            Node n = elements.item(0);
            Element xtPointer = (Element) n;
            validateSignature(smpRecord, xtPointer);
        }
    }
```

```
}
```

SignatureManager: this is the class from Signature Manager component which is called by other components to validate files (which in the OpenNCP correspond to the SAML Assertions). The method provided is very similar to Massi's, pasted here just for sake of completeness of the explanation.

SignatureManager

```
/*
 *  Copyright 2010 Jerry Dimitriou <jerouris at netsmart.gr>.
 *
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 *  under the License.
 */
package org.openncp.renewcert.test;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.crypto.MarshalException;
import javax.xml.crypto.dsig.XMLSignature;
import javax.xml.crypto.dsig.XMLSignatureException;
import javax.xml.crypto.dsig.XMLSignatureFactory;
import javax.xml.crypto.dsig.dom.DOMValidateContext;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

/**
 * The NCP Signature Manager is a JAVA library for applying and verifying
 * detached digital signatures on XML documents and for applying and
verifying
 * enveloped signatures on SAML assertions and Audit Trail Messages
 *
 * @author Jerry Dimitriou <jerouris at netsmart.gr>
 */
public class SignatureManager {
```

```
    private KeyStoreManager keyManager;

    public SignatureManager() throws IOException {
        //Default constructor now defaults to the test keyStoreManager
        //TODO: Create a keystore manager to handle official Keystores;
        keyManager = new NSTestKeyStoreManager();

    }

    public SignatureManager(KeyStoreManager keyStoreManager) throws
IOException {
        //Constructor for unit testing. Sort of DI
        this.keyManager = keyStoreManager;

    }


    /**
     * Verifies the enveloped XML signature and checks that the XML
Document is
     * signed against that signature. This method returns nothing when the
     * signature is valid. When the signature is not valid though, it
throws an
     * SMgrException with the Error Message that caused the signature to
fail
     * validation.
     *
     * @param doc The XML Document that will be validated.
     * @throws SMgrException When the validation of the signature fails
     * @throws java.io.IOException
     */
    public void verifyEnvelopedSignature(Document doc) throws
SMgrException, IOException {
        try {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            dbf.setNamespaceAware(true);
            XMLSignatureFactory fac =
XMLSignatureFactory.getInstance("DOM");

            NodeList nl = doc.getElementsByTagNameNS(XMLSignature.XMLNS,
"Signature");

            if (nl.getLength() == 0) {
                throw new SMgrException("Cannot find Signature element");
            } // and document context.

            CertificateValidator cv = new
CertificateValidator(keyManager.getTrustStore());

            DOMValidateContext valContext = new DOMValidateContext(cv,
nl.item(0));
```

```
            // Unmarshal the XMLSignature.
            XMLSignature signature = fac.unmarshalXMLSignature(valContext);
            // Validate the XMLSignature.

            boolean coreValidity = signature.validate(valContext);

            if (coreValidity == false) {
                throw new SMgrException("Invalid Signature: Mathematic
check failed");
            } else {
                System.out.println("Signature passed core validation from
secman!");
            }

        } catch (XMLSignatureException ex) {

Logger.getLogger(SignatureManager.class.getName()).log(Level.SEVERE, null,
ex);
            throw new SMgrException("Signature Invalid: " +
ex.getMessage(), ex);
        } catch (MarshalException ex) {

Logger.getLogger(SignatureManager.class.getName()).log(Level.SEVERE, null,
ex);
            throw new SMgrException("Signature Invalid: " +
ex.getMessage(), ex);
        }
```

```
    }
}
```

Now, proceeding to the explanation.

In the validateSignature method of the JUnit test class, you can see that there are 2 possible ways of instantiating the DOMValidateContext: either passing the CertificateValidator selector or passing the X509 selector.

```
// Create a DOMValidateContext and specify a KeySelector
        // and document context.
//        CertificateValidator cv = new CertificateValidator(new
NSTestKeyStoreManager().getTrustStore());
//        DOMValidateContext valContext = new DOMValidateContext(cv,
signatureel);
        DOMValidateContext valContext = new DOMValidateContext(new
X509KeySelector(new NSTestKeyStoreManager().getTrustStore()), signatureel);
```

Depending on which one is passed, the behavior for obtaining the key for validation is different:

- **CertificateValidator**: if this selector is used, then the public key of the certificate contained in the signature is retrieved. Before obtaining it, that certificate is validated. As it is today in the OpenNCP, only the keyUsage and the expiration date of the certificate are checked during the validation process because the code for checking the trust chain of the certificate in the truststore is not ran (the code is commented). If you provide a wrong truststore (one that doesn't contain the leaf certificate), validation is still successful. When the code for validating the certificate path is uncommented, we can verify that the validation of the certificate chain starting with the certificate contained in the signature is made (its root CA certificate is found, thus trust is verified due to it being a trust anchor). In the case the wrong truststore is provided (one that doesn't contain the leaf certificate), this validation fails. If OpenNCP doesn't check the truststore for the certificate chain, then what's the purpose of verifying the signature of the SMP file using the public key of the renewed certificate (the one that would be stored in the truststore)? The truststore is never accessed anyway. My guess is that this is incorrect behavior and should be fixed to, at least, check the trust chain. I believe that currently, the trust chain validation is only being made by the Tomcat itself (by means of its HTTPS connector), not by the OpenNCP application itself.
  I set the CRL check to false to avoid errors due to my self-signed certificates, so this part is not relevant.
- **X509Selector**: if this selector is used, first it tries to find a matching certificate in the truststore based on the signing certificate's Subject DN (/X509Data/X509Subject) and it fails. Then, it tries to obtain the alias of the certificate from the truststore (passing the signing certificate as a parameter) and fails again to find a matching certificate. In the end, validation of the signature fails because the public key cannot be returned from the key selector. If the truststore contains exactly the same certificate that is fetched from the signature, then the second test of this selector (obtaining the alias) is successful (albeit the failure of the Subject DN test). But I cannot understand why it's failing when trying to match the signing certificate with the renewed certificate. Do notice that this selector is not used by the OpenNCP, as far as I could check.

Test results using **CertificateValidator**:

```
2016-09-16 18:01:19 INFO  RenewCertTest.validateSignature:100 - Starting
signature validation...
2016-09-16 18:01:19 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:01:19 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ca.epsos-spms.pt
2016-09-16 18:01:19 INFO  RenewCertTest.validateSignature:140 - Created
DOMValidateContext from KeySelector.
2016-09-16 18:01:19 INFO  RenewCertTest.validateSignature:146 - Going to
validate signature now!
2016-09-16 18:01:19 INFO  CertificateValidator.validateCertificate:158 -
Validating the following certificate:
[
[
  Version: V3
```

```
  Subject: CN=qaepsos.min-saude.pt, SURNAME=Cunha, GIVENNAME=Joao, O=SPMS,
C=PT
  Signature Algorithm: SHA512withRSA, OID = 1.2.840.113549.1.1.13

  Key:  Sun RSA public key, 2048 bits
  modulus:
26330813520423591907557344818618737243735713482860433884526138260447119576
23007882500756859656235936291192320565323100433817457232804849706409746264
44226975888129294253656247012570865730314001200004230802667482328571450134
46787326307627944766419643564213695243011445603585010639089545437603982374
27144565161217789795347846581722567107861633580634257156303221918228585981
19199178045869979797391171257964684223355206194398530365345331841441474910
90663475491009670162744684733042123707951911742189491428646111289439767266
03456854133492467687785156145947987497677092372829065799490093379347748503
28457789794676380612340 49
  public exponent: 65537
  Validity: [From: Tue Sep 13 16:44:02 WEST 2016,
              To: Wed Sep 13 16:44:02 WEST 2017]
  Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
  SerialNumber: [    e3b85826 28c733d6]

Certificate Extensions: 6
[1]: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

[2]: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:false
  PathLen: undefined
]

[3]: ObjectId: 2.5.29.31 Criticality=false
CRLDistributionPoints [
  [DistributionPoint:
     [URIName: http://your.cert.authority/CRL/openncp-crl.pem]
]]

[4]: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  DigitalSignature
  Non_repudiation
]

[5]: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  RFC822Name: joao.cunha@spms.min-saude.pt
]
```

```
[6]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C4 22 21 A7 EB 87 D6 EB   15 BF 23 5D 48 7E F5 24  ."!.......#]H..$
0010: E6 80 C5 29                                        ...)
]
]

]
  Algorithm: [SHA512withRSA]
  Signature:
0000: 2C 9C 98 CC 2A 12 A2 F6   FC F3 0A 9D CD 4C CB 45  ,...*........L.E
0010: 9A 6E 56 F3 53 2E B0 8C   96 38 F6 FB E6 31 17 57  .nV.S....8...1.W
0020: EA 4F F8 51 20 51 FD 8F   28 64 49 AF 99 7C 44 E8  .O.Q Q..(dI...D.
0030: 06 1E 65 19 70 02 A4 4C   3E 91 F8 CD 79 34 CC EE  ..e.p..L>...y4..
0040: F8 45 BB 95 72 11 CC E4   BD 7B DC 5D 8F F7 5A 85  .E..r......]..Z.
0050: 80 B8 32 14 40 C3 3A 32   08 D7 BC D0 78 41 18 60  ..2.@.:2....xA.`
0060: BE FC 78 C2 EC C2 7A A4   E1 72 D8 AD C8 4E 1B F4  ..x...z..r...N..
0070: 25 F9 C5 E1 30 96 9F 10   B1 14 E1 52 E2 26 71 56  %...0......R.&qV
0080: 72 76 F5 0F EB 29 31 7A   1B 3B 9A DA 98 E7 59 C1  rv...)1z.;....Y.
0090: 53 2F 6D 02 54 A9 15 EA   3E FA 09 E1 96 21 92 DA  S/m.T...>....!..
00A0: 16 FA 9D A7 3D 06 A1 6C   3B AF E7 91 72 9A 95 44  ....=..l;...r..D
00B0: CC 52 41 3C C5 2F 18 86   A7 2C C6 AE 4C A7 0C 31  .RA<./...,..L..1
00C0: 9C C4 95 47 21 DD 5C 81   A9 7C 83 CE 3D 92 A9 A8  ...G!.\.....=...
00D0: CA BC 05 15 A1 4D FC 94   D6 AE 0F 07 BD 3E E8 DC  .....M.......>..
00E0: 9A F6 96 D0 9F FA 44 9E   D2 6E 45 36 D2 3D 58 FE  ......D..nE6.=X.
00F0: EA D2 9F A4 01 30 97 EC   1D 6D 8B C1 7E 5C B6 8F  .....0...m...\..
0100: AE 95 A7 17 50 78 DC 59   98 62 99 A2 49 F3 A6 AD  ....Px.Y.b..I...
0110: 8C 07 C3 C1 61 3A CB 47   FF 10 D4 B3 8B C7 99 EB  ....a:.G........
0120: DD E1 CB 09 91 81 4E AD   7B B1 F8 94 EC D4 CD DD  ......N.........
0130: DE C4 C6 E2 14 DC 55 4D   F9 8C 1C 89 A8 BB C6 41  ......UM.......A
0140: 1D 46 FB C2 25 54 35 28   5C FD 0F 45 D6 0A 69 4E  .F..%T5(\..E..iN
0150: 95 30 77 C8 15 EF 86 F7   65 EC 94 09 9E A8 EC 40  .0w.....e......@
0160: D7 FC E4 9D 64 D6 CF C5   19 68 F9 C3 6B 43 41 A9  ....d....h..kCA.
0170: F0 9A 16 10 9D D1 EA 42   73 32 10 0C B3 F0 D8 C1  .......Bs2......
0180: 40 1F 2B 27 E8 71 E9 21   69 41 08 C4 50 6E E0 93  @.+'.q.!iA..Pn..
0190: F9 AD 1D 37 6B AD 2B 12   39 77 BF 99 87 5A EB 5B  ...7k.+.9w...Z.[
01A0: 9F CE A0 D9 BD 93 AC 6B   2C 97 F6 6C 2C 9C 72 20  .......k,..l,.r
01B0: 13 35 BC 34 89 EF 98 06   60 8C 33 8D EF D0 98 A6  .5.4....`.3.....
01C0: 92 19 C9 EB D5 12 68 E5   9E 83 1B 6E 53 7F 49 1B  ......h....nS.I.
01D0: 59 45 89 C4 CC 86 8F C7   52 9E C7 C7 92 88 93 53  YE......R......S
01E0: 89 9F 72 86 5C 6A 67 DC   9D 71 9E 1E 6D 2F 2E F4  ..r.\jg..q..m/..
01F0: D2 DE 3D C6 DF 57 E8 D6   E6 4C 20 9E 77 57 9A F9  ..=..W...L .wW..

]
Sep 16, 2016 6:01:19 PM org.openncp.renewcert.test.CertificateValidator
validateCertificate
INFO: Found CRLDP Extension. Using CRLDP
2016-09-16 18:01:19 INFO  CertificateValidator.validateCertificate:215 -
PKIXCertPathValidatorResult: [
  Trust Anchor: [
  Trusted CA cert: [
```

```
[
  Version: V3
  Subject: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha,
OU=IOP, O=SPMS, L=Porto, ST=Porto, C=PT
  Signature Algorithm: SHA256withRSA, OID = 1.2.840.113549.1.1.11

  Key:  Sun RSA public key, 4096 bits
  modulus:
890048260450988741879851089225250970253887534881451607720022576946068986671238358062690692632279185840036890791307964833669886357700817215036080902930900114407251977638317110902262193003310314771705023171654811614151922305877992599772737972407119944045001589834820943564332308020639518879784315512394849963594426474956563979606895953277008176677623734703392591876977927882303257781876808553686190918998518749039547746577768894459308697220722629422003989681145328846020348686847021414843426546928694350438647657177420226785128945299653174712777603420733615468937303635573097948205786374723447435949011827400850399031843801059433807806886348636784664699421341321711792677398646747974180556192955223593334046395879728956052050712997445683956357618421485578400767538935440285263617311398181532238190309450172034429818850697670580421570011311898852469423141076661713390303414241813603727429583469739151320312315993286303279975291773846536156304618996019382457988239622046834470081108873152569807165442133742726331294504589223131572551693742038789467046368844634456481671636127587930145026819541783180125347254041186340463071680908279777814391892064730144542996771243148875340182118579777509295299541764656673767489110764226281016669184089
  public exponent: 65537
  Validity: [From: Tue Sep 13 15:55:38 WEST 2016,
              To: Fri Sep 11 15:55:38 WEST 2026]
  Issuer: EMAILADDRESS=joao.cunha@spms.min-saude.pt, CN=joao.cunha, OU=IOP,
O=SPMS, L=Porto, ST=Porto, C=PT
  SerialNumber: [    fbb9a3d5 6e7d84a3]

Certificate Extensions: 3
[1]: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
]

[2]: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

[3]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 75 B3 E9 C2 7D EB 50 A0   BF 0E 77 40 3E 7C 2C DE  u.....P...w@>.,.
0010: D7 CF 16 5C                                        ...\
]
```

```
]

]
  Algorithm: [SHA256withRSA]
  Signature:
0000: 3E DD BC 0F BA 8F 9F 18   16 34 86 E8 BB 55 08 81  >........4...U..
0010: 6E 03 D8 93 5C 14 98 9E   B0 4D 5D 6B 93 36 32 04  n...\....M]k.62.
0020: D2 52 7D 4A 36 00 4D C1   17 D5 5F F1 80 25 1D 57  .R.J6.M..._..%.W
0030: 93 C7 79 15 F0 EB D3 61   51 2D 17 36 86 9B 7C 27  ..y....aQ-.6...'
0040: A7 CC 0F 7A D4 6E 53 80   4E 3E B9 4A 32 96 88 17  ...z.nS.N>.J2...
0050: AF B5 37 7A 5A 9C 04 FD   BE 31 F2 B4 0C AE 38 A2  ..7zZ....1....8.
0060: 9C A0 C3 5E 4D D1 50 84   36 DB A8 66 54 9B 98 B8  ...^M.P.6..fT...
0070: 7A EE BA 81 CA 3E DF 78   C9 63 1F B3 B7 B1 A6 21  z....>.x.c.....!
0080: C0 1A 4E AA B6 D7 42 19   07 14 6F 83 00 12 DB 5C  ..N...B...o....\
0090: 32 2A 16 B1 5E 47 6B F0   E8 49 BD D1 6F 60 88 68  2*..^Gk..I..o`.h
00A0: 5E CA 30 1B 77 CF 73 C8   04 B6 1A 63 59 F7 4E AB  ^.0.w.s....cY.N.
00B0: 27 A8 DA A8 49 9C 5B 66   05 A2 62 37 05 B8 87 39  '...I.[f..b7...9
00C0: F1 0E 9D B5 86 8D 7D CE   F3 2A 9D F0 68 7B 28 DB  ........*..h.(.
00D0: EF FF 3F 9F 34 BC 3A 50   12 5F 78 E0 7E 03 77 DE  ..?.4.:P._x...w.
00E0: B9 CD 71 53 0B FF CC A3   F2 90 01 B6 8D 54 E5 B0  ..qS.........T..
00F0: 3A 11 41 E4 29 5F 1C F8   97 F8 27 82 D6 72 7D 31  :.A.)_....'..r.1
0100: D2 95 EE F4 CB BA 96 79   22 A1 5F 70 06 D5 69 6A  .......y"._p..ij
0110: FB 6C 69 03 D9 3D FF 0C   09 65 2D F6 35 42 8C AF  .li..=...e-.5B..
0120: 91 34 93 EA C3 4A 39 28   F0 65 54 58 B6 2C 38 8D  .4...J9(.eTX.,8.
0130: 90 3A D5 66 5D 01 A2 13   EA 78 90 9D 89 0C 3E 2E  .:.f]....x....>.
0140: E2 10 82 0D 5E A7 B6 44   3E BC 05 B2 56 89 5C 7B  ....^..D>...V.\.
0150: 48 A9 24 87 87 25 A0 6A   CB A7 DF 07 EF 9A 8E 2F  H.$..%.j......./
0160: B9 82 DD FF AA B7 C9 4D   78 CC EA 19 22 18 54 79  .......Mx...".Ty
0170: 45 FB DE 4F AC 97 57 57   EC D1 07 B4 5A 8B A7 AC  E..O..WW....Z...
0180: 98 D8 12 64 CF 04 A8 48   D9 A4 95 E4 7C E7 16 19  ...d...H........
0190: B6 05 AF CF 24 8D 33 33   88 0C 83 22 01 95 62 92  ....$.33..."..b.
01A0: 66 4A 06 93 E0 04 E5 D0   94 FA 55 80 50 7A B4 DB  fJ........U.Pz..
01B0: E1 DA F8 0D E1 8A A7 A7   FA 2A 57 83 10 E6 F6 B3  .........*W.....
01C0: 58 60 B6 71 BC E2 80 1A   6E 23 50 0A 3B A8 9C 97  X`.q....n#P.;...
01D0: FC 60 65 13 DA 07 60 10   BE 12 5E 4E 5B FF 1B B4  .`e...`...^N[...
01E0: 78 03 33 2B 6F 3F 60 8A   93 E1 8D 76 E1 F2 08 21  x.3+o?`....v...!
01F0: 52 7C D5 A1 89 71 6C 73   50 88 28 D1 4F 6F 56 07  R....qlsP.(.OoV.

]

  Policy Tree: null
  Subject Public Key: Sun RSA public key, 2048 bits
  modulus:
26330813520423591907557344818618737243735713482860433884526138260447119576
23007882500756859656235936291192320565323100433817457232804849706409746264
44226975888129294253656247012570865730314001200004230802667482328571450134
46787326307627944766419643564213695243011445603585010639089545437603982374
27144565161217789795347846581722567107861633580634257156303221918228585981
19199178045869979797391171257964684223355206194398530365345331841441474910
90663475491009670162744684733042123707951911742189491428646111289439767266
03456854133492467687785156145947987497677092372829065799490093379347748503
28457789794676380612340 49
  public exponent: 65537
```

```
]
2016-09-16 18:01:19 DEBUG RenewCertTest.validateSignature:151 - Iterating:
0
2016-09-16 18:01:19 DEBUG RenewCertTest.validateSignature:156 - Reference:
```

```
2016-09-16 18:01:19 DEBUG RenewCertTest.validateSignature:179 - +++
Signature passed core validation +++
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.291 sec
```

Test results using **X509Selector** and the truststore containing the renewed certificate:

```
2016-09-16 18:02:47 INFO  RenewCertTest.validateSignature:100 - Starting
signature validation...
2016-09-16 18:02:47 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:02:47 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ca.epsos-spms.pt
2016-09-16 18:02:47 INFO  X509KeySelector.<init>:78 - Alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:02:47 INFO  X509KeySelector.<init>:78 - Alias:
ppt.ca.epsos-spms.pt
2016-09-16 18:02:47 INFO  RenewCertTest.validateSignature:140 - Created
DOMValidateContext from KeySelector.
2016-09-16 18:02:47 INFO  RenewCertTest.validateSignature:146 - Going to
validate signature now!
2016-09-16 18:02:47 INFO  X509KeySelector.select:143 - Found X509Data!
2016-09-16 18:02:47 INFO  X509KeySelector.x509DataSelect:302 - Processing
x509Data
2016-09-16 18:02:47 INFO  X509KeySelector.x509DataSelect:336 - CertSelector
info:
 AlgID: 1.2.840.113549.1.1
 SubjectName:
CN=qaepsos.min-saude.pt,2.5.4.4=#130543756e6861,2.5.4.42=#13044a6f616f,O=S
PMS,C=PT
2016-09-16 18:02:47 INFO  X509KeySelector.keyStoreSelect:206 - Searching
keystore with the CertSelector
2016-09-16 18:02:47 INFO  X509KeySelector.keyStoreSelect:210 - Trying to
match cert with the one from truststore with alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:02:47 INFO  X509KeySelector.keyStoreSelect:210 - Trying to
match cert with the one from truststore with alias: ppt.ca.epsos-spms.pt
2016-09-16 18:02:47 INFO  X509KeySelector.keyStoreSelect:217 - Didn't find
matching certificate!
2016-09-16 18:02:47 INFO  X509KeySelector.x509DataSelect:313 - Found
X509Certificate!
2016-09-16 18:02:47 INFO  X509KeySelector.certSelect:248 - Couldn't obtain
certificate from truststore
Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 0.219 sec
<<< FAILURE!

Results :

Tests in error:
  test(org.openncp.renewcert.test.RenewCertTest): the keyselector did not
find a validation key
```

Test results using **X509Selector** and the truststore containing the original (signing) certificate:

```
2016-09-16 18:03:55 INFO  RenewCertTest.validateSignature:100 - Starting
signature validation...
2016-09-16 18:03:55 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:03:55 INFO  NSTestKeyStoreManager.<init>:70 - Alias:
ppt.ca.epsos.spms.pt
2016-09-16 18:03:55 INFO  X509KeySelector.<init>:78 - Alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:03:55 INFO  X509KeySelector.<init>:78 - Alias:
ppt.ca.epsos.spms.pt
2016-09-16 18:03:55 INFO  RenewCertTest.validateSignature:140 - Created
DOMValidateContext from KeySelector.
2016-09-16 18:03:55 INFO  RenewCertTest.validateSignature:146 - Going to
validate signature now!
2016-09-16 18:03:55 INFO  X509KeySelector.select:143 - Found X509Data!
2016-09-16 18:03:55 INFO  X509KeySelector.x509DataSelect:302 - Processing
x509Data
2016-09-16 18:03:55 INFO  X509KeySelector.x509DataSelect:336 - CertSelector
info:
 AlgID: 1.2.840.113549.1.1
 SubjectName:
CN=qaepsos.min-saude.pt,2.5.4.4=#130543756e6861,2.5.4.42=#13044a6f616f,O=S
PMS,C=PT
2016-09-16 18:03:55 INFO  X509KeySelector.keyStoreSelect:206 - Searching
keystore with the CertSelector
2016-09-16 18:03:55 INFO  X509KeySelector.keyStoreSelect:210 - Trying to
match cert with the one from truststore with alias:
ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:03:55 INFO  X509KeySelector.keyStoreSelect:210 - Trying to
match cert with the one from truststore with alias: ppt.ca.epsos.spms.pt
2016-09-16 18:03:55 INFO  X509KeySelector.keyStoreSelect:217 - Didn't find
matching certificate!
2016-09-16 18:03:55 INFO  X509KeySelector.x509DataSelect:313 - Found
X509Certificate!
2016-09-16 18:03:55 INFO  X509KeySelector.certSelect:241 - Cert Alias
found: ppt.ncp-signature.epsos.spms.pt
2016-09-16 18:03:55 DEBUG RenewCertTest.validateSignature:151 - Iterating:
0
2016-09-16 18:03:55 DEBUG RenewCertTest.validateSignature:156 - Reference:
2016-09-16 18:03:55 DEBUG RenewCertTest.validateSignature:179 - +++
Signature passed core validation +++
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.229 sec
```

So based on these findings, I think we can summarize the following issues **FAIL**:

1. Validation of the trust chain of the certificate is not being made against the truststore by the Security Manager (only made by Tomcat?);
2. The public key used for signature validation is being selected from the certificate contained in the signature (/X509Data/X509Certificate), not from the matching certificate contained in the truststore. Thus, what is the point in obtaining the renewed certificate from the truststore? And how is a renewed certificate used anyway? What am I missing?
3. Have the SMP/SML TF and OpenNCP TC take a look on this document and discuss. **TODO**

a. (And hopefully prove me that I'm wrong).