TD : CRYPTOGRAPHIE EN JAVA ET EN JAVACARD

Concepteur : Didier DONSEZ
But : Ce TD vise à vous familiariser avec le JCE et l'utilisant dans un dialogue avec un JavaCard.
Durée : 4 heures (gérez votre temps !)
Conseil : Des nombreuses solutions à ce TD se trouvent dans les exemples !!!
Logiciels : JCE ABA, JCE Cryptix
Documents http://www-adele.imag.fr/~donsez/cours/jce.pdf

INSTALLATION DU JCE (JAVA CRYPTOGRAPHY EXTENSION)

En suivant les indications données dans le cours, installez les implantations ABA et Cryptix du JCE. Compilez et testez quelques classes de test de ABA et de Cryptix.

RESUME SECURISE (MESSAGE DIGEST)

Ecrivez un programme `MD` qui permet de produire le résumé d'un fichier (`-digest`) et qui permet de vérifier le résumé d'un fichier (`-check`).

- `MD -digest cesar.txt cesar.md5`
- `MD -check cesar.txt cesar.md5`

La seule option `-algo` donne l'algorithme de résumé (SHA1,SHA2,MD5). L'algorithme MD5 est utilisé par défaut.

- `MD -digest cesar.txt cesar.sha -algo SHA`

Si un - est passé à la place du nom du fichier, l'entrée System.in ou la sortie System.out seront utilisés.
Testez votre programme md -check sur le fichier monfic.txt que vous aurez trafiqué !

CHIFFRAGE ET DECHIFFRAGE A CLE SECRETE

Ecrivez deux programmes `Encrypt` et `Decrypt` qui chiffre et déchiffre un fichier avec une clé fabriquée à partir du mot de passe passé en paramètre.

- `Encrypt -key MonMotDePasse cesar.txt cesar.crypt`
- `Decrypt -key MonMotDePasse cesar.crypt cesar2.txt`

La seule option `-algo` donne l'algorithme de PBE.
Testez vos programmes en modifiant le fichier intermédiaire, en utilisant des mots de passe différents, ou en vous trompant sur l'algorithme (-algo).

MANIPULATION D'UN KEYSTORE

Après avoir consulter la documentation concernant l'outil KeyTool du JDK , créez plusieurs alias (Un par élément du binôme) que vous protégerez avec des clés distinctes. Vous utiliserez des DN de la forme suivante : `"CN=Didier DONSEZ, O=IMAG/LSR, OU=UJF, L=Grenoble, S=Isere, C=FR"`
Testez les différentes commandes du keytool (listez les alias, exportez un certificat, importez les certificats de vos voisins de salle …).
Listez également les autorités de certification (CA) configurées sur votre JDK.

- `keytool –list –keystore %JAVA_HOME%\lib\security\cacerts –storepass changeit`

SIGNATURE DE FICHIER

Compilez et utilisez la classe Signer qui est passée en annexe pour signer un fichier avec la clé privée de votre alias du keystore et pour vérifier celle ci avec votre clé publique.
Que se passe t'il si vous trafiquez le fichier, la signature ou si vous vous trompez d'alias.

SIGNATURE DU JAR

Utilisez l'outil JarSigner du JDK pour signer/vérifier une archive Java avec l'alias de votre KeyStore.
Vous pourrez

1. créer l'archive cryptotools.jar avec les fichiers MD.class, Encrypt.class, Decrypt.class.
   - `jar cvf cryptotools.jar MD.class Encrypt.class Decrypt.class`
2. lister l'archive cryptotools.jar.
   - `jar tvf cryptotools.jar`
3. signer l'archive avec votre alias
4. lister l'archive scryptotools.jar.
5. vérifier les signatures de l'archive
6. ajouter un fichier Signer.class
   - `jar uvf scryptotools.jar Signer.class`
7. vérifier les signatures de l'archive
8. remplacer le MD.class par un MD.class issu de la recompilation de MD.java modifié
9. vérifier les signatures de l'archive

JCE ET PROVIDER

Ecrivez un Provider JCE qui fournit un seul algorithme de chiffrement/déchiffrement à clé secrète : CESAR.
Vous pourrez suivre les indications données dans
%JAVA_HOME%\docs\guide\security\HowToImplAProvider.html et utiliser la classe
cryptix.jce.provider.cipher.Null pour écrire votre CypherSpi.

Testez ce provider avec vos deux programmes `Encrypt` et `Decrypt` que vous avez écrit. Essayez aussi avec l'alias CAESAR pour l'algorithme.

CERTIFICAT SIGNE

Récupérez un certificat (gratuit) auprès d'une autorité de certification (CA) VeriSign, Entrust, GlobalSign …
Intégrez le à votre navigateur et votre courrieleur (mailer).
Echangez vous des mails signés **et non chiffrés**[1].
Déplacez les mails dans une mailbox. Editez cette mailbox et changez le contenu des messages. Constatez dans le MUA que la signature est corrompue !
Exportez ce certificat (DER ou PKCS#7) depuis la base de certificat de votre navigateur puis intégrer le dans votre Keystore.
Exportez la pair de clés de ce certificat (PKCS#12) depuis la base de certificat de votre navigateur puis visualisez cette paire avec keytool (et un provider JCE adéquat : celui de JSSE).

JSSE (JAVA SECURE SOCKET EXTENSION)

Téléchargez et installez JSSE (http://www.javasoft.com/products/jsse/).
Testez les exemples présentés dans l'article «Secure Your Sockets with JSSE » http://www.onjava.com/lpt/a//onjava/2001/05/03/java_security.html) et dans les exemples de JSSE.

JCE ET CARTE A PUCE

Ecrivez un client simple OCF qui vérifie le chiffrage produit par l'applet JavaCard développé lors du TP JavaCard. Le client envoie un tableau d'octets (`byte`) généré aléatoirement avec une instance de `SecureRandom`. (Basez vous sur vos 2 programmes `Encrypt` et `Decrypt`,)

[1] ATTENTION il est aussi possible de chiffrer ses mails avec le certificat de votre correspondant et avec votre mailer (NS Messenger, MS Outlook, Eudora).
CEPENDANT le chiffrage est encore réglementé en France et une autorisation doit être demandée auprès du SCSSI si la clé de session dépasse 40 bits !
Pour information, un rapport du Parlement Européen souligne la nécessité pour les entreprises européens de protéger leurs échanges contre des systèmes d'espionnage (économiques et technologiques) hérités de la Guerre Froide (ECHELON, …).

CARTE A PUCE PKI "LIGHT"

Développez une applet JavaCard Authentication.class qui comporte 3 commandes :

- une première qui retourne le nom de l'alias dont la clé privée RSA est stockée dans la carte
- une seconde qui valide un PIN code
- une troisième qui retourne une signature d'une chaîne d'octets (`byte`) passée en paramêtre avec la clé privée

Développez un client OCF qui authentifie le porteur de la carte et la carte avec les alias stockés dans le keystore passé en paramètre (basez vous sur Hancock).

Remarque : ce dialogue est similaire à celui utilisé par SmartSession :

- *http://www.gemplus.fr/developers/technologies/smartsession/index.htm*
- *http://www.gemplus.fr/developers/technologies/smartsession/html/7.html*

ANNEXE : SIGNER

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

public class Signer {
    private static final int MODE_SIGN=0;
    private static final int MODE_VERIFY=1;

    private static void printUsage()throws Exception {
       System.out.println(
           "Usage: Signer -sign keystore storepass keypass alias messagefile signaturefile");
       System.out.println(
           "Usage: Signer -verify keystore storepass alias messagefile signaturefile");
    }
  public static void main(String[] args) throws Exception {

    String options = args[0];
    String keystorefile = args[1];
    String storepass = args[2];
    String keypass = null;
    String alias = null;
    String messagefile = null;
    String signaturefile = null;
    int mode;

    if (args.length == 7 && options.equals("-sign")) {
       keypass = args[3];
       alias = args[4];
       messagefile = args[5];
       signaturefile = args[6];
       mode=MODE_SIGN;
    } else  if (args.length == 6 && options.equals("-verify")) {
       alias = args[3];
       messagefile = args[4];
       signaturefile = args[5];
       mode=MODE_VERIFY;
    } else {
       printUsage();
       return;
    }

    Signature signature = Signature.getInstance("DSA");

    KeyStore keystore = KeyStore.getInstance("JKS");
    keystore.load(new FileInputStream(keystorefile), storepass.toCharArray());

    if (mode==MODE_SIGN)
      signature.initSign((PrivateKey)keystore.getKey(alias, keypass.toCharArray()));
    else
      signature.initVerify(keystore.getCertificate(alias).getPublicKey());

    FileInputStream in = new FileInputStream(messagefile);
    byte[] buffer = new byte[8192];
    int length;
    while ((length = in.read(buffer)) != -1)
      signature.update(buffer, 0, length);
    in.close();

    if (mode==MODE_SIGN) {
      FileOutputStream out = new FileOutputStream(signaturefile);
      byte[] sig = signature.sign();
      out.write(sig);
      out.close();
    } else {
      FileInputStream sigIn = new FileInputStream(signaturefile);
      byte[] sigToVerify = new byte[sigIn.available()];
      sigIn.read(sigToVerify);
      sigIn.close();
      if (signature.verify(sigToVerify))
        System.out.println("The signature is correct.");
      else
        System.out.println("signature/message is/are CORRUPTED !!!");
    }
  }
}
```