CENTRALESUPÉLEC
Campus de Rennes
Majeure SIS

TL Programmation - Cryptographie

Remarque préliminaire : l'objectif du TL est avant tout d'utiliser une bibliothèque cryptographique pour générer et manipuler des certificats, le contexte global décrit n'étant là que pour donner du sens à cela.

I. Contexte

Un réseau domestique est formé d'un ensemble d'équipements interconnectés. Un réseau domestique évolue au gré de l'achat et de la vente des équipements, cette évolution devant se faire le plus simplement possible pour l'utilisateur. En particulier, l'insertion d'un nouvel équipement (PC, PDA, TV, mobile, lecteur DVD, ...) dans le réseau ne doit pas exiger une quelconque expertise, au plus peut-on demander à l'utilisateur de valider la phase d'insertion (par exemple, sous la forme d'une question « Insérer l'équipement DVD X452 (Oui/Non) ? »).

Du point de vue de la sécurité, protéger un réseau domestique consiste d'une part à sécuriser l'évolution du réseau, et d'autre part, à sécuriser les échanges entre les équipements constituant le réseau. Concernant la sécurité de l'évolution du réseau, il s'agit de permettre à l'utilisateur / l'administrateur du réseau domestique de contrôler les insertions et les retraits d'équipements. Concernant la sécurité des échanges entre les équipements du réseau, un prérequis est de permettre à chaque équipement de pouvoir authentifier les équipements qui appartiennent à son réseau. Dans le cadre de ce TL, nous proposons d'implémenter une solution permettant de contrôler l'évolution du réseau domestique tout en permettant aux équipements d'échanger le matériel cryptographique (i.e., les certificats) qui leur servira pour s'authentifier mutuellement. Nous décrivons brièvement notre solution.

II. Solution

La solution proposée consiste à mettre en place une infrastructure à clé publique distribuée *à la PGP* : chaque équipement se considère comme l'autorité racine de l'infrastructure à clé publique. Au gré des échanges avec les autres équipements du réseau, un équipement doit donc pouvoir maintenir l'arbre de certification de lui vers l'ensemble des autres équipements du réseau, *mais également l'arbre inverse de n'importe quel équipement vers lui-même* afin de pouvoir *prouver* qu'ils appartiennent au même réseau domestique.

Plus précisément, lors de l'évolution du réseau domestique (e.g., lors de l'insertion d'un nouvel équipement), le principe est que l'utilisateur n'a à intervenir que sur l'un des équipements appartenant déjà au réseau domestique en répondant à une question simple telle que « Insérer l'équipement DVD X452 (Oui/Non) ? ». Pour l'équipement appartenant au réseau, en tant qu'autorité racine, il s'agit donc de certifier un nouvel équipement. Pour le nouvel équipement, il s'agit donc de certifier l'équipement utilisé par l'utilisateur pour le faire rentrer dans le réseau domestique.

Une fois introduit dans le réseau domestique, nous souhaitons que le nouvel équipement puisse connaître l'ensemble des équipements appartenant déjà au réseau domestique, et *prouver* à chacun d'eux qu'il appartient bien au même réseau domestique qu'eux. Cette preuve prend la forme d'une chaîne de certification allant de n'importe quel équipement appartenant au réseau domestique vers le nouvel équipement. Il s'agit donc de maintenir sur chaque équipement la chaine de certification pour l'ensemble des équipements du réseau domestique. Pour cela, les équipements du réseau domestique vont se « synchroniser ».

II.1. Modélisation d'un équipement.

Suivant notre approche, un équipement est identifié par un nom et par une paire de clé (publique, privée). Un réseau domestique est alors défini par l'ensemble :

$$RD = \{(Id_i, Pub_i) \mid 0 < i < N\}$$

où Id_i est l'identité d'un équipement, et Pub_i la clé publique de cet équipement. Chaque équipement gère localement la connaissance qu'il a de son réseau domestique à l'aide de deux ensembles : CA et DA. Pour un équipement A, un équipement B est dans CA[A] (lire « l'ensemble des Autorités de Certification *racine* reconnues par A ») lorsque A possède un certificat généré par B et portant sur sa clé publique Pub_A, noté $Cert_B(Pub_A)$. En d'autres termes, A peut prouver, à l'aide du certificat $Cert_B(Pub_A)$ qu'il appartient au même réseau que B. L'ensemble CA[A] peut alors s'écrire :

$$CA[A] = \{(Id_i, Pub_i, Cert_{Idi}(Pub_A)) \mid 0 < i < N\}$$

Pour un équipement A, un équipement B est dans DA[A] (lire « l'ensemble des Autorités Dérivées reconnues par A ») lorsque A peut exhiber une chaîne de certification sous la forme :

$$(Cert_B(Pub_{C1}), Cert_{C1}(Pub_{C2}), \ldots, Cert_{Cn}(Pub_{Cn+1}), Cert_{Cn+1}(Pub_A))$$

avec $C_i \in DA[A]$ pour $0 < i < n+1$. A l'aide de cette chaîne de certification, A peut prouver à B qu'ils appartiennent au même réseau domestique. De manière similaire à CA[A], l'ensemble DA[A] est constitué de triplet (C_i, Pub_{Ci}, $Cert_{Ci}(Pub_{Cj})$).

Attention : par abus de langage, nous noterons $B \in CA[A]$ pour désigner que A possède un certificat de B portant sur sa clé publique Pub_A, c'est-à-dire, (B, Pub_B, $Cert_B(Pub_A)$) $\in CA[A]$. De manière similaire, nous noterons $B \in DA[A]$ pour désigner que A possède une chaîne de certification pour B.

II.2. Insertion d'un équipement.

Considérons, dans un premier temps, l'insertion d'un nouvel équipement C via l'équipement A, cet équipement C n'appartenant au préalable à aucun réseau domestique. Afin de contrôler l'opération d'insertion, l'utilisateur, en tant qu'administrateur des équipements A et C, doit valider l'insertion sur chaque équipement (i.e., répondre positivement à la requête « Insérer l'équipement C (Oui/Non) ? » sur l'équipement A et de même pour la requête « Insérer l'équipement A (Oui/Non) ? » sur l'équipement C). Après validation des opérations d'insertion par l'utilisateur (sur A ***et*** sur C), l'équipement A certifie la clé publique de C (i.e., A calcule $Cert_A(Pub_C)$), envoie le certificat ainsi obtenu à C, qui met à jour CA[C]. Inversement, A obtient le certificat $Cert_C(Pub_A)$ et met à jour CA[A].

Une fois l'échange de certificats effectué, il s'agit de mettre à jour l'ensemble DA[C]. Ceci s'effectue de la manière suivante :

- $DA[C] := CA[A] \cup DA[A]$

En effet, l'obtention du certificat de A (i.e., $Cert_A(Pub_C)$) et l'insertion de A dans CA[C] signifie que C reconnait maintenant A comme autorité de certification. A ce titre, C admet que les équipements de CA[A] appartiennent maintenant à son réseau domestique, et qu'il s'agit d'autorités dérivées. En d'autres termes, les équipements de CA[A] appartiennent à DA[C] :

- si $B \in CA[A]$ alors $B \in DA[C]$

Maintenant, les équipements de DA[A] correspondent aux équipements que A reconnaît comme autorités dérivées. De ce fait, C doit également les reconnaître comme autorités dérivées :

- si $B \in DA[A]$ alors $B \in DA[C]$

Il est aisé de vérifier que, par construction, à la fin de la fin de l'insertion, pour chacun des équipements B de DA[C], C est en mesure de présenter une chaîne de certification lui permettant de prouver à B qu'ils appartiennent au même réseau domestique.

II.3. Synchronisation des équipements.

Dans la section précédente, nous avons vu les opérations à effectuer lors de l'insertion d'un nouvel équipement. L'évolution se faisant via un unique équipement, il s'agit de propager l'information entre les différents équipements. Pour cela, nous allons permettre aux équipements d'un réseau domestique de se synchroniser lorsqu'ils sont mis en présence. Considérons deux équipements A et B d'un même

réseau domestique. Considérons, dans un premier temps que B $\in$ CA[A] (et donc que A $\in$ CA[B]). Les opérations de synchronisation sont les suivantes :

- Mise à jour de DA[A] :
  - Si C $\in$ CA[B] $\cup$ DA[B] et C $\notin$ CA[A] $\cup$ DA[A] alors DA[A] := DA[A] $\cup$ C
- Mise à jour de DA[B] :
  - Si D $\in$ CA[A] $\cup$ DA[A] et D $\notin$ CA[B] $\cup$ DA[B] alors DA[B] := DA[B] $\cup$ D

Considérons maintenant que B $\in$ DA[A] (notons que dans ce cas, B ne connaît pas nécessairement A). En premier lieu, puisque B $\in$ DA[A], A va présenter à B la chaîne de certification permettant de prouver qu'ils appartiennent au même réseau domestique. Une fois la chaîne de certification vérifiée, A et B vont mutuellement se certifier (à l'instar de l'insertion d'un nouvel équipement, mais cette fois, sans demander la validation de l'utilisateur puisque la chaîne de certification démontre qu'ils appartiennent au même réseau) et s'insérer mutuellement dans leur CA. Une fois cette opération effectuée, ils peuvent synchroniser leur ensemble DA comme précédemment.

A nouveau, il est aisé de vérifier que, par construction, à la fin de la fin d'une synchronisation, pour chacun des équipements C de DA[A], A est en mesure de présenter une chaîne de certification lui permettant de prouver à C qu'ils appartiennent au même réseau domestique.

Ainsi, au fil des rencontres entre équipements, l'information se propage. En faisant l'hypothèse que les équipements interagissent plus souvent que n'évoluent le réseau, hypothèse vérifiée dans le cadre des réseaux domestiques, la connaissance locale de chaque équipement de son réseau tend vers la connaissance globale du réseau.

III. Cahier des charges

Il s'agit d'implémenter, en ***mode caractère***[1], l'insertion d'un équipement et la synchronisation des équipements. Le développement sera fait en Java sous Windows ou Linux. Pour ce qui est des fonctions cryptographiques (génération de clés RSA, certification de clés ...), le développement s'appuiera sur les classes fournies nativement par Java, ainsi que sur la librairie ***Bouncy Castle***, disponible à l'adresse `www.bouncycastle.org`. Pour être plus précis, la librairie ***Bouncy Castle***, sera essentiellement utilisée pour générer les certificats, alors que les autres opérations cryptographiques utiliseront les classes natives de Java.

Des exemples d'utilisation de la librairie ***Bouncy Castle*** sont disponibles dans la distribution, ainsi qu'en annexe 4.2.

Le développement s'effectuera sur une seule machine, chaque équipement prenant la forme d'un processus. Les communications entre les équipements seront donc réalisées par des sockets. En annexe, vous trouverez les fonctions permettant de créer et utiliser des sockets.

[1] Les plus motivés d'entre vous peuvent viser le mode graphique.

A la suite, nous détaillons les travaux attendus sur chacune des 3 premières séances. La 4ème séance sera consacrée aux finitions et à la rédaction. Finalement, les résultats du TL feront l'objet d'une soutenance orale.

III.1. 1ère séance : Modélisation des équipements et de leurs attributs

L'objectif de cette première séance va être de modéliser nos équipements. Pour cela, nous allons créer notre classe `Equipement` (exemple de classe fournie en annexe, à compléter) dont chaque instance représentera un équipement de notre communauté.

Cette classe Equipement devra contenir tous les attributs et méthodes pertinentes à son fonctionnement. La principale difficulté de cette classe reste la définition de ses attributs cryptographiques.

Rappelons que, suivant notre approche, un équipement est identifié par une identité et une paire de clés (publique, privée). Lors de sa *création*, un équipement commence donc par obtenir son identité (dans notre cas, elle sera fournie par l'utilisateur), générer une bi-clé, et auto-certifier sa clé publique.

Concrètement, dans le cadre de cette séance, nous nous donnons les classes suivantes :

- la classe `PaireClesRSA` correspondant à la paire de clés RSA, et dont un exemple est donné en annexe.
- la classe `Certificat` regroupant l'ensemble des méthodes pour la certification des clés publiques, et dont un exemple est donné en annexe.

Etant données ces deux classes, la *création* d'un équipement suit les étapes suivantes :

1. Demande de l'identité de l'équipement Id.
2. Génération de la paire de clé RSA (PrivId, PubId).
3. Auto-certification de la clé publique RSA CertId(PubId).
4. Vérification du certificat autosigné.
5. Affichage du certificat.

Il est également demandé d'implémenter les opérations suivantes :

1. Génération de deux paires de clés *pk1* et *pk2*.
2. Certification de la clé publique de *pk1* à l'aide de la clé privée de *pk2*.
3. Vérification du certificat portant sur la clé publique de *pk1*.
4. Affichage du certificat.

Au final, à l'issue de cette première séance, les classes `Equipement`, `PaireClesRSA` et `Certificat` devront être implémentées et testées.

III.2. 2ème séance : notre équipement et ses attributs

L'objectif de cette seconde séance est de permettre aux équipements de communiquer à l'aide de `sockets`. En annexe, vous trouvez le code pour l'implémentation en Java des `sockets`. A noter qu'il faudra choisir l'équipement jouant le rôle de serveur, et celui jouant le rôle de client. Vous testerez votre implémentation en permettant à deux équipements de s'échanger des chaines de caractère (par exemple, leur nom).

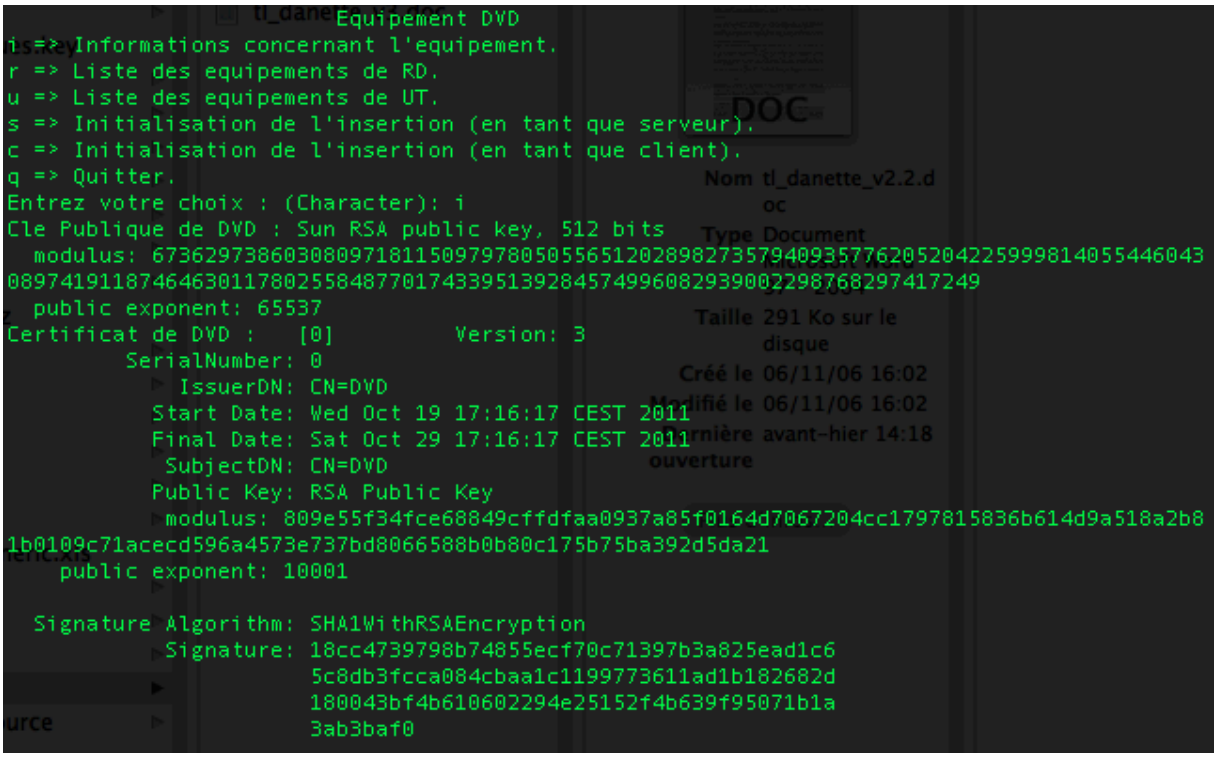
Une fois la communication entre équipements établie, vous implémenterez l'opération d'insertion d'un équipement. Pour cela, il faudra permettre aux équipements d'échanger des certificats (n'oubliez pas de vérifier le certificat lors de la réception), en n'oubliant pas de faire intervenir l'utilisateur (la fameuse question simple « Insérer l'équipement DVD X452 (Oui/Non) ? »). Pour l'échange des certificats, nous vous conseillons de les coder au format PEM avant l'envoie, et de les regénérer à partir du format PEM reçu (voir code en annexe).

Vous êtes maintenant en mesure d'implémenter les ensembles CA et DA sur chaque équipement. Nous vous laissons le choix de l'implémentation …

Afin de pouvoir enchainer les opérations d'insertion et visualiser les ensembles CA et DA de chaque équipement, créez l'interface ***en mode caractère*** permettant à l'utilisateur d'effectuer les actions suivantes sur chaque équipement (un exemple de rendu est fourni ci-dessous) :

- Afficher les informations de l'équipement (par exemple son identifiant, son nom, son certificat, les autres équipements qu'il connaît …).
- Afficher la liste des équipements de CA, ainsi que le certificat obtenu d'eux.
- Afficher la liste des équipements de DA, ainsi que le certificat obtenu d'eux.
- Initialiser l'insertion en tant que serveur.
- Initialiser l'insertion en tant que client.
- Tout autre option/commande que vous jugeriez utile …

Vous pouvez bien évidemment implémenter toute autre option/commande que vous jugeriez utile …



A l'issue de cette seconde séance, la phase d'insertion doit fonctionner, et il doit être possible de visualiser la connaissance de chaque équipement.

III.3. 3ème séance : Synchronisation entre équipements

Cette 3ème séance va nous permettre d'implémenter la synchronisation de deux équipements. Dans une première étape, nous enrichissons l'interface de chaque équipement afin de rajouter les options suivantes :

- Initialiser la synchronisation en tant que serveur.
- Initialiser la synchronisation en tant que client.

Le protocole à mettre en œuvre entre les équipements doit permettre à chaque équipement de compléter sa connaissance en fonction de la connaissance de l'autre. Dans un premier temps, nous considérerons que les deux équipements se connaissent mutuellement, c'est à dire que chacun des équipements appartient à l'ensemble CA de l'autre. Dans un second temps, nous traiterons le cas général où un des équipements appartient à l'ensemble DA de l'autre, mais pas forcément l'inverse : il devra alors fournir la preuve qu'ils appartiennent au même réseau domestique.

Une attention particulière devra être apportée à la vérification des chaînes de certification.

A la fin de cette 3ème séance, nous obtenons des équipements qui sont à même de gérer la synchronisation avec des équipements, et ainsi de propager de proche en proche la connaissance du réseau domestique.

III.4. 4éme séance : finalisation et rédaction

Cette dernière séance servira à terminer l'ensemble du programme et à rédiger un rapport le présentant. Il conviendra dans le rapport d'insister sur les points qui ont posés problème et sur les méthodes employées pour les résoudre ou les contourner.

Pour les plus « motivés », il pourra être envisager de développer une interface graphique plutôt qu'une interface en ligne de commande.

IV. Annexes techniques

On trouvera ici :
– des suggestions pour la définition des classes de votre application,
– des exemples d'implémentation de sockets en Java,
– un guide d'installation de Bouncy Castle, et
– des exemples d'utilisation des fonctions cryptographiques de Java et de Bouncy Castle.

IV.1. Suggestions pour la définition de vos classes

IV.1.1 Classe `Equipement`

```
public class Equipement {
```

```
        private PaireClesRSA maCle; // La paire de cle de l'equipement.
        private Certificat monCert; // Le certificat auto-signe.
        private String monNom; // Identite de l'equipement.
        private int monPort; // Le numéro de port d'ecoute.

        Equipement (String nom, int port) throws Exception {
                // Constructeur de l'equipement identifie par nom
                // et qui « écoutera » sur le port port.
        }
        public void affichage_da() {
                // Affichage de la liste des équipements de DA.
        }

        public void affichage_ca() {
                // Affichage de la liste des équipements de CA.
        }

        public void affichage() {
                // Affichage de l'ensemble des informations
                // de l'équipement.
        }

        public String monNom (){
                // Recuperation de l'identite de l'équipement.
        }

        public PublicKey maClePub() {
                // Recuperation de la clé publique de l'équipement.
        }

        public Certificat monCertif() {
                // Recuperation du certificat auto-signé.
        }
}
```

IV.1.2 Classe PaireClesRSA, à compléter

```
public class PaireClesRSA {
        private KeyPair key;

        PaireClesRSA() {
                // Constructeur : génération d'une paire de clé RSA.
        }

        public PublicKey Publique() {
                // Recuperation de la clé publique.
        }
        public PrivateKey Privee() {
                // Recuperation de la clé privée.
        }
}
```

IV.1.3 Classe Certificat, à compléter

```
public class Certificat {
        static private BigInteger seqnum = BigInteger.ZERO;
        public X509Certificate x509;

        Certificat(String nom, PaireClesRSA cle, int validityDays) {
                // Constructeur d'un certificat auto-signé avec
                // CN = nom, la clé publique contenu dans PaireClesRSA,
                // la durée de validité.
        }
        public boolean verifCertif (PublicKey pubkey) {
                // Vérification de la signature du certificat à l'aide
                // de la clé publique passée en argument.
```

```
	}
}
```

IV.2. Implémentation de sockets en Java

IV.2.1 Socket coté serveur

```
ServerSocket serverSocket = null;
Socket NewServerSocket = null;
InputStream NativeIn = null;
ObjectInputStream ois = null;
OutputStream NativeOut = null;
ObjectOutputStream oos = null;
// Creation de socket (TCP)
try {
	serverSocket = new ServerSocket(this.monPort);
} catch (IOException e) {
	// Gestion des exceptions
}
// Attente de connextions
try {
	NewServerSocket = serverSocket.accept();
} catch (Exception e) {
	// Gestion des exceptions
}
// Creation des flux natifs et evolues
try {
	NativeIn = NewServerSocket.getInputStream();
	ois = new ObjectInputStream(NativeIn);
	NativeOut = NewServerSocket.getOutputStream;
	oos = new ObjectOutputStream(NativeOut);
} catch (IOException e) {
	// Gestion des exceptions
}
// Reception d'un String
try {
	String res = (String) ois.readObject();
	System.out.println(res);
} catch (Exception e) {
	// Gestion des exceptions
}
// Emission d'un String
try {
	oos.writeObject("Au revoir");
	oos.flush();
} catch (Exception e) {
	// Gestion des exceptions
}
// Fermeture des flux evolues et natifs
try {
	ois.close();
	oos.close();
	NativeIn.close();
	NativeOut.close();
} catch (IOException e) {
	// Gestion des exceptions
}
// Fermeture de la connexion
try {
	NewServerSocket.close();
} catch (IOException e) {
	// Gestion des exceptions
}
// Arret du serveur
try {
	serverSocket.close();
```

```
} catch (IOException e) {
        // Gestion des exceptions
}
```

IV.2.2 Socket coté client

```
int ServerPort;
String ServerName;
Socket clientSocket = null;
InputStream NativeIn = null;
ObjectInputStream ois = null;
OutputStream NativeOut = null;
ObjectOutputStream oos = null;
// Creation de socket (TCP)
try {
        clientSocket = new Socket(ServerName,ServerPort);
} catch (Exception e) {
        // Gestion des exceptions
}
// Creation des flux natifs et evolues
try {
        NativeOut = clientSocket.getOutputStream();
        oos = new ObjectOutputStream(NativeOut);
        NativeIn = clientSocket.getInputStream();
        ois = new ObjectInputStream(NativeIn);
} catch (Exception e) {
        // Gestion des exceptions
}
// Emission d'un String
try {
        oos.writeObject("Bonjour");
        oos.flush();
} catch (Exception e) {
        // Gestion des exceptions
}
// Reception d'un String
try {
        String res = (String) ois.readObject();
        System.out.println(res);
} catch (Exception e) {
        // Gestion des exceptions
}
        // Fermeture des flux evolues et natifs
try {
        ois.close();
        oos.close();
        NativeIn.close();
        NativeOut.close();
} catch (IOException e) {
        // Gestion des exceptions
}
// Fermeture de la connexion
try {
        clientSocket.close();
} catch (IOException e) {
        // Gestion des exceptions
}
```

IV.3. Bibliothèque Bouncy Castle

La bibliothèque Bouncy Castle peut être téléchargée à l'adresse suivante :

www.bouncycastle.org

Choisissez la version correspondant à votre JDK (`java -version` dans un terminal ou une fenêtre Ms-Dos).

Pour l'utiliser, plusieurs méthodes sont envisageables : la première consiste à copier l'ensemble des fichiers JAR dans le répertoire de votre JRE Java, l'autre est d'inclure chaque fichier JAR comme bibliothèques référencées dans votre projet Eclipse. Sur les machines de l'école, comme vous n'avez pas les droits `root` : vous ne pouvez faire que la seconde méthode.

Pour cela, clique droit sur votre projet, « Propriétés », « Bibliothèques », « Ajouter des fichiers JAR externes » et vous sélectionnez les fichiers téléchargés (en fait, nous n'avons besoin que des fichiers `bcpg-jdk`, `bcprov-jdk`, et `jce-jdk`).

Pour exécuter un programme utilisant les fichiers JAR précédents, il convient d'ajouter - *-classpath* à la ligne de commande *java.* Ceci est fait automatiquement par Eclipse, mais si vous utilisez plusieurs fenêtres terminal, une fenêtre par équipement, vous devrez le faire *manuellement*.

IV.4. Exemple de code pour les fonctions cryptographiques

Remarque importante : les exemples de code ci-dessous sont donnés à titre indicatif. Dans certains cas, ils peuvent être utilisé tel quel. Dans d'autres, vous aurez besoin de chercher sur le Net afin de corriger les erreurs de compilation.

IV.4.1 Génération d'une paire de clés RSA

```
// On va mettre un peu d'alea :
SecureRandom rand = new SecureRandom();
// On initialise la structure pour la generation de cle :
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
// On definit la taille de cle :
kpg.initialize(512, rand);
// On genere la paire de cle :
KeyPair key = kpg.generateKeyPair();
```

IV.4.2 Génération d'un certificat auto-signé

```
// On recupere la cle publique et la cle privee :
PublicKey pubkey = cle.Publique();
PrivateKey privkey = cle.Privee();

// On cree la structure qui va contenir la signature :
ContentSigner sigGen = new
       JcaContentSignerBuilder("SHA1withRSA").setProvider("BC").build(privkey);

// On cree la structure qui contient la cle publique a certifier :
SubjectPublicKeyInfo subPubKeyInfo =
             SubjectPublicKeyInfo.getInstance(pubkey.getEncoded()) ;


// Le nom du proprietaire et du certifieur :
// ici, les memes car auto-signe.
X509Name issuer = new X509Name ("CN="+nom);
X509Name subject = new X509Name ("CN="+nom);

// Le numero de serie du futur certificat
seqnum=seqnum.add(BigInteger.ONE);

// Le certificat sera valide a partir d'hier ...
Date startDate = new Date(System.currentTimeMillis()-24*60*60*1000);
// ... et pour 10 jours
Date endDate = new Date(System.currentTimeMillis()+10*24*60*60*1000);

// On cree la structure qui va nous permettre de creer le certificat
X509v1CertificateBuilder v1CertGen = new X509v1CertificateBuilder(
       issuer, seqnum, startDate, endDate, subject, subPubKeyInfo);

// On calcule la signature et on cree un certificate !
X509CertificateHolder x509holder = v1CertGen.build(sigGen);

// On transforme le x509holder en certificate x509 !
X509Certificate x509 = new JcaX509CertificateConverter().
                                  setProvider( "BC" ).getCertificate( x509 );
```

IV.4.3 Vérifier un certificat

```
// A partir de la cle publique de l'issuer, on construit
// une structure pour verifier le certificat !
ContentVerifierProvider verifier =
       new JcaContentVerifierProviderBuilder().setProvider("BC").build(pubKey);

 // Verification d'un certificat  !
if (!x509.isSignatureValid(contentVerifierProvider))
{
       System.err.println("signature invalid");
}
```