Programmation réseaux

TP 9
Sécurité

Avril 2007

Dans ce TP, nous allons écrire un client et un serveur simple et rajouter un mécanisme d'authentification puis pour des communications cryptées.

Le client pourra se contenter d'envoyer un message au serveur qui l'affichera sur sa sortie standard.

Exercice 1 – *Authentification par secret partagé*

Le serveur demande un nom d'utilisateur au client avant d'autoriser le dialogue. Dans la méthode d'authentification par secret partagé, le client et le serveur partagent tous les deux un secret (mot de passe). Le serveur tire au sort un *nonce* (mot aléatoire) et demande au client de calculer une somme de contrôle (algo MD5 ou SHA1...) pour la concaténation du secret et du nonce. Le client envoie cette somme de contrôle au serveur, qui peut comparer avec le résultat de son propre calcul.

Classes à utiliser :

- SecureRandom permet de créer un générateur aléatoire sûr.

```
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
byte[] random = new byte[20];
sr.nextBytes(random); // remplit le tableau de valeurs aléatoires
```

- MessageDigest implémente les algorithmes de hachage MD5 ou SHA. Exemple :

```
MessageDigest md = MessageDigest.getInstance("SHA-1");
byte[] data = {'t','o','t','o'};

byte[] digest = md.digest(data);
System.out.println(digest.length);
```

Exercice 2 – *Authentification par clé publique/privée*

Les méthodes de cryptographie par *clé asymétrique* (ou clé publique/privée) reposent sur des algorithmes « à sens unique ». Un message codé avec la clé publique ne peut être déchiffré que si l'on connaît la clé privée et vice-versa.

Implémenter une méthode d'authentification par clé publique/privée. Le client envoie son nom d'utilisateur, le serveur lui envoie un nonce (tiré au hasard). Le client signe le nonce avec sa clé privée et l'envoie au serveur, qui la décrypte grâce à la clé publique et compare avec l'original.

Classes à utiliser :

- KeyPairGenerator Génération de paires (clé publique/clé privée). Exemple :

```
KeyPairGenerator dsagenerator = KeyPairGenerator.getInstance("DSA");
dsagenerator.initialize(1024);
KeyPair dsapair = dsagenerator.generateKeyPair();
```

- KeyPair Paire de clés, avec les méthodes getPrivate() et getPublic()
- Key Une clé
- Signature

```
Signature s = Signature.getInstance("DSA"); // On utilise l'algo DSA
s.initSign(dsapair.getPrivate());       // initialisation avec la clé privée
s.update(data);                         // les données à crypter
byte[] sign = s.sign();                 // le résultat (signature)
System.out.println(sign.length);

   // De l'autre côté :
Signature v = Signature.getInstance("DSA"); // On utilise l'algo DSA
s.initVerify(dsapair.getPublic());  // initialisation de la vérification avec la clé publique
s.update(data);                         // les données à crypter
System.out.println(s.verify(sign)); // Vérification
```

- Serializable Interface (vide) à implémenter pour pouvoir sérialiser les objets

Exercice 3 – *Chiffrement avec clé partagée (symétrique)*

Le client et le serveur partagent un secret qui permet de crypter et décrypter le message.
Classes à utiliser :

- KeyGenerator

```
SecretKey sk = (KeyGenerator.getInstance("AES")).generateKey(); // Génère une clé secrète
```

- Cipher Chiffrer un message :

```
Cipher cphr1 = Cipher.getInstance("AES");
cphr1.init(Cipher.ENCRYPT_MODE, sk);
byte[] ciphertext1 = cphr1.doFinal(data);
cphr1.init(Cipher.DECRYPT_MODE, sk);
byte[] cleartext1 = cphr1.doFinal(ciphertext1);
System.out.println(new String(data));
System.out.println(new String(cleartext1));
```

- CipherInputStream Flux chiffré
- CipherOutputStream

Exercice 4 – *Chiffrement avec clé asymétrique*

Cette fois-ci le serveur envoie sa clé publique au client pour qu'il l'utilise pour crypter le message. Le serveur décrypte avec sa clé privée.
Classes à utiliser :

- KeyPairGenerator et KeyPair (voir ci-dessus)
- Cipher Exemple :

```
KeyPairGenerator rsagenerator = KeyPairGenerator.getInstance("RSA");
rsagenerator.initialize(1024);
KeyPair rsapair = rsagenerator.generateKeyPair();

Cipher cphr2 = Cipher.getInstance("RSA");
cphr2.init(Cipher.ENCRYPT_MODE, rsapair.getPublic());
byte[] ciphertext2 = cphr2.doFinal(data);
cphr2.init(Cipher.DECRYPT_MODE, rsapair.getPrivate());
byte[] cleartext2 = cphr2.doFinal(ciphertext2);
System.out.println(new String(data));
System.out.println(new String(cleartext2));
```

- CipherInputStream
- CipherOutputStream