
HSM Proxy Developer's Guide

Version 0.5.0

Frank Cornelis

Copyright © 2013 FedICT

Abstract

This manual describes how to use the HSM Proxy JCA security provider.

1. Introduction	1
2. JCA Security Provider	1
3. Creation of PKCS#12 credential key stores	3
4. eID Card as credential key store	4
5. Security Considerations	5
A. HSM Proxy Developer's Guide License	5
B. HSM Proxy Project License	5

1. Introduction

The HSM Proxy web service is implemented according to OASIS DSS and WS-Security standards.

To ease integration in Java environments, we implemented a JCA security provider.

First of all you need to register your application via the HSM Proxy Administrator. This procedure requires you to communicate the application name, and the required key aliases. You also need to provide the administrator with your (self-signed) credential certificate. This credential is used to secure the communication between your application and the HSM Proxy web service. In [Section 3](#), “*Creation of PKCS#12 credential key stores*” we describe how to create a credential key store. In [Section 4](#), “*eID Card as credential key store*” we describe how to use your eID card as credential.

After the HSM Proxy Administrator mapped your application key aliases to the appropriate HSM key store entries, you are ready to use the HSM Proxy via the JCA security provider.

2. JCA Security Provider

An example of how to use the HSM Proxy JCA security provider is given below.

```
import java.security.Security;
import be.fedict.hsm.jca.HSMProxyProvider;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.X509Certificate;
```

```
import be.fedict.hsm.jca.HSMProxyKeyStoreParameter;
import java.security.Signature;

Security.addProvider(new HSMProxyProvider());
KeyStore keyStore = KeyStore.getInstance("HSMProxy");

PrivateKey credentialPrivateKey = ... load from credential key store ...
X509Certificate credentialCertificate = ... load from credential key store ...

HSMProxyKeyStoreParameter keyStoreParameter = new HSMProxyKeyStoreParameter(
    credentialPrivateKey, credentialCertificate,
    "http://localhost:8080/hsm-proxy-ws/dss");
//keyStoreParameter.setProxy("proxyHost", 8080);
keyStore.load(keyStoreParameter);
PrivateKey privateKey = (PrivateKey) keyStore.getKey("your-key-alias", null);

Signature signature = Signature.getInstance("SHA1withRSA");
signature.initSign(privateKey);
byte[] toBeSigned = "hello world".getBytes();
signature.update(toBeSigned);
byte[] signatureValue = signature.sign();
```

The supported signature algorithms are listed in [Table 1, “Signature Algorithms”](#).

Table 1. Signature Algorithms

Algorithm
SHA1withRSA
SHA256withRSA
SHA512withRSA

When using Maven you can include the HSM Proxy JCA security provider dependency by adding the following under `<dependencies>` within your `pom.xml` :

```
<dependency>
  <groupId>be.fedict.hsm-proxy</groupId>
  <artifactId>hsm-proxy-jca</artifactId>
  <version>0.5.0</version>
</dependency>
```

All HSM Proxy dependencies are available within the e-Contract.be Maven repository. Add the following within your `pom.xml` under `<repositories>` :

```
<repository>
  <id>e-contract</id>
```

```
<url>https://www.e-contract.be/maven2/</url>
<releases>
  <enabled>true</enabled>
</releases>
</repository>
```

3. Creation of PKCS#12 credential key stores

This section described how to create PKCS#12 key stores using OpenSSL.

Create a 1024 bit RSA key pair with default public exponent via:

```
openssl genrsa -out key.pem -F4 1024
chmod og-rw key.pem
```

Create a new self-signed certificate via:

```
openssl req -config openssl.conf -new -x509 -key key.pem -out cert.pem -verbose
-days 365
```

with the configuration file `openssl.conf` containing the following:

```
[req]
distinguished_name = req_distinguished_name
prompt = no
x509_extensions = req_x509_extensions

[req_distinguished_name]
commonName=Test Credential

[req_x509_extensions]
```

View the new X509 certificate via:

```
openssl x509 -noout -text -in cert.pem
```

Create a PKCS#12 key store via:

```
openssl pkcs12 -export -out keystore.p12 -inkey key.pem -in cert.pem -name alias
```

View the content of the PKCS#12 key store via:

```
openssl pkcs12 -info -in keystore.p12
```

Convert a certificate from PEM format to DER format via:

```
openssl x509 -in cert.pem -out cert.der -outform DER
```

View the content of the PKCS#12 via the Java `keytool` tool:

```
keytool -list -storetype PKCS12 -keystore keystore.p12
```

4. eID Card as credential key store

For testing purposes you can use your eID card as HSM Proxy credential. Via the [Commons eID](http://code.google.com/p/commons-eid/) [http://code.google.com/p/commons-eid/] project you can easily load and use your eID certificates.

First add the following dependency to your `pom.xml` under `<dependencies>` :

```
<dependency>
  <groupId>be.fedict.commons-eid</groupId>
  <artifactId>commons-eid-jca</artifactId>
  <version>0.4.0</version>
</dependency>
```

Loading the eID authentication certificate and private key can be done via:

```
import java.security.Security;
import be.fedict.commons.eid.jca.BeIDProvider;
import java.security.KeyStore;
import java.security.cert.X509Certificate;
import java.security.PrivateKey;

Security.addProvider(new BeIDProvider());
KeyStore keyStore = KeyStore.getInstance("BeID");
keyStore.load(null);
X509Certificate authnCert = (X509Certificate)
    keyStore.getCertificate("Authentication");
PrivateKey authnPrivateKey = (PrivateKey) keyStore.getKey(
    "Authentication", null);
```

5. Security Considerations

The HSM Proxy web service SHOULD be invoked over SSL. For the moment the web service SOAP response messages are not signed at the WS-Security level. Hence unilateral SSL is the only way to safely authenticate the HSM Proxy instance.

A. HSM Proxy Developer's Guide License



This document has been released under the [Creative Commons 3.0](http://creativecommons.org/licenses/by-nc-nd/3.0/) [http://creativecommons.org/licenses/by-nc-nd/3.0/] license.

B. HSM Proxy Project License

The HSM Proxy Project source code has been released under the GNU LGPL version 3.0.

This is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License version 3.0 as published by the Free Software Foundation.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, see <http://www.gnu.org/licenses/>.

