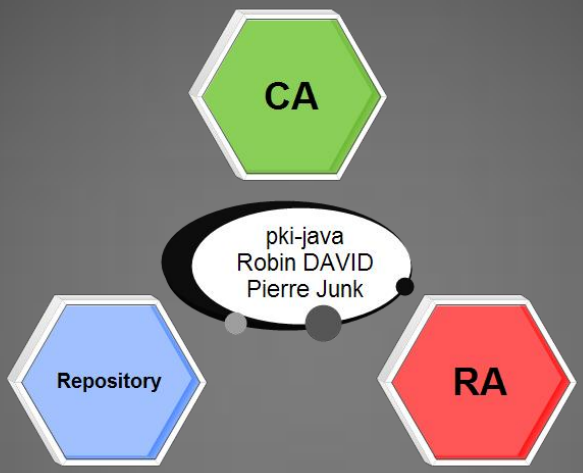


Architecture PKI en Java

Robin David & Pierre Junk

Ce rapport présente l'architecture et l'implémentation en Java d'un PKI dans un contexte d'entreprise.

http://code.google.com/p/pki-java

5/13/2012

Table of Contents

Introduction

1. Infrastructure à clé publique ou (Public Key Infrastructure PKI)

Une infrastructure à clé publique, est un ensemble d'entité dont le rôle est d'assurer la création, la distribution et la révocation de certificats numériques. Certificats qui peuvent être utilisés pour des utilisations aussi varié que la signature de mail pour un client, le chiffrement de session SSL ou encore la signature de code.

2. Objectif du Projet

L'objectif de ce projet est de réaliser une architecture à clé publique en Java en utilisant la librairie BouncyCastle. Ceci dans l'objectif de pouvoir faire dialoguer deux clients de manière sécurisé chiffré grâce aux certificats. Ainsi les deux utilisateurs doivent sont sûrs que leur correspondant est bien celui qu'il prétend être. L'infrastructure à clé publique nous permet de répondre à ce problème grâce au système de certificat. Bien entendu cette architecture doit permettre toutes les fonctionnalités offertes par un PKI tel que l'enregistrement auprès d'une autorité d'enregistrement, la demande de certificat, la gestion des CRL ou encore la gestion de l'OCSP.

3. Plan du rapport

Ce rapport se sépare en trois parties distinctes. La première présente de manière générale le déroulement du projet, le fonctionnement de notre PKI et les choix qui ont été fait en termes d'implémentation. La deuxième partie approfondie les explications sur le fonctionnement du PKI en expliquant des aspects beaucoup plus techniques mais tout aussi important pour la sécurité et la qualité de conception. Enfin la dernière partie beaucoup plus pratique explique comment configurer le PKI, comment l'utiliser mais aussi comment l'administrer.

I. Fonctionnement global du PKI

1. Plan de travail et déroulement de la réalisation

La réalisation de ce projet s'est déroulé en plusieurs grandes phases toutes aussi importantes les unes que les autres.

a. Familiarisation BouncyCastle

La familiarisation avec la librairie BouncyCastle a été la phase la plus longue et la plus pénible de ce projet.
En effet la librairie BouncyCastle est très changeante et il est très dur vu le niveau technique du projet à réaliser de trouver des exemples.
Il y a donc eu un très gros travail pour la réalisation de petits programmes de tests pour tester chacune des fonctionnalités de BouncyCastle telles que la création de certificat, la gestion des CRL, la signature du CSR (au format PKCS10CertificationRequest) ou encore la gestion des requêtes OCSP avec les objets OCSPReq etc.
Il est très important de noter que aucune des méthodes utilisées dans ce projet ne sont dépréciées !

b. Elaboration théorique de l'architecture

La deuxième phase de ce projet et l'une des plus importantes est le travail de documentation sur les PKI. Il s'agit de bien comprendre en détail le fonctionnement et les principes techniques.
Cette étude nous a permis de mettre au point notre propre infrastructure qui respecte tous les principes théoriques et pratiques des PKI.

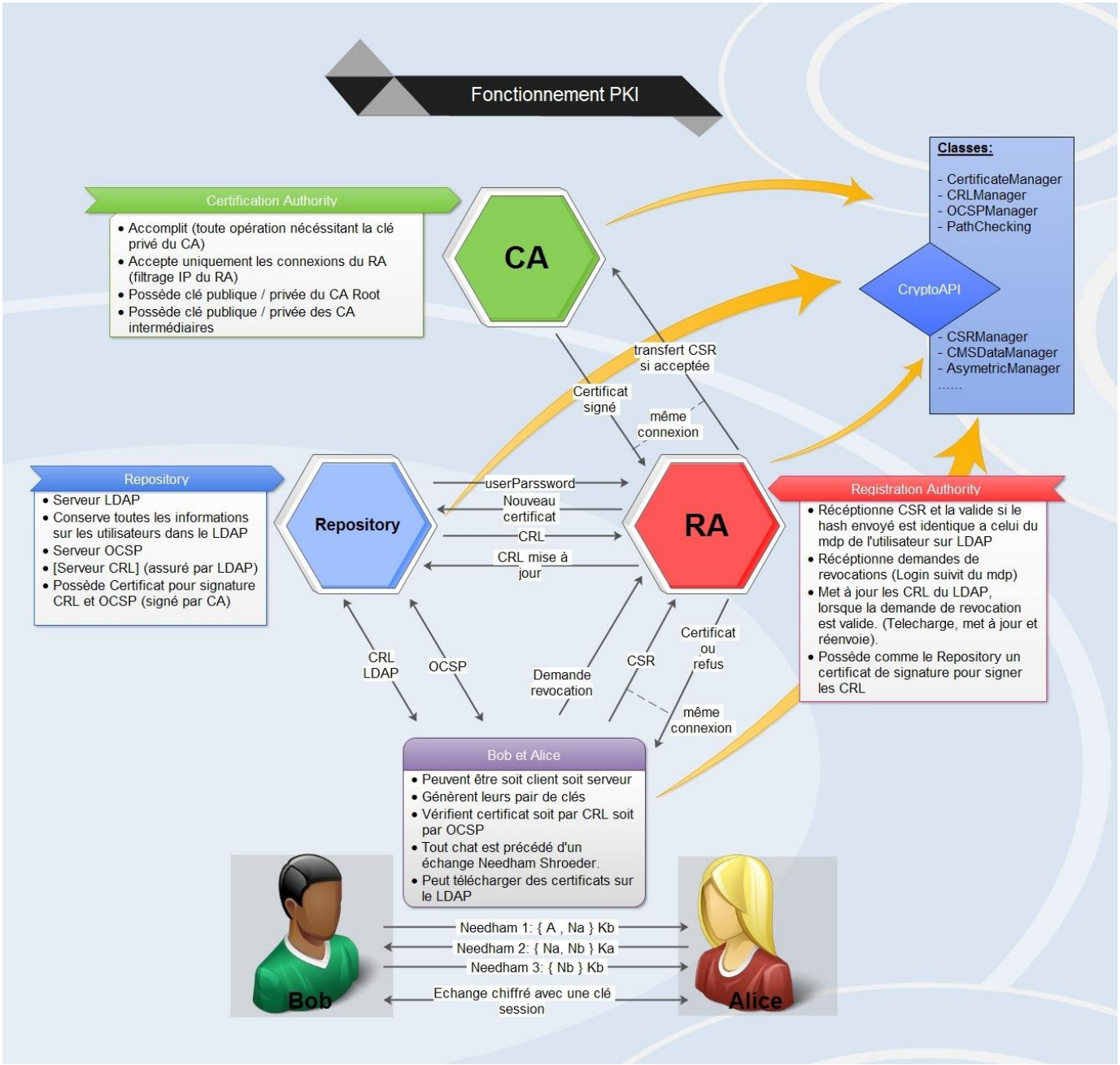
c. Réalisation du projet

Pour rapprocher le plus possible notre projet d'un cadre de travail en entreprise, nous avons essayé de développer notre infrastructure comme si elle devait être mise en place dans une entreprise. Dans cet optique, nous avons établi un pseudo cahier des charges d'une entreprise voulant mettre en place une Infrastructure à Clé Publique pour permettre à ses employés de pouvoir avoir des conversations textuelles sécurisées. Le personnel de l'entreprise autorisé est enregistré dans un annuaire LDAP.

La réalisation du projet en Java a été la phase pratique de ce projet et a pu être menée sans grands problèmes grâce à tous les programmes de tests de BouncyCastle qui avaient été créés au préalable. Ceci nous a permis d'avancer vite et efficacement même si bien sûr certains éléments ont dû être changés par rapport aux prévisions.

2. Architecture globale du PKI

Le schéma ci-dessous présente notre architecture du PKI tel que nous l'avons conçue et qui est quasi identique aux implémentations commerciales de PKI.



Voici quelques compléments d'informations :

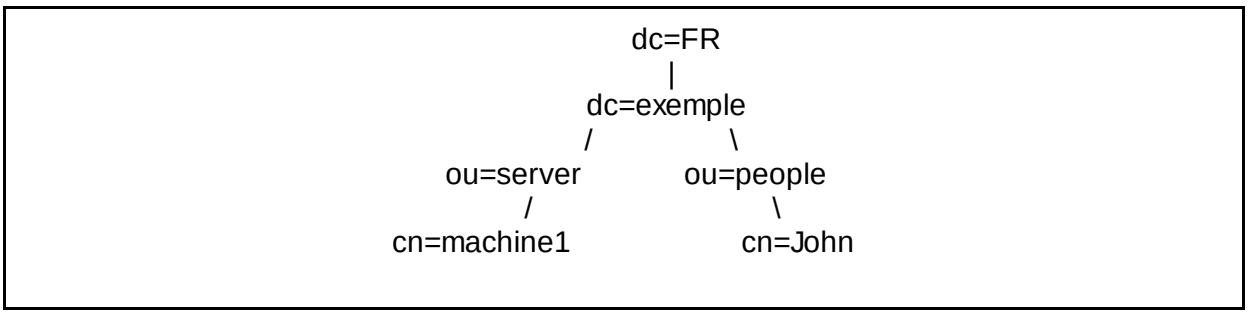
- Tout ce qui touche à la cryptographie ou à BouncyCastle se trouve dans la CryptoAPI qui contient un ensemble de classes. Ceci permet une bonne couche d'abstraction entre la cryptographie et le PKI lui-même. Ainsi chaque entité y accède selon ses besoins.
- Le CA n'accepte que les connexions du RA et ne fait que signer les CSR (c'est le rôle du RA de vérifier si elles sont valides).
- Le repository et le RA possède un certificat qui leurs sert à la signature des CRL et des réponses OCSP.

- Le client interagit avec le RA, soit pour une CSR soit pour demande de révocation.
- Le client contact le LDAP soit pour obtenir le certificat d'un autre utilisateur soit pour faire une requête OCSP.

3. Architecture du LDAP

a. Définition

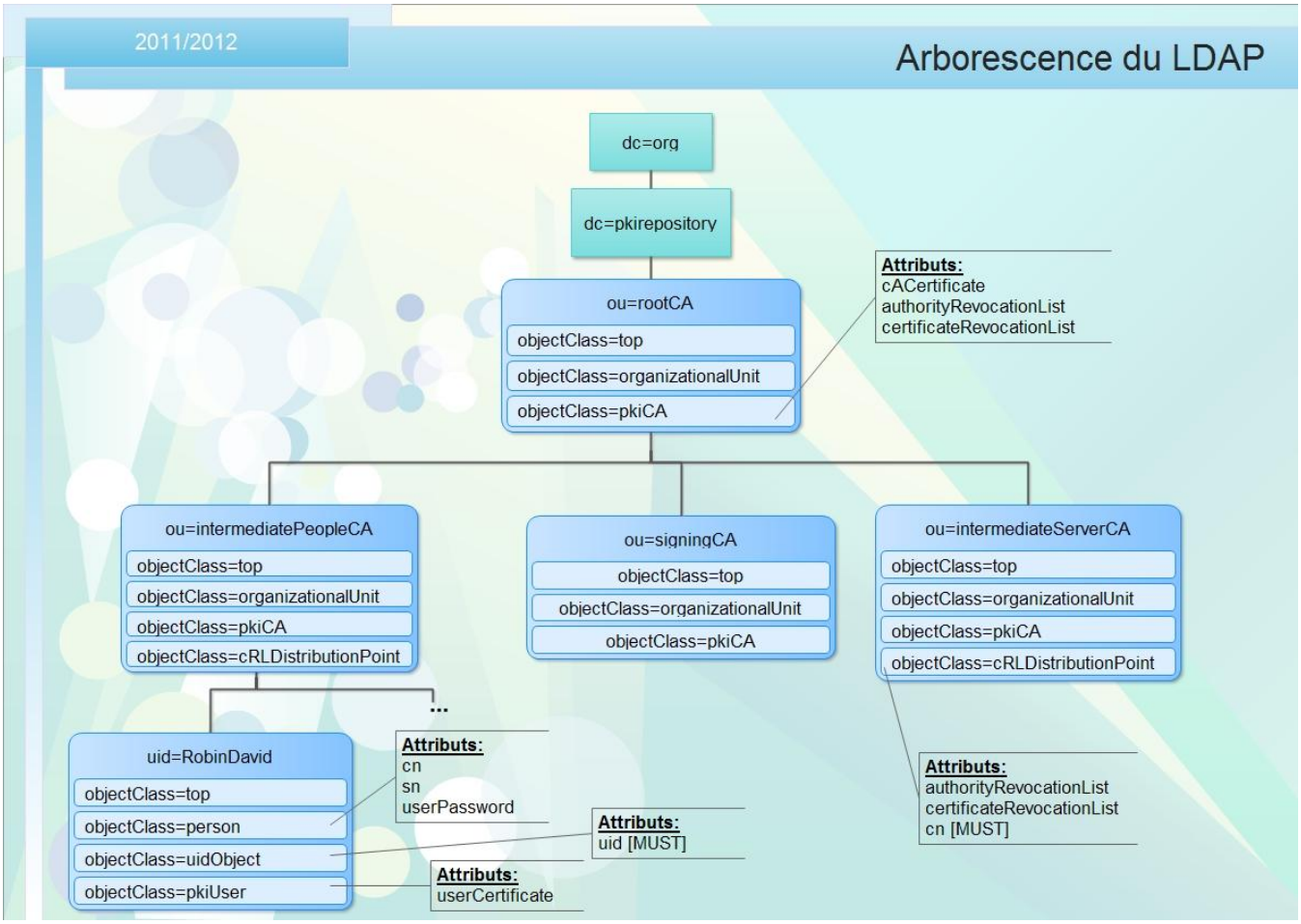
Lightweight Directory Access Protocol - LDAP - est un système annuaire modélisé sous forme d'arbre. Chaque nœud correspond à un attribut lié à une valeur.



Un objet de l'arbre d'une feuille à la racine est appelé *distinguished name* par exemple cn=machine1,ou=server,dc=exemple,dc=FR.

Nous avons choisi d'utiliser un LDAP pour jouer le rôle de Repository car c'est la méthode la plus utilisée et la plus standard. De plus elle correspond tout à fait à notre besoin pour les certificats, car ils utilisent la norme X509 qui est dérivée de la norme X500 des LDAP. L'autre solution envisagée était l'utilisation de keystore mais ceux-ci ne satisfaisaient pas nos besoins d'accessibilité. De plus la structure LDAP permet de stocker toutes les informations nécessaires sur l'utilisateur (Nom et Prénom, Identifiant, Certificat, Clé,..).

b. Notre arbre d'entrées



Compléments d'information :

- rootCA : Nœud qui possède le certificat root auto signé.
- intermediatePeopleCA : Unité organisationnelle dans lequel se trouvent tous les utilisateurs de l'entreprise. Elle implémente aussi le schéma pkiCA et possède elle aussi son certificat signé par rootCA.
- signingCA : Nœud final qui contient le certificat dédié à la signature des CRL OCSP. Ainsi un client qui reçoit une CRL ou une réponse OCSP peut vérifier la signature grâce au certificat qui sur signingCA
- intermediateServerCA : Unité organisationnelle dans lequel peuvent se trouver d'éventuels certificat serveurs (montre la grande modularité de notre arbre).
- Chaque unité organisationnelle intermédiaire implémente crlDistributionPoint qui est le schéma standard permettant de stocker des listes de révocation.
- Chaque utilisateur implémente pkiUser ce qui lui permet de disposer d'un attribut (userCertificate).
- **Cette architecture est très modulaire et peut s'adapter à n'importe quel LDAP existant sous réserve d'utiliser les schémas standard pkiCA, pkiUser, crlDistributionPoint...**

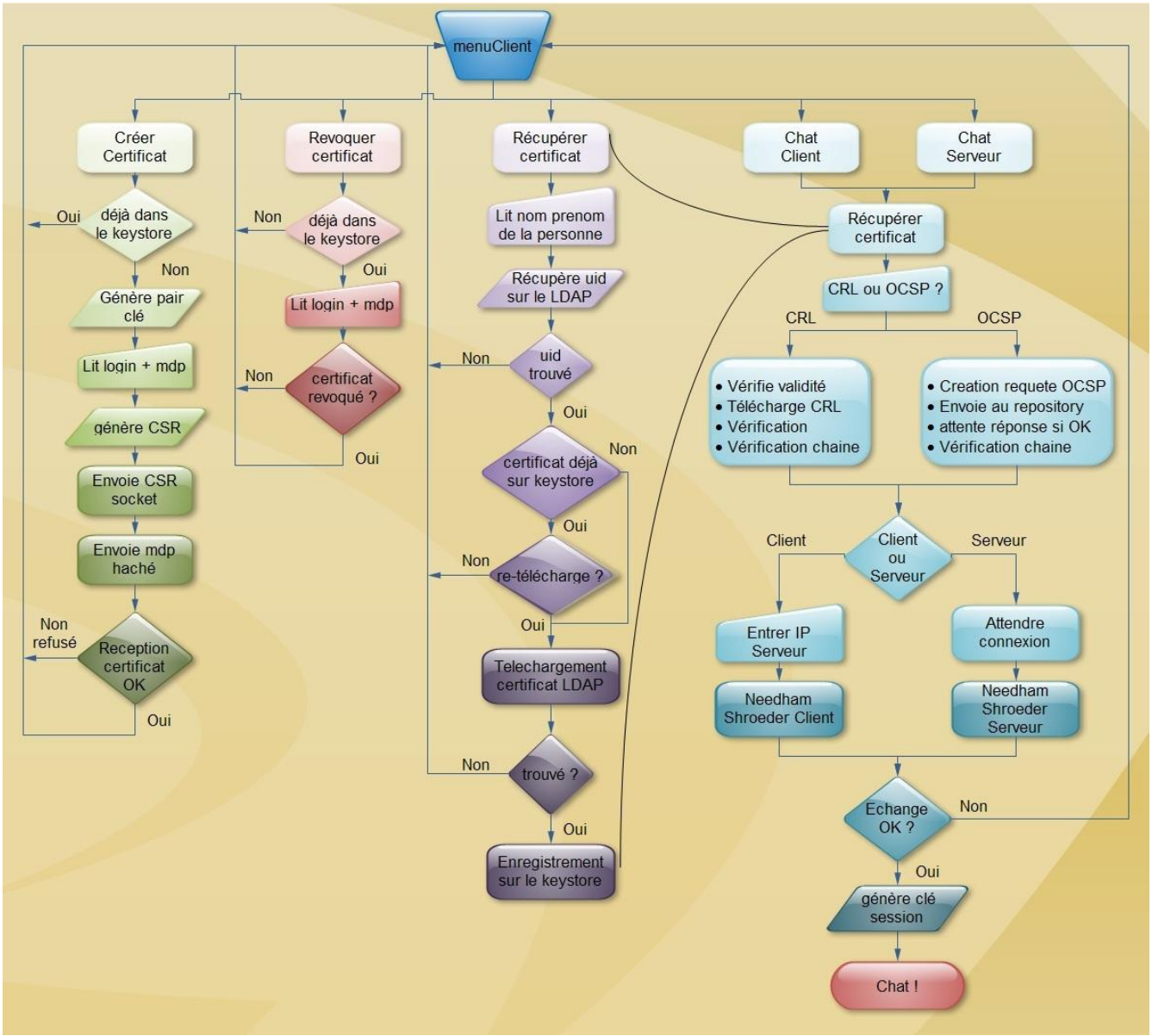
4. Problèmes rencontrés

La première difficulté consistait à trouver une architecture adaptable et correspondante à nos besoins pour l'infrastructure à clé publique que nous devions mettre en place. Nous avons commencé par nous documenter sur le sujet afin de dégager une première version de notre architecture. Quelques modifications ont eu lieu pendant le développement du projet jusqu'à aboutir à la version finale.
De très loin le plus gros problème rencontré a été de n'utiliser que des méthodes non-dépréciée pour tout ce qui touche au BouncyCastle. En effet il est très dur voire impossible de trouver des exemples de code utilisant les méthodes non dépréciées. Même les exemples fournis sur le site de BouncyCastle utilisent des méthodes dépréciées. Cependant il était pour nous inconcevable dans un contexte sécuritaire d'utiliser des méthodes dépréciées. Mais nous avons réussis.
Autrement les autres problèmes rencontrés sont plutôt des dilemmes techniques sur le comportement que doit adopter le PKI dans certaines situations. Voici quelques exemples ainsi que le choix effectué :

- La question de l'entité devant recevoir les demandes de révocation n'est pas vraiment définie dans l'architecture d'un PKI. Notre choix s'est porté sur le RA qui reçoit déjà les CSR
- Le RA doit-il avoir la clé privé du CA intermédiaire pour signer les CRL ? Nous avons été décidés que non, seul le CA doit posséder les clés privés. Pour signer le RA utilise un certificat dédié.
- Quelle entité héberge le répondeur OCSP ? Notre choix s'est porté sur le Repository qui doit consulter les CRL qu'il héberge.
- La transmission des objets entre les différents acteurs nous a, au début, posé quelques problèmes. Beaucoup d'objet de types différents sont envoyés et lus par les autorités de l'infrastructure. Cette diversité entraînait des soucis avec l'utilisation des ObjectInputStream et ObjectOutputStream. Pour résoudre ce problème, nous avons décidé d'envoyer toutes les données devant transiter sous forme de tableau de Byte, envoyés et lus par des DataOutputStream et DataInputStream.

5. Fonctionnement client

Voici un schéma qui synthétise le fonctionnement du Client dont les 4 grandes fonctionnalités sont de créer un certificat, révoquer un certificat, télécharger le certificat d'un pair et démarrer une session de Chat.



II. Caractéristiques techniques

1. Choix techniques

a. Extensions des certificats

La question la plus importante qui s'est posé pendant le projet est de savoir quelles sont les extensions à donner aux certificats et donc quels droits donner aux certificats. Voici les droits associés à chaque type de certificats :

	Certificat root	Certificat intermédiaire	Certificat signature	Certificat client	Description
BasicConstraints	true	true	false	false	Définit si est une autorité pouvant signer d'autres certificats
Pathlen	1	0	/	/	Longueur de la chaine de sous certificats pouvant signer d'autres certificats
SubjectKeyId	Respectivement la clé publique de chaque certificat				Identifie la clé publique associée au subject
KeyUsage	keyCertSign				Droit que possède le certificat
ExtendedKeyUsage	anyExtendedKeyUsage				Droit étendu que possède le certificat
crlDistributionPoint	Adresse du Repository définit en fonction de l'IP du LDAP.				Point d'accès pour les CRL
authorityInfoAccess	Adresse du Repository définit en fonction de l'IP du Repository dans le fichier de configuration				Adresse de l'autorité, utilisée pour mettre l'adresse du répondeur OCSP

b. Choix techniques divers

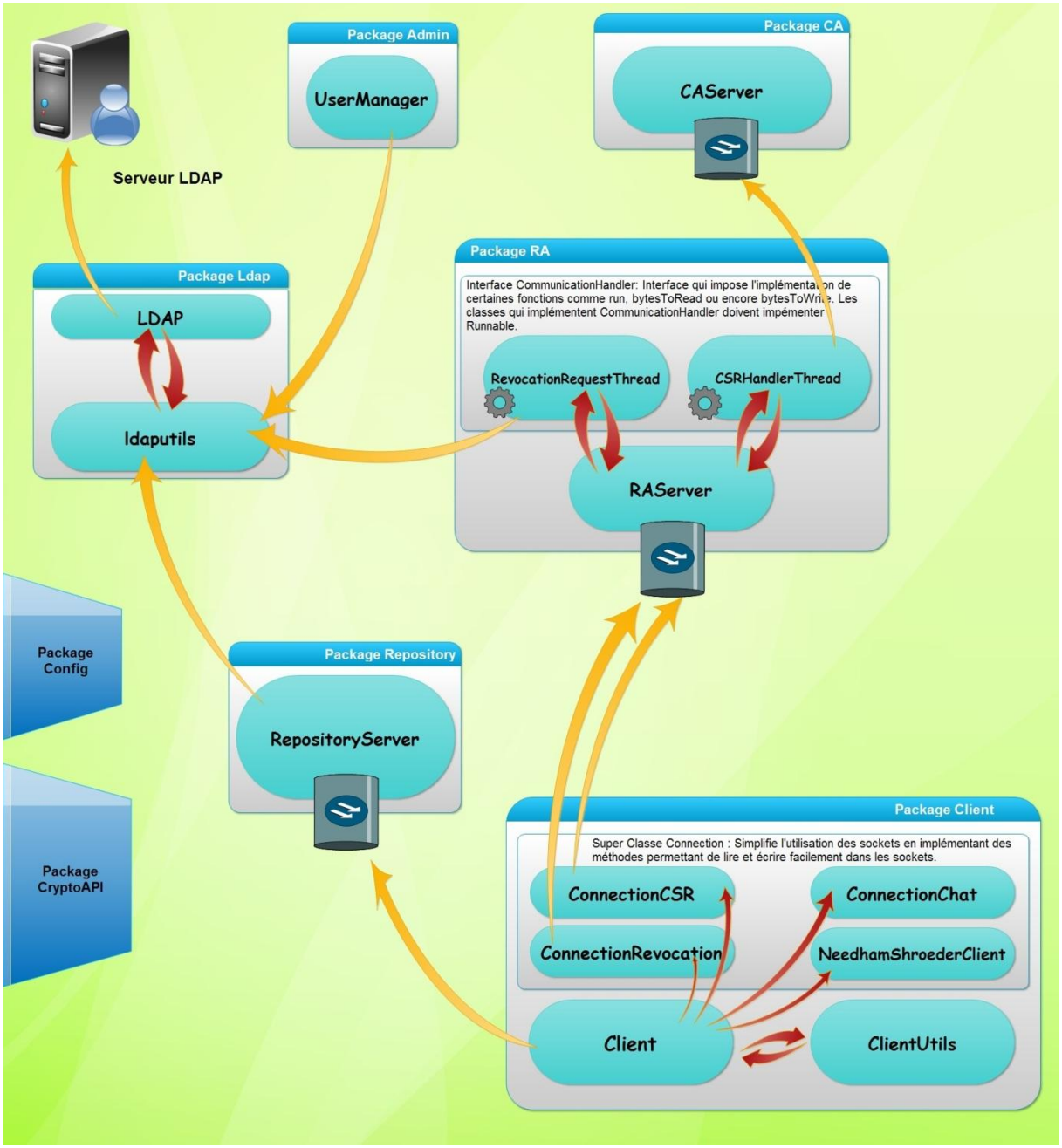
- L'implémentation de ce PKI utilise un fichier de configuration pour fonctionner. Cela permet de configurer chaque paramètres et chaque comportement de chacune des entités et ce via un seul fichier de configuration. Nous évitons ainsi de devoir modifier le code pour obtenir un comportement différent.
- Le CA possède son couple clé privé/certificat mais aussi le couple clé privé/certificat des autorités de certification intermédiaire. Par ce choix aucune autre entité que le CA ne possède de clé privé à risque.
- Le Repository et le RA possède le couple clé privé/certificat du certificat de signature ce qui leurs permet de signer les CRL et les requêtes OCSP.
- Pour mettre à jour une CRL le RA la télécharge la met à jour la signe et la ré-upload.
- La CryptoAPI développé pour ce projet est complètement indépendante et les méthodes génériques. Aucune méthode ne dépend des autres packages du projet ou même du fichier de configuration. Ce choix en fait une librairie flexible et réutilisable telle quelle dans d'autres projets.

2. Architecture des packages et interaction

Le projet s'articule autour des packages suivants :

- Admin : Contient une classe pour la gestion du LDAP et plus particulièrement des utilisateurs.
- CA : Contient le code du CA, qui écoute sur un port les CSR envoyés par le RA
- Clients : Contient toutes les classes relatives au Client et au Chat
- CryptoAPI : Contient toutes les classes pour les diverses opérations cryptographiques.
- Ldap : Contient deux classes qui agissent comme un Wrapper pour l'accès au LDAP
- RA : Contient le code du RA qui agit comme un serveur et attend les CSR et les demandes de révocation du client.
- Repository : Contient le code du Repository dont le rôle et de faire répondeur OCSP
- Setup : Contient la classe permettant la mise en place et la configuration du CA
- Utils : Contient entre autre une classe « Config » permettant l'accès aux attributs du fichier de configuration facilement.

Le diagramme ci-dessous montre de manière global comment interagissent les différentes classes et packages du projet.



Voici le rôle de chacune des classes schématisé ci-dessus :

- **UserManager** : Crée les utilisateurs et les ajoutes dans le LDAP via la classe ldaputils qui possède une méthode « createNewUser » ajoutant l'utilisateur de manière transparente. UserManager permet aussi la suppression d'utilisateur de la même manière.
- **LDAP** : Fournit des méthodes bas niveau pour se connecter à un LDAP récupérer des attributs, ajouter des attributs. Toutes les méthodes de LDAP sont génériques.
- **Ldaputils** : Fournit des méthodes plus haut niveau pour accéder au LDAP parmi celles-ci ont peut noter :
    - createNewUser
    - setCRL
    - setCertificatUser
    - getUserPassword
    - getCertificate
- **CAServer** : Ecoute les connexions du RA en filtrant son adresse IP définit dans le fichier de configuration et signe toute les CSR reçue qui normalement été validée par le RA

- **RAServer** : Ecoute les requêtes des Clients. Pour chaque nouvelle requête suivant que c'est une CSR ou une demande de révocation il crée un nouvel objet respectivement CSRHandlerThread ou RevocationRequestThread qui sont lancé de manières autonomes.
- **RepositoryServer** : Ecoute les requêtes des Clients et pour chaque requêtes OCSP, vérifie que le serial est valide et renvoie une réponse OCSP.
- **CSRHandlerThread** : De manière indépendante de RAServer vérifie la validité de la CSR et si c'est le cas ce connecte au CA envoie la CSR et attend en retour le certificat signé.
- **RevocationRequestThread** : Comme CSRHandlerThread tourne de manière indépendante mais ce thread vérifie que la demande de révocation est valide (login+mdp identique à ceux du LDAP). Si c'est le cas il télécharge la CRL du issue la met à jour et la renvoie sur le LDAP.
- **Client** : Gère toute les interactions avec l'utilisateur notamment le menu présenté plus haut pour savoir quel actions faire. En fonction de l'action à effectuer créer un nouvel objet de la classe associé qui est soit ConnectionCSR, ConnectionRevocation, NeedhamShroederClient, ConnectionChat. Ceci permet de répartir un peu le code du Client pour ne pas surcharger le code du client.
- **ConnectionCSR** : Agit comme un client pour le RA. Elle crée une paire de clé, crée une CSR, l'envoie au RA, envoie le mot de passe de l'utilisateur haché pour l'identifié et récupère le certificat en retour si la demande a été accepté.
- **ConnectionRevocation** : Agit de manière similaire a ConnectionCSR mais pour la révocation. Récupère en retour si le certificat a bien été révoqué ou si une erreur est survenue.
- **NeedhamShroederClient** : Permet d'effectuer la « 3-way handshake » pour authentifier les deux pairs. Elle prend en compte si on joue le rôle de serveur ou client. Retourne si tout s'est bien déroulé.
- **ConnectionChat** : Récupère la clé de session généré au cours de la session NeedhamShroeder et commence une session avec l'interlocuteur. Pour quitter le chat l'utilisateur doit taper « quit » ce qui a pour effet d'envoyer « bye » à l'interlocuteur et fermer la socket. A l'inverse si l'on reçoit un « bye » on l'acquitte en le renvoyant et on ferme aussi la connexion.

3. Protocole de Révocation

a. Formalisation

1 - A -> RA : { A }
2 - RA -> A : { M1} // M1="Ok"
3 - A -> RA : { h(Ka1) } // h(Ka1) est le hash du mot de passe envoyé par A
4 - RA -> LDAP : { A, Kldap }
5 - LDAP -> RA : { h(Ka2), CRL } // h(Ka2) est le mot de passe de A
// Enregistré dans le LDAP
// Ka1 et Ka2 doivent être identique
6 - RA -> LDAP : { CRL U (A) }
7 - RA -> A : {M2} // M2="Done" si le certificat est bien ajouté à la CRL

b. Assertion

Vitalité (A, RA)	Non

Entente faible (A, RA)	Oui
Entente non-injective (A, RA)	Non
Vitalité (RA, A)	Oui
Entente faible (RA, A)	Non
Entente non-injective (RA, A)	Non
Le mot de passe de A ne passe jamais en clair	Oui
Confidentialité du mot de passe du LDAP	Oui

c. Etat de la connaissance

Cn0(A) = { A,RA, }
Cn0(RA) = { RA, LDAP, Kldap }
Cn0(LDAP) = { LDAP, Kldap, CRL, CertA, CertB,..., h(Ka2), h(Kb2),...}

Cn1(RA) = { A, RA, LDAP, Kldap }

Cn2(A) = { M1 } U Cn0(A)
Cn2(RA)={A, M1} U Cn1(A)

Cn3(A) = { h(Ka1) } U Cn2(A)
Cn3(RA) = { h(Ka1) } U Cn2(RA)

Cn4(LDAP) = { RA, A } U Cn1(A)

Cn5(RA) = { h(Ka2), CRL } U Cn3(RA)

Cn6(RA) = { CRL U A } U Cn5(RA)
Cn6(LDAP) = { CRL U A } U Cn4(LDAP)

Cn7(A) = { M2 } U Cn3(A)

4. Protocole de Chat

a. Formalisation

1 - A -> B : { A, Na }Kb
2 - B -> A : { Na, Nb }Ka
3 - A -> B : { Nb }Kb
4 - A -> B : { M1 }Knanb //échange de messages entre A et B chiffré avec
5 - B -> A : { M2 }Knanb // une clé de session généré à partir des nonces.

b. Assertion

Vitalité (A, B)	Non
Entente faible (A, B)	Oui
Entente non-injective (A, B)	Non
Vitalité (B, A)	Oui

Entente faible (B, A)	Non
Entente non-injective (B, A)	Non
Confidentialité des nonces échangées	Oui

c. Etats de la connaissance

Cn0(A) = { A, B, Ka, Ka-1, Kb, Na}
Cn0(B) = { B, Kb, Kb-1, Nb }

Cn1(B) = { A, B, Kb, Kb-1, Ka, Nb, Na, Knbna}

Cn2(A) = { Nb, Knbna } U Cn0(A)

III. Administration et utilisation du PKI

1. Configuration préalable

Tout le comportement du PKI est conditionné par le fichier de configuration appelé « config » qui doit être placé dans le dossier courant à l'exécution du programme ou du .jar.
Le tableau ci-dessous décrit toutes les options disponibles à quoi elles servent ainsi que la valeur utilisé pour notre projet. Il est très important de configurer consciencieusement le fichier de configuration.

	Valeur utilisé pour le projet	Description
LDAP_IP	localhost / 87.98.166.65	IP ou DNS du serveur LDAP
LDAP_PORT	389	Port du serveur LDAP
LDAP_ROOT_DN	dc=pkirepository,dc=org	DN de base
LDAP_ADMIN_DN	cn=admin,dc=pkirepository,dc=org	DN de l'administrateur (pour authentification)
USERS_BASE_DN	ou=intermediatePeopleCA,ou=rootCA, dc=pkirepository,dc=org	DN de là où se trouvent les utilisateurs
CLIENT_CERT_ALIAS	user_certificate	Alias du certificat personnel dans le keystore
CLIENT_KEY_ALIAS	user_private	Alias de la clé privée personnel dans le keystore
CLIENT_KEY_PASS	monpassP1	Mot de passe de la clé privé dans le keystore
KS_PATH_USER	keystores/user_keystore.ks	Chemin d'accès au keystore pour les clients
KS_PASS_USER	passwd	Mot de passe du keystore pour les clients
PORT_LISTEN	7000	Port d'écoute lorsque l'on lance un chat en tant que serveur
IP_CA	localhost / 87.98.166.65	IP du CA
PORT_CA	7001	Port du CA
KS_PASS_CA	passwd	Mot de passe du keystore du CA
KS_PATH_CA	keystores/ca_keystore.ks	Chemin d'accès au keystore du CA
IP_RA	localhost / 87.98.166.65	IP du RA
PORT_RA	7002	Port du RA
KS_PASS_RA	passwd	Mot de passe du keystore du RA
KS_PATH_RA	keystores/ra_keystore.ks	Chemin d'accès au keystore du RA
IP_REPOSITORY	localhost / 87.98.166.65	IP du Repository
PORT_REPOSITORY	7003	Port du Repository
KS_PASS_REPOSITORY	passwd	Mot de passe du keystore du Repository
KS_PATH_REPOSITORY	keystores/repo_keystore.ks	Chemin d'accès au keystore du Repository
KS_ALIAS_CERT_CA	CA_Certificat	Alias du certificat du CA Root
KS_ALIAS_KEY_CA	CA_Private	Alias de la clé privé du CA Root
KS_ALIAS_CERT_CA_INTP	CA_IntermediairePeople_Certificate	Alias du certificat du CA intermédiaire d'utilisateurs

KS_ALIAS_KEY_CA_INTP	CA_IntermediairePeople_Private	Alias de la clé privé du CA intermédiaire d'utilisateurs
KS_ALIAS_CERT_CA_INTS	CA_IntermediaireServer_Certificate	Alias du certificat du CA intermédiaire pour serveurs
KS_ALIAS_KEY_CA_INTS	CA_IntermediaireServer_Private	Alias de la clé privé du CA intermédiaire pour serveurs
KS_ALIAS_CERT_CA_SIG	CA_SigningOnly_Certificate	Alias du certificat de signature
KS_ALIAS_KEY_CA_SIG	CA_SigningOnly_Private	Alias de la clé privé du CA de signature
PASSWORD_CA_ROOT	passwordRootCA	Mot de passé de la clé privé du CA Root
PASSWORD_CA_INTP	passwordPeopleCA	Mot de passé de la clé privé du CA intermédiaire d'utilisateurs
PASSWORD_CA_INTS	passwordServerCA	Mot de passé de la clé privé du CA intermédiaire pour serveurs
PASSWORD_CA_SIG	passwordSigningCert	Mot de passe de la clé privé du CA de signature

La plupart des attributs peuvent être utilisés avec la valeur proposée. Les seuls qui sont vraiment critiques sont les adresses IP, les ports et les chemins pour les keystore ou le LDAP. A noter qu'il est conseillé de renseigner chacun des attributs.

2. Configuration du LDAP

La configuration du LDAP est de loin l'opération la plus délicate pour faire fonctionner le PKI. Le serveur LDAP utilisé est OpenLDAP. Voici les différentes étapes pour installer et configurer le LDAP tel qu'il l'a été pour le projet. A noter que toute les manipulations qui suivent doivent être faites en administrateur.

a. Installation de OpenLDAP

La première chose à faire est d'installer OpenLDAP avec toute ses dépendances. Sous Linux il faut taper la commande : apt-get install slapd ldap-utils libldap2 libdb4.6.
Si openldap est déjà installé il faut le reconfigurer avec la commande suivante :
dpkg-reconfigure slapd

Il faut répondre aux questions comme suit :

```
Omit OpenLDAP server configuration? No
DNS domain name: pkirepository.org
Organization name? pkirepository.org
Administrator password: [MOT DE PASSE]
Confirm password: [MOT DE PASSE]
Database backend to use: BDB
Do you want the database to be removed when slapd is purged? No
Allow LDAPv2 protocol? No
```

b. Configuration du ldap.conf

Maintenant OpenLDAP il faut modifier son fichier de configuration pour pouvoir le paramétrer. Voici la configuration utilisée pour le serveur LDAP. Sur une architecture Unix il se trouve normalement dans /etc/sldap/ldap.conf.

```
# See ldap.conf(5) for details
# This file should be world readable but not world writable.

BASE    dc=pkirepository,dc=org
URI     ldap://127.0.0.1 ldap://127.0.0.1:666

#SIZELIMIT      12
#TIMELIMIT      15
#DEREF          never

include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/person.schema
#include        /etc/ldap/schema/inetorgperson.schema

loglevel 256

index       objectClass eq
index       cn      eq
index       sn      eq

access to attrs=userPassword
    by dn="cn=admin,dc=pkirepository,dc=org" write
    by * none

access to *
    by dn="cn=admin,dc=pkirepository,dc=org" write
    by * read
```

Le plus important à noter ici est les droits accordés aux différents attributs du ldap.

```
access to attrs=userPassword
    by dn="cn=admin,dc=pkirepository,dc=org" write
    by * none
```

Cette partie signifie que pour l'attribut userPassword, on autorise l'attribut en écriture pour l'admin et on le refuse pour tous les autres. En effet les utilisateurs anonymes qui se connectent au site pour récupérer un certificat ne doivent pas pouvoir avoir accès au mot de passe des utilisateurs même si celui-ci est haché.

Puis :

```
access to *
    by dn="cn=admin,dc=pkirepository,dc=org" write
    by * read
```

Ce passage autorise tout le reste du LDAP en lecture pour tout le monde et toujours en write pour l'admin.

Enfin il faut redémarrer le service pour que les changements soient pris en compte :
service slapd restart

c. Ajout de la structure de base

Le LDAP est maintenant installé et configuré seulement il est vide. Il faut donc y ajouter un arbre. Voici le schema de l'arbre utilisé pour le LDAP. Il se trouve dans « structure.ldif ».

```
dn: ou=rootCA,dc=pkirepository,dc=org
ou: rootCA
objectClass: top
objectClass: organizationalUnit
objectClass: cRLDistributionPoint
objectClass: pkiCA

dn: ou=intermediatePeopleCA,ou=rootCA,dc=pkirepository,dc=org
ou: intermediatePeopleCA
objectClass: top
objectClass: organizationalUnit
objectClass: cRLDistributionPoint
objectClass: pkiCA

dn: ou=intermediateServerCA,ou=rootCA,dc=pkirepository,dc=org
ou: intermediateServerCA
objectClass: top
objectClass: organizationalUnit
objectClass: cRLDistributionPoint
objectClass: pkiCA

dn: ou=signingCA,ou=rootCA,dc=pkirepository,dc=org
ou: signingCA
objectClass: top
objectClass: organizationalUnit
objectClass: pkiCA
```

Pour ajouter ce fichier au ldap il suffit de taper les commandes suivantes :
service slapd stop
slapadd -c -v -l structure.ldif
service slapd start

Le LDAP est maintenant opérationnel, pour recevoir les utilisateurs et les certificats.

3. Configuration avec le setup

Maintenant que le fichier de configuration est bien renseigné et que le LDAP est opérationnel il faut lancer le setup qui permet de configurer un Client et/ou l'ensemble CA/RA et Repository. Voici ce que fait la configuration du CA/RA/Repository :

- Création du certificat root auto signé ajout dans le keystore du CA avec la clé privé. + envoie du certificat sur le LDAP avec une CRL vide.
- Création du certificat pour la signature, ajout dans le keystore du RA et du Repository + envoie sur le LDAP.
- Création des deux certificats intermédiaires et ajout des deux couples certificats/clé privé dans le keystore du CA. + ajout des deux certificats sur le LDAP avec une CRL vide signé par le CA de signature.

Ce qui se passe lors de la configuration du client est la suivante :

- Ajout du certificat root, du certificat intermédiaire et du certificat de signature dans le keystore du client.

A noter que la configuration du client ne lui crée nullement un certificat. Il devra faire la demande manuellement dans le programme Client.

La configuration de l'architecture est maintenant terminée. Et elle est complétement opérationnelle.

4. Opération d'administration et utilisation du PKI

L'administration du serveur LDAP peut se faire directement en se connectant avec un client LDAP tel que Jxplorer, mais l'architecture fournis un programme appelé UserManager qui permet l'ajout et la suppression d'utilisateur sur le LDAP. Bien entendu l'utilisation de ce programme nécessite le mot de passe administrateur du LDAP.

L'utilisation du PKI est très simple il suffit de lancer le ou les services dont on souhaite disposer. Ils sont tous indépendant bien que si on lance le RA sans lancer le CA aucune CSR ne pourra être signé. Une fois le fichier de configuration renseigné et les services lancés il suffit de lancer le Client qui va permettre de dialoguer avec l'interlocuteur.

IV. Conclusion

Ce projet aussi intéressant qu'instructif a été l'occasion pour nous de mettre en application tout ce qui a été vu en cour dans l'application d'un projet concret. Les connaissances acquises sur la programmation java orienté sur la sécurité ont été indispensables pour ce projet.

Cela nous a permis d'apprendre à maitriser la librairie BouncyCastle en mettant en application beaucoup de principes liés à la cryptographie (Chiffrement, signature, digest, chiffrement asymétrique..).

Cela a aussi été l'occasion de travailler sur l'implémentation concrète d'un protocol cryptographique de ce cas-là Needham Shroeder. Mais il a aussi été enrichissant car l'on a pour la première fois mis en service un serveur LDAP pour une application concrète, le tout en gardant à l'idée le contexte de mis en œuvre. Dans notre cas le contexte est la mise en place d'un PKI dans une entreprise possédant un LDAP et qui pour des raisons de sécurité veut permettre à ses employés de disposer d'un certificat pour chater ou encore envoyé des mails chiffrés/signé.

De plus le projet a été développé dans un souci de modularité, ainsi il peut facilement s'interfacer avec un LDAP existant, la CryptoAPI peut être utilisé pour d'autres projets, et tout le code respecte les dernières recommandations en n'utilisant aucune méthode dépréciée !