Sécurité en Java 2

La page Trail : Security Features in Java SE (Sun) propose de nombreux cours concernant tous les aspects liés à la sécurité en Java 2. Cette fiche d'exercices s'inspire fortement des exemples trouvés à partir de cette page.

1 Génération de clés et certificats

L'outil `keytool` (ligne de commandes) fourni par le JDK de Sun permet de gérer les clés et les certificats, qui sont stockés dans un *keystore*.

1. Lisez le manuel de la commande `keytool` en utilisant la commande `man` d'Unix ou en allant à la page keytool - Key and Certificate Management Tool.
2. Générez un fichier keystore contenant une paire de clés DSA et une paire de clés RSA (`keytool -genkeypair ...`).
3. Visualisez le contenu de votre keystore (`keytool -list ...`).
4. À partir de votre keystore, exportez un certificat auto-signé contenant la clé publique DSA et un certificat auto-signé contenant la clé publique RSA (`keytool -export ...`).

2 Création d'une archive `jar` contenant une application

L'outil `jar` (ligne de commandes) fourni par le JDK de Sun permet de rassembler plusieurs fichiers au sein d'une même archive.

5. Lisez la page Lesson : Packaging Programs in JAR Files et les liens Using JAR Files : The Basics et Working with Manifest Files : The Basics qu'elle contient.
6. Écrivez en Java un programme qui prend deux arguments sur la ligne de commandes : le nom d'un fichier texte et une chaîne de caractères valant soit « read » soit « write » ; si la chaîne vaut « read », le programme doit écrire la première ligne du fichier sur la sortie standard ; si la chaîne vaut « write », le programme doit demander à l'utilisateur une chaîne de caractères terminée par un saut de ligne puis écrire la chaîne dans le fichier. Le programme devra vérifier que la ligne de commandes contient bien les arguments attendus ; si la liste des arguments n'est pas correcte, le programme doit afficher un message indiquant la façon de l'invoquer.
7. Créez une archive `jar` contenant les fichiers `.class` issus de la compilation de votre programme ; visualisez le contenu du fichier manifest de votre archive.
8. Votre archive est-elle exécutable (*i.e.* une commande du style `java -jar monArchive.jar ...` marche t-elle) ? Si ce n'est pas le cas, rendez-la exécutable.

3 Signer et vérifier une archive

L'outil `jarsigner` (ligne de commandes) fourni par le JDK de Sun permet de signer et vérifier des archives `jar`.

9. Lisez la page Signing and Verifying JAR Files.

10. À partir de votre archive, créez deux archives signées : l'une en utilisant les clés DSA de votre keystore et l'autre en utilisant les clés RSA de votre keystore ; visualisez le contenu des fichiers manifest et signature (fichier en `.SF`) de vos archives signées.
11. Récupérez une archive signée par un autre étudiant. Vérifiez la validité de cette archive (`jarsigner -verify -verbose -certs ...`). Bien qu'aucune information concernant le signataire de l'archive ne soit présente dans votre keystore, que constatez-vous ? Que cela signifie t-il ?
12. Récupérez les certificats du même étudiant :
    (a) vérifiez-les en visualisant leur contenu (`keytool -printcert ...`) ;
    (b) importez-les dans votre keystore (`keytool -import ...`) ; ceci a pour effet d'ajouter à votre keystore des certificats auxquels vous accordez votre confiance ;
    (c) vérifiez à nouveau la validité de l'archive récupérée (`jarsigner -verify -verbose -certs...`) ; quelle différence y a t-il avec le résultat de la vérification de la question 11 (regardez bien) ?

4 Politiques de sécurité

Par défaut, aucun contrôle d'accès n'est effectué pour les applications qui tournent en local. Pour appliquer une politique de sécurité, il faut invoquer l'interpréteur Java avec l'option `-Djava.security.manager`, ce qui provoque l'exécution de l'application avec utilisation du gestionnaire de sécurité par défaut.

13. Exécutez votre archive signée sans puis avec l'option `-Djava.security.manager`. Quelle différence constatez-vous ? Pourquoi ?
14. Les grandes lignes pour la sécurité sont configurées dans le fichier

    `<java.home>/lib/security/java.security`

    où `<java.home>` est le répertoire où est installé le JRE (Java Runtime Environment) ; visualisez le contenu de ce fichier et repérez-y les lignes donnant l'emplacement et le nom des fichiers où sont décrites la politique de sécurité pour le système et les politiques de sécurité des utilisateurs.
15. Créez votre fichier de politique de sécurité de sorte que les classes signées par vous (en utilisant la clé DSA ou la clé RSA créée à la question 2) aient le droit de lire les fichiers placés dans toute l'arborescence de votre répertoire `home`.
16. Vérifiez que tout fonctionne correctement : lancez votre archive signée avec l'option « read » puis avec l'option « write », ensuite faites la même chose avec l'archive signée que vous avez récupérée à la question 11.
17. Récupérez une nouvelle archive signée auprès d'un autre étudiant. Faites ce qu'il faut pour que l'exécution avec l'option « read » de cette archive fonctionne bien.

5 API de Java 2 pour la sécurité

Dans les exercices qui suivent, on utilisera l'API « Security » du JDK, disponible *via* la commande :

```
import java.security.*;
```

5.1 Génération d'une signature digitale

L'objectif de cet exercice est d'écrire un programme Java qui génère une paire de clés DSA ainsi qu'une signature digitale pour un fichier dont le nom est donné sur la ligne de commandes. Le programme devra également exporter la clé publique et la signature digitale dans des fichiers. On utilisera le squelette de code donné à la figure 1.

```
import java.io.*;        // méthodes pour lectures/écritures fichiers
import java.security.*;  // méthodes pour signatures digitales

class GenererSignature {

    public static void main(String[] args) {

        /* Génération d'une signature DSA */

        if (args.length != 1) {
            System.out.println("Usage: GenererSignature nomDuFichierASigner");
        }
        else try {

        // ... à compléter ...

        } catch (Exception e) {
            System.err.println("Exception " + e.toString());
        }
    }
}
```

FIGURE 1 – Squelette de la classe `GenererSignature`

18. La classe `KeyPairGenerator` de l'API permet de générer une paire de clés. Lisez la documentation et les exemples concernant cette classe à partir de la page JCA Reference Guide. Complétez le squelette de la figure 1 en écrivant le code nécessaire pour générer une paire de clés DSA de longueur 1024 bits.
19. Créez un objet de la classe `Signature` et initialisez-le en mode « signature ». Soit `dsa` une référence à cet objet.
20. Générez une signature du fichier dont le nom est fourni sur la ligne de commandes. Il faudra d'abord fournir à l'objet de la classe signature toutes les données du fichier :

    ```
    FileInputStream fis = new FileInputStream(args[0]);
    BufferedInputStream bufin = new BufferedInputStream(fis);
    byte[] buffer = new byte[1024];
    int len;
    while (bufin.available() != 0) {
        len = bufin.read(buffer);
        dsa.update(buffer, 0, len);
    };
    bufin.close();
    ```

    À la fin de cette étape, la signature doit se trouver dans un tableau de `byte` (résultat de la méthode `sign` de la classe `Signature`).
21. Sauvegardez la signature dans un fichier nommé `sig`.
22. Sauvegardez la clé publique générée à la question 18 dans un fichier nommé `clepub`. La méthode `getPublic()` renvoie la partie publique d'une paire de clés. La méthode `getEncoded()` renvoie la représentation d'une clé sous la forme d'un tableau de `byte`. C'est ce tableau qui sera sauvegardé dans le fichier (tout comme à la question 21).
23. Compilez et testez votre programme.

```
import java.io.*;
import java.security.*;
import java.security.spec.*;

class VerifierSignature {

    public static void main(String[] args) {

        /* Vérification d'une signature DSA  */

        if (args.length != 3) {
            System.out.println("Usage: VerifierSignature " +
                               "fichierSigne fichierClePublique " +
                               "fichierSignature");
        }
        else try {

        // ... à compléter ...

        } catch (Exception e) {
            System.err.println("Caught Exception " + e.toString());
        }
    }
}
```

FIGURE 2 – Squelette de la classe `VerifierSignature`

5.2 Vérification d'une signature digitale

L'objectif de cet exercice est d'écrire un programme Java prenant en entrée (sur la ligne de commandes) le nom d'un fichier signé, une clé publique et une signature et qui vérifie l'authenticité de la signature. On utilisera le squelette de code donné à la figure 2.

24. Ajoutez le code suivant au squelette pour lire le fichier contenant la clé publique et stocker cette clé dans un tableau de `byte` :

    ```
    FileInputStream keyfis = new FileInputStream(args[1]);
    byte[] encKey = new byte[keyfis.available()];
    keyfis.read(encKey);
    keyfis.close();
    ```

    Créez un objet de la classe `PublicKey` à partir du tableau `encKey` (lisez les exemples disponibles depuis la page The Signature Class).

25. Créez un objet de la classe `Signature`. Initialisez-le en mode « vérification » en utilisant la clé publique récupérée à la question précédente. Ajoutez un code similaire à celui de la question 20 pour fournir à cet objet les données contenues dans le fichier signé.

26. Ajoutez un code similaire à celui de la question 24 pour lire le fichier contenant la signature et stocker cette signature dans un tableau de `byte`.

27. Ajoutez le code effectuant la vérification de la signature.

28. Compilez et testez votre programme.