SSL/TLS

Consignes

Le TP doit être réalisé en Java. A l'issue des 2 séances de TP un rapport devra être rendu avec votre code. Le rapport, d'environ 5 pages, expliquera votre code et présentera vos résultats. Une attention particulière sera portée à la qualité du code (structure, commentaires).

Le rapport et le code (dossier *src*) seront placés dans une archive nommée MI44_NOM_Prenom.zip et envoyée à l'adresse alexandre.lombard@utbm.fr

Le TP peut être réalisé et rendu en binôme.

Introduction

La plupart des langages de programmation disposent de bibliothèques standards permettant d'utiliser les fonctions de cryptographie les plus communes.
En C par exemple, nous avons la bibliothèque OpenSSL. En Java les classes permettant d'utiliser ces fonctions sont regroupées dans les packages java.security et javax.crypto.

Ces bibliothèques permettent :

- La création de clés RSA, DSA (signature)
- La création de certificats X509
- Le calcul d'empreintes (MD5, SHA, RIPEMD160, …)
- Le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish, …)
- La réalisation de tests de clients et serveurs SSL/TLS
- La signature et le chiffrement de courriers (S/MIME)

Dans le cadre de ce TP nous utiliserons les fonctionnalitées cryptographiques proposées en Java afin de chiffrer des messages et de signer des fichiers.

Chiffrement AES

L'objectif de cette partie est de prendre en main l'API de chiffrement Java. La première application que nous ferons est de chiffrer symétriquement un message en utilisant AES.

En utilisant la classe *KeyGenerator*, générez une clé secrète. La commande suivante vous permettra d'obtenir une instance de la classe *KeyGenerator* pour AES :

```
KeyGenerator.getInstance("AES");
```

Utilisez ensuite les méthodes *init()* et *generateKey()* pour obtenir un objet de type *SecretKey*.

La commande suivante vous permettra d'obtenir une instance de la classe de chiffrage *Cipher* :

```
Cipher aesCipher = Cipher.getInstance("AES/CBC/NoPadding");
```

L'initialisation de la classe de chiffrage pour le chiffrement avec la clé secrète se fera ainsi :

```
IvParameterSpec ivspec = new IvParameterSpec(new byte[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0 });
aesCipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
```

Appelez la commande *doFinal()* pour chiffrer un message.

Réinitialisez ensuite la classe Cipher en mode déchiffrement afin de déchiffrer le message chiffré.

Afficher à l'écran le message initial, le message chiffré puis le message déchiffré afin de valider le fonctionnement.

Chiffrement RSA

L'intérêt est de chiffrer un document avec une clé publique que seul son créateur pourra déchiffrer.

Chiffrement avec la clé publique :

1. Préparatifs
   - Générer une paire de clés RSA 2048 bits
   - Extraire ensuite la clé publique et envoyez la à votre voisin
2. Quand vous recevez une clé publique
   - Chiffrer un fichier de votre choix avec cette clé
   - Renvoyer le fichier chiffré à votre voisin
3. Quand vous recevez un fichier chiffré avec votre clé
   - Déchiffrer le fichier

Envoi sécurisé de clé avec RSA

1. Générer une paire de clés RSA 512 bits, puis envoyer la clé publique à votre voisin.
   - Générer une clé ***key*** aléatoire de 256 bits
   - Chiffrer cette clé avec la clé publique RSA reçue (de votre voisin)
   - Envoyez-lui votre clé chiffrée.

2. Quand vous recevez la clé chiffrée de votre voisin, déchiffrez-là, puis chiffrez un fichier de votre choix avec l'algorithme AES-256-CBC et cette clé. Envoyez ce fichier chiffré à votre voisin.

Signature de fichiers

https://docs.oracle.com/javase/tutorial/security/apisign/index.html

Il n'est possible de signer que des documents de taille réduite. La parade est de calculer au préalable une empreinte et de la signer à la place d'un document de taille plus conséquente. Si cela ne suffit pas, les documents devront être également compressés.

Plusieurs fonctions de signature sont disponibles dans l'API security de Java. Entre autres :

- MD5, qui calcule des empreintes de 128 bits,
- SHA1, qui calcule des empreintes de 160 bits,
- RIPEMD160, qui calcule des empreintes de 160 bits.

Signature

Afin d'obtenir une empreinte d'un fichier avec SHA, vous commencerez par générer une paire de clé de la façon suivante :

```
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA", "SUN");
KeyPair keyPair = keyPairGenerator.generateKeyPair();
```

Ensuite vous utiliserez la classe *Signature* dont vous obtiendrez une instance de la façon suivante :

```
Signature dsa = Signature.getInstance("SHA256withDSA", "SUN");
```

Pour signer un fichier il vous faudra :

- Initialiser l'objet de la classe Signature avec la clé privée généré :

```
dsa.initSign(keyPair.getPrivate());
```

- Lire un fichier et le passer en paramètre de la méthode *update()* :

```
try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(filePath))) {
            byte[] buffer = new byte[1024];
            int len;
            while ((len = bis.read(buffer)) >= 0) {
            dsa.update(buffer, 0, len);
            }
}
```

- Obtenir la signature :

```
byte[] signature = dsa.sign();
```

Vous sauvegarderez ensuite la signature et la clé publique dans deux fichiers séparés.

Vérification de la signature

Il reste ensuite à vérifier que l'empreinte ainsi produite est la même que celle que l'on peut calculer.

Il faudra :

- Lire et initialiser la clé publique depuis le fichier préalablement généré
- Lire la signature
- Initialiser la classe *Signature* avec *initVerify()*
- Fournir le fichier à valider à l'objet de classe *Signature*
- Obtenir la vérification de la signature avec la méthode *verify()*

Suivez les instructions données au lien suivant afin de valider la signature préalablement générée :
https://docs.oracle.com/javase/tutorial/security/apisign/versig.html

Certification

L'objectif de cette partie est d'élaborer un certificat pour votre clé publique.

Créer votre paire de clés RSA de taille 2048 bits, puis établir une requête pour obtenir un certificat : le certificat contient des informations sur la clé publique du sujet ainsi que d'autres informations sur l'identité de son propriétaire. Ces informations sont demandées lors de la création de la requête.

La classe que vous utiliserez pour générer le certificat est *X509CertInfo*. Cette classe sera initialisée avec les différentes informations relatives au certificat : date de création, date de validité, propriétaire (voir classe *X500Name*), clé publique.

Vous signerez ensuite ce certificat avec la clé privée :

```
X509CertImpl cert = new X509CertImpl(info);
cert.sign(privkey, "MD5withRSA");
```

Demande de signature de certificat : Une fois que vous avez établi la requête de certificat, il vous reste à contacter une autorité de certification qui vous délivrera un certificat signé, après avoir procédé (normalement) à quelques vérifications vous concernant.

La création d'une requête de signature de certificat à partir du certificat se fait de la façon suivante :

```
PKCS10 request = new PKCS10(certificate.getPublicKey());

Signature signature = Signature.getInstance("MD5WithRSA");
signature.initSign(privateKey);
```

```
X500Name subject = new X500Name(((X509Certificate)certificate).getSubjectDN().toString());
request.encodeAndSign(subject, signature);
```

Vous stockerez ensuite le contenu de la classe request dans un fichier portant l'extension CSR. Ce fichier doit ensuite normalement être envoyé à une autorité de certification.