Utilisation de certificats X509 en Java

Références

- Documentation de keytool
- FAQ de l'IGC du CRU

Les magasins de clés (KeyStores) de Java

Un keystore est un fichier protégé par mot de passe, qui peut contenir différentes clés et certificats.

Par défaut, le JDK est livré avec un keystore qui contient le certificats de différentes Autorités de Certification 'de confiance' ; ce keystore se trouve à : *$JAVA_HOME/jre/lib/security/cacerts*

Si on connait le mot de passe, on peut importer/exporter ou lister le contenu du keystore avec l'utilitaire java keytool (dans /usr/java/j2sdk1.4.1_02, accessible directement en ligne de commande grâce à /etc/profile.d/java.sh et /etc/java/java.conf).

Par exemple, pour lister le contenu du magasin de certificats du JDK :

```
% keytool -list [-v] -keystore $JAVA_HOME/jre/lib/security/cacerts
Enter keystore password: [vide]
***************** WARNING WARNING WARNING *****************
* The integrity of the information stored in your keystore *
* has NOT been verified\! In order to verify its integrity, *
* you must provide your keystore password. *
***************** WARNING WARNING WARNING *****************
Keystore type: jks
Keystore provider: SUN

Your keystore contains 15 entries

thawtepersonalfreemailca, Feb 12, 1999, trustedCertEntry,
Certificate fingerprint (MD5): 1E:74:C3:86:3C:0C:35:C5:3E:C2:7F:EF:3C:AA:3C:D9
baltimorecodesigningca, May 10, 2002, trustedCertEntry,
Certificate fingerprint (MD5): 90:F5:28:49:56:D1:5D:2C:B0:53:D 4:4B:EF:6F:90:22
[...]
verisignclass2ca, Jun 29, 1998, trustedCertEntry,
Certificate fingerprint (MD5): EC:40:7D:2B:76:52:67:05:2C:EA:F2:3A:4F:65:F0:D8
```

Chaque entrée est identifiée par un alias (thawtepersonalfreemailca par exemple).

Le format de keystore propriétaire de SUN est JKS.

Un utilitaire GUI est fourni avec le jdk pour gérer les keystores : policytool.

Remarques

$JAVA_HOME/jre/lib/security/cacerts est le magasin de clé global de Java, mais keytool peut manipuler n'importe quel magasin.
Le mot de passe du magasin global est par défaut changeit.

Génération d'un certificat auto-signé RSA

```
% keytool -genkey -alias CAS -keyalg RSA -dname "CN=cas.univ-xxx.fr/Email=reseau@univ-xxx.fr,O=01234567W,C=FR" -
keypass KeyPass -storepass StorePass -keystore CAS.keystore
```

Cela génère une paire de clé dans un certificat X509 auto-signé (CAS.keystore). On a spécifié un mot de passe pour protéger la clé (KeyPass), et un mot de passe pour protéger le keyStore (StorePass).

Si on ne précise pas le paramètre dname, les différents paramètres sont demandées par keytool de manière interactive.

On peut vérifier ce certificat :

```
% keytool -list [-v] -keystore CAS.keystore
```

Exportation d'un certificat (d'une clé publique)

On peut exporter le certificat vers la console :

```
% keytool -export -keystore CAS.keystore -alias CAS -storepass StorePass -rfc
```

Pour exporter la clé publique du certificat précédemment généré et stocké dans le magasin CAS.keystore :

```
% keytool -export -keystore CAS.keystore -alias CAS -storepass StorePass -file CAS.bin.export
```

Le fichier CAS.bin.export contient ainsi le certificat contenant la clé publique générée. C'est un fichier qu'on peut communiquer au monde entier.

Contrôle :

```
% keytool -printcert -file CAS.bin.export
```

Importation d'un certificat (d'une clé publique)

Dans l'exemple suivant, on va importer dans le magasin CAS.keystore le certificat public d'un serveur avec lequel on désire communiquer (ici uportal.bin. export par exemple).

```
% keytool -import -alias uportal -file uportal.bin.export -keystore CAS.keystore -storepass StorePass
```

Contrôle :

```
% keytool -list -v -keystore CAS.keystore
```

Signer les certificats par une AC locale

Dans ce cas, ce ne sont plus des certificats autosignés. On va utiliser openSSL. On suppose que le certificat de l'autorité de certification est auto-signé.

On génère la paire de clés non signée comme auparavant, avec keytool. Cette fois-ci, one ne précise pas l'option -dname, la création sera interactive :

```
% keytool -genkey -alias CAS -keyalg RSA -storepass PassKeystore -keystore CAS2.keystore
```

On génère une requête de certificat :

```
% keytool -certreq -alias CAS -storepass PassKeystore -keystore CAS2.keystore -file CAS2.csr
```

On signe cette requête de certification avec l'autorité de CA locale (on peut également utiliser sign.sh CAS2.csr ou CA.pl -sign) :

```
% openssl ca -in CAS2.csr -out CAS2.pem -keyfile ca.key
```

On concatène le certificat généré et celui de notre CA (Les 2 doivent être en format pem, donc lisibles), afin d'avoir la chaine de certification au complet. Attention, l'ordre a de l'importance :

```
% (cat CAS2.pem ; echo ; cat cacert.pem) > cas-chain.pem
```

On édite à ce moment le fichier cas-chain.pem afin de supprimer tout ce qui ne serait pas entre des lignes

```
-----BEGIN CERTIFICATE-----
```

et

```
-----END CERTIFICATE-----
```

Le résultat doit être comme suit, avec un retour chariot en dernière ligne :

```
% -----BEGIN CERTIFICATE-----
MIIEBzCCAu+gAwIBAgICANcwDQYJKoZIhvcNAQEEBQAwUDELMAkGA1UEBhMCRlIx
... certificat de serveur
Qi5LxBjyiK5xCvkUge/2+oCyB4bxsikWf1sz
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIID1zCCAr+gAwIBAgIBAzANBgkqhkiG9w0BAQQFADBPMQswCQYDVQQGEwJGUjEM
... certificat de CA
6cKi0FjWaBYjBOH6/ULalf1g4P38vbUAT4A=
-----END CERTIFICATE-----
```

On peut contrôler la validité de ce fichier :

```
% keytool -printcert -file cas-chain.pem
```

Et ensuite, on importe la chaine de certification complète dans le keystore :

```
% keytool -import -alias CAS -file cas-chain.crt -trustcacerts -keystore CAS2.keystore
```

Maintenant, le fichier CAS2.keystore contient le couple de clés, le certificat public et la chaine de certification.

Faire signer un certificat par une autorité de certification externe

On procède comme dans le paragraphe précédent pour générer le bi-clé et la requête de certification :

Génération du keystore contenant le bi-clé, et de la requête de certification

```
% keytool -genkey -alias CAS -keyalg RSA -dname "CN=cas.univ-xxx.fr,O=Universite machin,C=FR" \
    -keystore CAS2.keystore -storepass truc
% keytool -certreq -alias CAS -keystore CAS2.keystore \
    -file CAS2.csr
```

Contrôle de la requête générée :

```
% openssl req -noout -text -in CAS2.csr
```

Demande du certificat auprès de l'AC

On transmet la demande de certification (CAS2.csr) à son autorité de certification, qui va la signer et vous retourner le certificat correspondant.

Différents cas peuvent alors se produire.

Le fichier recu contient toute la chaine de certification, y compris le serveur, en format DER

Appelons ce fichier cas-chain.crt

C'est le cas le plus favorable. Pour vous en assurer :

```
%keytool -printcert -file cas-chain.crt
```

S'il n'y a pas d'erreur, on peut passer à l'étape finale d'import de la chaine de certification dans le keystore.

Le fichier recu contient toute la chaine de certification, en format pem

C'est le cas, par exemple, si vous utilisez des certificats 'SCS' : choisir l'option "The certificate packaged with the signing certificates (PEM)".

Ce format est un format 'lisible'. keytool est capable de le lire, à condition de supprimer tout ce qui ne serait pas entre des lignes

```
-----BEGIN CERTIFICATE-----
```

et

```
-----END CERTIFICATE-----
```

Il faut également respecter un ordre : d'abord le certificat du serveur, puis celui de l'AC immédiatement supérieure, et ainsi de suite

Vérifier ensuite à l'aide de la même commande que précédemment.

Le fichier reçu ne contient que le certificat du serveur (cas de l'ancienne IGC du CRU)

Construire à la main le certificat contenant le certificat du serveur, et ceux de la chaine de certification, en tenant compte de l'ordre indiqué avant.

Importation du certificat et de la chaine de certification

```
% keytool -import -alias CAS -file ca-chain.crt -trustcacerts -keystore CAS2.keystore -storepass StorePass
```

Contrôle :

```
% keytool -list -v -keystore CAS2.keystore
```

Suppression d'une entrée d'un magasin

```
% keytool -delete -keystore <keystore> -alias <alias>
```

Importation d'une clé privée

Il est parfois nécessaire d'avoir une clé privée dans le keystore, notamment quand tomcat doit répondre en https (exemple : ent avec load balancing nécessitant esup.real.port.https).

keytool ne permet pas l'import d'une clé privée. Par contre, depuis Java 6, keytool/tomcat gèrent le format PKCS12. On peut donc soit donner directement un PKCS12 à tomcat, soit faire :

```
% openssl pkcs12 -export -in xxx.univ-yyy.fr.crt -inkey xxx.univ-yyy.fr.key -out xxx.univ-yyy.fr.p12 -name xxx.
univ-yyy.fr -CAfile ca.crt -chain
% keytool -importkeystore -srckeystore xxx.univ-yyy.fr.p12 -srcstoretype pkcs12 -srcstorepass changeit -
srcalias xxx.univ-yyy.fr -destkeystore CAS.keystore -deststoretype jks -deststorepass changeit
```

nb : dans le cas d'un self-signed certificate, on peut supprimer les options "-CAfile ca.crt -chain"

Correspondence entre apache mod_ssl et truststore/keystore

- keyStore : SSLCertificateFile + SSLCertificateKeyFile + SSLCertificateChainFile
- trustStore : SSLCACertificateFile

Précisions :

- si SSLCertificateChainFile n'est pas donné, apache utilise SSLCACertificateFile
- SSLCACertificateFile est utilisé avec SSLVerifyClient
- le trustStore est utilisé quand java se connecte en https
- le keyStore est utilisé par tomcat pour écouter en https