Gestion des certifications

Baptiste Dolbeau, Florian Guilbert

1er mars 2013

Table des matières

Le projet est livré avec une PKI permettant à chaque composant de communiquer de manière sécurisé. Il est possible d'utiliser vos propres certificats. Nous présentons dans ce document la marche à suivre permettant d'adapter nos outils pour utiliser vos certificats.

Pour la suite, et pour faciliter l'établissement de la PKI, nous supposerons que tous les certifications que nous allons utilisé sont sujettes à la même autorité. Et que vous disposez déjà d'un certificat racine, dans les exemples ci-dessous, le certificat racine sera `ca.pem` et il y aura un certificat intermédiaire `cassl.pem`.

1 SoftCard

Ce composant a besoin d'un certificat serveur que nous créons avec les commandes suivantes :

```
$ openssl genrsa -out softCardServer.key -des3 2048
$ openssl req -new -key softCardServer.key -out softCardServer.csr \
    -config ./openssl.cnf
$ openssl ca -config ./openssl.cnf -name CA_ssl_default -extensions \
    SERVER_RSA_SSL -infiles cassl/softCardServer.csr
```

Qui nous permettent d'obtenir un certificat pour le SoftCard que nous appellerons dans la suite `softCardServer.pem`.

Pour pouvoir utiliser ce certificat au sein de l'application java, nous allons créer un PKCS#12 que nous convertissons en *Java KeyStore* :

```
$ openssl pkcs12 -export -inkey softCardServer.key -in softCardServer.pem \
    -out softCardServer.p12 -name "SoftCard Server Certificate "
$ keytool -importkeystore -deststorepass motDePasse -destkeypass \
    motDePass -destkeystore softCardServer.jks -srckeystore \
    softCardServer.p12 -srcstoretype PKCS12 -srcstorepass motDePasse \
    -alias "Softcard Server Certificate"
```

Pour finir, il nous faut un *package* contenant les certificats pour lesquels l'application aura confiance, ce *package* est un *trustStore* contenant des certificats au formats *DER*.

```
$ openssl x509 -outform der -in facecryptClient.pem -out facecryptClient.der
$ keytool -import -file facecryptClient.der -keystore \
    truststoreSoftCard.jks -storepass motDePasse
```

Il faut ensuite placer les fichiers .jks dans le répertoire `cert` de SoftCard. Au cas où vos *passphrases* sont différents de « lolilol » il faut modifier le code source du fichier `SoftCardServer` puis recompiler l'application.

2 FaceCrypt

La démarche est très semblable à celle pour SoftCard sauf qu'il faut créer deux certificats pour FaceCrypt un client servant pour sa communication avec SoftCard et l'autre serveur pour le dialogue avec l'extension.

Les commandes :

```
$ openssl genrsa -out facecryptClient.key -des3 2048
$ openssl req -new -key facecryptClient.key -out facecryptClient.csr \
    -config ./openssl.cnf
$ openssl ca -config ./openssl.cnf -name CA_ssl_default -extensions \
    SERVER_RSA_SSL -infiles facecryptClient.csr
```

et

```
$ openssl genrsa -out facecryptServer.key -des3 2048
$ openssl req -new -key facecryptServer.key -out facecryptServer.csr \
    -config ./openssl.cnf
$ openssl ca -config ./openssl.cnf -name CA_ssl_default -extensions \
    CLIENT_RSA_SSL -infiles facecryptServer.csr
```

créent les deux certificats. Pour le certificat serveur de FaceCrypt, il est impératif de préciser l'adresse IP de la machine l'hébergeant, pour un fonctionnement en local, nous mettons `127.0.0.1`.

Ensuite, comme pour SoftCard, il faut créer une enveloppe PKCS#12 pour ces deux certificats et les convertir en `.jks` :

```
$ openssl pkcs12 -export -inkey facecryptClient.key -in facecryptClient.pem \
    -out facecryptClient.p12 -name "FaceCrypt Client Certificate"
$ keytool -importkeystore -deststorepass motDePasse -destkeypass \
    motDePass -destkeystore facecryptClient.jks -srckeystore \
    facecryptClient.p12 -srcstoretype PKCS12 -srcstorepass motDePasse \
    -alias "FaceCrypt Client Certificate"
$ openssl pkcs12 -export -inkey facecryptServer.key -in facecryptServer.pem \
    -out facecryptServer.p12 -name "FaceCrypt Server Certificate"
$ keytool -importkeystore -deststorepass motDePasse -destkeypass \
    motDePass -destkeystore facecryptServer.jks -srckeystore \
    facecryptServer.p12 -srcstoretype PKCS12 -srcstorepass motDePasse \
    -alias "FaceCrypt Server Certificate"
```

Une fois cette étape terminée, il faut maintenant créer un *trustStore* pour indiquer les certicats dont FaceCrypt a confiance.

```
$ openssl x509 -outform der -in softCardServer.pem -out softCardServer.der
$ keytool -import -file softCardServer.der -keystore \
    truststoreSoftCard.jks -storepass motDePasse
$ openssl x509 -outform der -in extensionClient.pem -out extensionClient.der
$ keytool -import -file extensionClient.der -keystore \
    truststoreFacecrypt.jks -storepass motDePasse
```

Il faut ensuite placer les fichiers .jks dans le répertoire `cert` de FaceCrypt. Au cas où vos *passphrases* sont différents de « lolilol » il faut modifier le code source du fichier `ServerSSL` puis recompiler l'application.

3 SSNExt

Cette partie n'a besoin que d'un certificat :

```
$ openssl genrsa -out cassl/extensionClient.key -des3 2048
$ openssl req -new -key extensionClient.key -out extensionClient.csr \
    -config ./openssl.cnf
$ openssl ca -config ./openssl.cnf -name CA_ssl_default \
    -extensions CLIENT_RSA_SSL -infiles cassl/extensionClient.csr
$ openssl pkcs12 -export inkey extensionClient.key in extensionClient.pem \
        -out extensionClient.p12 -name "Extension Client Certificate"
```

Une fois cela fait, il faut importer dans le navigateur cette enveloppe PKCS#12 ainsi que les certificats `cassl.pem` et `ca.pem`, à placer respectivement dans les onglets serveurs et autorités.