LDAP and certification

2

Logiciels-
Réseaux

Maryline Maknavicius
Michel Gardie

04001-LOR
2004

Janvier 2004

LDAP AND CERTIFICATION

---oOo---

Abstract : The objective of this report is to present notions related to certificates (generation, d and puiblishing), the LDAPv3 protocol, and the possibilities to use LDAPv3 as a public key ce publishing service. This report was financially supported by Groupe des Ecoles de Télécommunications (GET) under project CADDISC (Combining LDAP and DNSsec to distribute keys securely).

Key words: Key publishing, LDAP, PKI, certificate, CRL, certificate revocation

Résumé : Le but de ce document est de présenter les notions sur la certification (génération distribution et publication de certificats), le protocole LDAPv3, et les possibilités d'exploiter LDAPv3 pour un service de publication de certificats de clés publiques. Il a été financé par le Groupe des Ecoles de Télécommunications (GET) dans le cadre du projet CADDISC (Couplage des Annuaires LDAP et DNSsec pour la Distribution Sécurisée de Clés).

Mots-clés : Publication de clés, L DAP, IGC, ICP, certificat, CRL, révocation de certificat

---oOo—

Maryline Laurent-Maknavicius
Maître de Conférences
Institut National des Télécommunications
Département LOR
9 rue Charles Fourier 91011 Evry cedex
E-mail: Maryline.Maknavicius@int-evry.fr

Michel Gardie
Ingénieur de recherche
Institut National des Télécommunications Département LOR
9 rue Charles Fourier 91011 Evry cedex
E-mail: Michel.Gardie@int-evry.fr

Janvier 2004

LDAP et la certification

Maryline Laurent-Maknavicius

Maryline.Maknavicius@int-evry.fr

INT

Michel Gardie

Michel.Gardie@int-evry.fr

INT

Janvier 2004

SOMMAIRE

1 Introduction

La cryptographie asymétrique (dite aussi « à clés publiques ») sert à de multiples usages : la signature d'un document électronique, l'authentification d'un serveur web lors de la mise en place d'une session SSL/TLS… Cette cryptographie repose sur la génération d'un bi-clé pour chaque entité (e.g. utilisateurs, machines) :

- Une clé privée détenue par une entité et à maintenir secrète par cette entité. C'est cette clé qui permettra à cette entité de générer une signature électronique lui permettant de s'authentifier ;
- Une clé publique à largement diffuser sur le réseau. Cette clé sert à vérifier la validité d'une signature. Cela consiste à déchiffrer la signature avec la clé publique du signataire et à vérifier la cohérence du résultat vis à vis de l'information signée.

Pour permettre à n'importe quelles entités de s'authentifier mutuellement, il est nécessaire de publier les clés publiques de toutes les entités, et ce de façon sûre. Les certificats électroniques permettent de garantir électroniquement l'association clé publique – entité. Ils sont générés par des Autorités de Certification (CA) de confiance, parfois organisées entre elles dans une infrastructure appelée PKI *(Public Key Infrastructure)*, IGC (Infrastructure de Gestion de Clé) ou encore ICP (Infrastructure à Clés Publiques).
Dans ce document, nous décrivons les aspects liés à la certification dans le chapitre 2. Le chapitre 3 présente brièvement LDAPv3 avec les services de sécurité offerts, et le chapitre 4 une analyse sur la possibilité de rendre un service de publication de certificats sécurisé par LDAP.

2 Aspects liés à la certification

2.1 Définition d'un certificat

Un certificat est un document électronique, résultat d'un traitement fixant les relations qui existent entre une clef publique, son propriétaire (une personne, une application, un site) et l'application pour laquelle il est émis :

- pour une personne il prouve l'identité de la personne au même titre qu'une carte d'identité, dans le cadre fixé par l'autorité de certification qui l'a validé ;
- pour une application il assure que celle-ci n'a pas été détournée de ses fonctions ;
- pour un site il offre la garantie lors d'un accès vers celui-ci que l'on est bien sur le site auquel on veut accéder.

On ne parlera dans ce document que des certificats qui s'appuient sur un protocole normalisé X.509 (ITU-T X.509 international standard V3 - 1996) *(RFC2459)*. Il existe des applications ou des systèmes de cryptologie qui ne s'appuient pas sur les certificats X.509 (PGP par exemple).
Le certificat est signé (au sens signature électronique) par le CA émetteur. C'est-à-dire, lors de la génération de cette signature, il va utiliser sa clé privée pour prouver qu'il en est à l'origine. Pour vérifier la signature, il suffira de connaître la clé publique du CA ;

2.1.1 Contenu d'un certificat

Le certificat contient un certain nombre de champs obligatoires et des extensions dont certaines sont facultatives :

- **Version :** indique à quelle version de X.509 correspond ce certificat
- **Numéro de série :** numéro de série du certificat
- **Algorithme de signature :** identifiant du type de signature utilisée
- **Emetteur :** *Distinguished Name* (DN) de l'autorité de certification qui a émis ce certificat
- **Valide à partir de :** la date de début de validité du certificat
- **Valide jusqu'à :** la date de fin de validité du certificat
- **Objet :** *Distinguished Name* (DN) du détenteur de la clef publique
- **Clé publique :** valeur de la clef publique de ce certificat
- **Extensions X.509v3 :** extensions génériques optionnelles, introduites avec la version 3 de X.509
- **Signature** : Signature numérique du CA sur l'ensemble des champs précédents (générée à l'aide de la clé privée du CA)

Le certificat contient également des extensions qui permettent de spécifier un certain nombre d'informations en fonction de l'usage prévu d'un certificat. Les extensions ci-dessous ne sont pas exhaustives :

- X509v3 Basic Constraint : indique s'il s'agit du certificat d'une Autorité de Certification ou non, c'est à dire permettant d'émettre des certificats
- X509v3 Key Usage : Donne une ou plusieurs fonctions de sécurité auxquelles la clé publique est destinée. Ce champ permet de spécifier plusieurs services de sécurité
- X509v3 subjectAltName: Ce champ contient un ou plusieurs noms alternatifs pour le porteur de certificat, exprimé sous diverses formes possibles.
- X509v3 issuerAltName: Ce champ contient un ou plusieurs noms alternatifs pour le CA qui a généré ce certificat, exprimé sous diverses formes possibles.
- X509v3 CRL Distribution Points : adresse de la CRL (Certificate Revocation List) permettant de connaître le statut de ce certificat

Une extension dans un certificat peut être qualifiée de critique ou non. Le fait qu'une extension soit critique rend obligatoire la conformité du certificat aux informations contenues dans l'extension. Si une application traitant un certificat ne reconnaît pas une extension contenue dans ce certificat et marquée comme critique, ce certificat doit être déclaré invalide par l'application. En particulier les extensions *key Usage* et *Basic Constraint* doivent être marquées critiques.

Avant que la norme X.509v3 n'ait été créée, Netscape a introduit plusieurs extensions propriétaires permettant de limiter l'usage d'un certificat.

Ces extensions propriétaires sont toujours utilisables, et parfois même nécessaires avec Netscape. Elles apparaissent dans la partie *X509v3 Extensions* du certificat.

2.2 Autorité de Certification

Une Autorité de Certification (CA) est une organisation qui délivre des certificats électroniques. Un CA possède lui-même un certificat (autosigné ou délivré par un autre CA) et utilise sa clé privée pour créer les certificats qu'il délivre.
N'importe qui peut se déclarer Autorité de Certification. Un CA peut être organisationnel, spécifique à un corps de métiers (exemple : notaires) ou encore institutionnel.
Selon le crédit accordé au CA, les certificats délivrés auront un champ d'applications plus ou moins importants :

- limité à l'intérieur d'un organisme
- échanges inter-organismes
- etc.

En délivrant un certificat, le CA se porte garant de l'identité de l'entité qui se présentera avec ce certificat. Par rapport aux entités (personnes ou applications) qui utilisent ses certificats, un CA joue le rôle de tiers de confiance.
Le niveau de garantie offert par un CA dépendra des moyens mis en œuvre pour s'assurer de l'identité des demandeurs, de la sécurité mise en œuvre pour la protection de la clé privée du CA, etc.

2.3 Certification croisée et certification hiérarchique

Un CA peut cautionner un autre CA en lui générant un certificat portant sur sa clé publique. On peut ainsi mettre en place des relations de confiance hiérarchiques ou croisées entre CAs.
Dans le cas d'une certification hiérarchique (fig.1), un CA dit racine délivre un certificat à un ou plusieurs autres CAs, qui eux-mêmes peuvent délivrer un certificat à d'autres CAs et ainsi de suite.

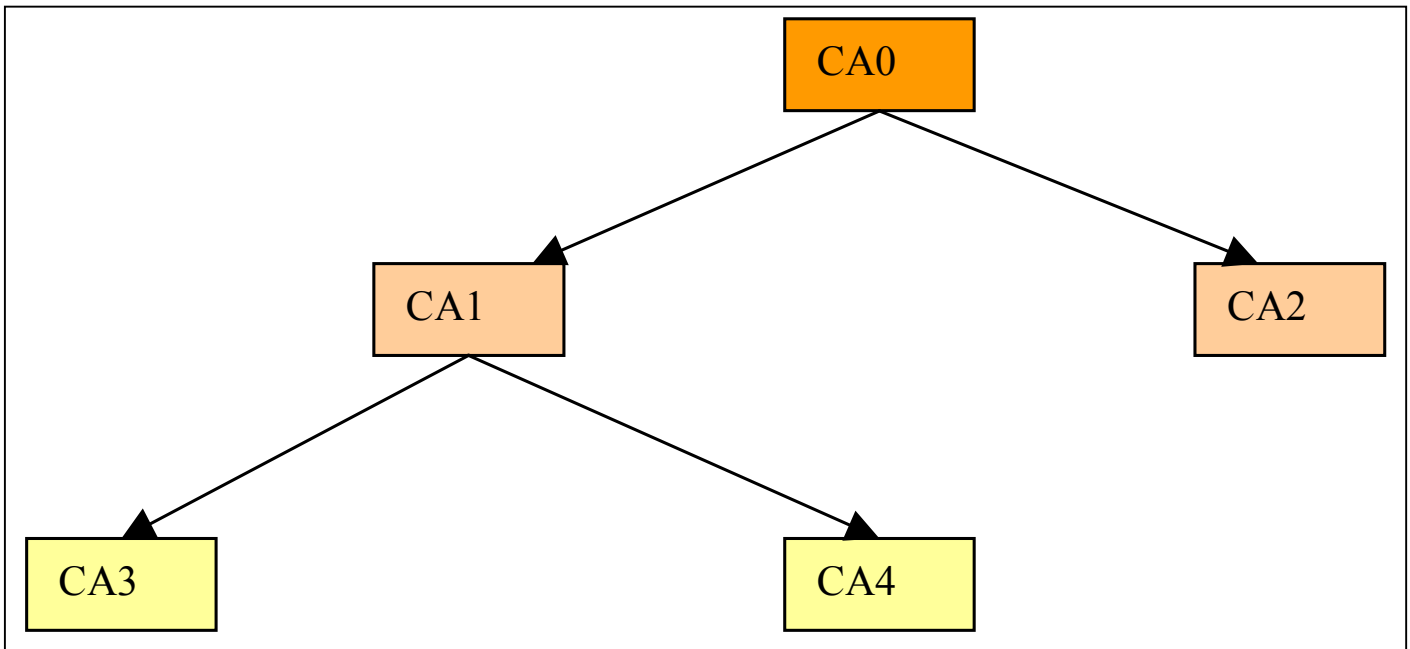

Figure 1 : Hiérarchie de CAs

Au sommet de ces arborescences, on trouve les CAs racines dont le certificat est signé avec leur propre clé privée (certificat auto-signé).
Dans le cas de la figure ci-dessus, pour valider un certificat issu du CA CA4, il faut les certificats des Autorités de Certification CA0, CA1 et CA4. On appelle cela une **chaîne de certification**.
En général dans ce type de schéma, seuls les CAs "feuilles" délivrent des certificats à des personnes ou à des services.
La confiance accordée à un CA est héritée par tous ses CAs "filles". Ainsi si l'on décide de faire confiance à un CA racine, cette confiance sera également accordée à l'ensemble des CAs appartenant à la même hiérarchie.
La certification croisée est utile lorsque des organismes O1 et O2 ayant chacun leur propre CA, CA1 et CA2, n'appartenant pas à une même arborescence, veulent se faire confiance. Dans ce cas, CA1 et CA2 peuvent chacun créer un certificat en signant la clé publique de l'autre. Ainsi un utilisateur de O1 pourra vérifier le certificat d'une personne de O2.

2.4 Infrastructure à Clés Publiques

L'ICP est constituée de l'ensemble des matériels, logiciels, personnes, règles et procédures nécessaire à une Autorité de Certification pour créer, gérer, distribuer et publier des certificats X.509.
Une ICP est donc une structure à la fois technique et administrative permettant une mise en place, lors de l'échange de clef, de ***relations de confiance entre des entités morales et/ou physiques et/ou logique*s**.
Les fonctions principales d'une ICP sont :
- Émettre et révoquer des certificats
- Publier les certificats dans un annuaire
- Éventuellement, fournir un service de dépôt et de recouvrement des clés privées

Elle est constituée par :
- Une autorité de certification (**CA**)
- Une autorité d'enregistrement (**RA**)
- Un annuaire de publication de certificats
- Un service de validation
- Éventuellement, un service de dépôt de clés

2.4.1 Autorité d'Enregistrement

Le RA a pour tâche principale de recevoir et de traiter les demandes de création, de renouvellement et de révocation de certificats.
Pour cela, il doit :
- assurer le contrôle d'identification du demandeur de certificat,
- valider les demandes de révocation,
- assurer lors de la délivrance d'un nouveau certificat (sur date de péremption atteinte) un recouvrement des certificats afin d'assurer la continuité pour la fonctionnalité signature et/ou chiffrement.

2.4.2 Autorité de certification

L'Autorité de Certification prend en charge toutes les opérations nécessitant la clé privée du CA : création et distribution sécurisée des certificats, révocation, production de cartes à puces…
Le CA gère en collaboration avec l'autorité d'enregistrement les cycles de vie des certificats. En fonction de la politique de certification, il peut générer les bi-clefs pour le compte des utilisateurs.
Pour des raisons de sécurité, le CA n'est en général pas connecté au réseau. En effet la compromission de la clef privée du CA étant le risque majeur dans une ICP, tout doit être fait pour l'éviter. Cela entraîne en particulier que le transfert des requêtes entre le RA et le CA n'est pas automatisée.
En outre, la sécurisation physique du CA doit également être étudié avec soin.

2.4.3 Annuaire de publication

Les certificats émis par une ICP doivent être rendus publics afin que les différents partenaires qui les utilisent puissent s'échanger leur clé publique. Pour cela, les certificats sont publiés dans un annuaire d'accès libre.
Cet annuaire peut également contenir le certificat du CA et les CRLs *(Certificate Revocation List)*. Des annuaires LDAP sont généralement utilisés pour cette fonction.

2.4.4 Révocation de certificats

Même si la date de péremption du certificat n'est pas atteinte, un certificat peut devenir invalide pour différentes raisons :

- perte ou vol de la clé privée associée
- fin de mandat
- etc.

Le cas de la date de péremption est traité par le fait que les certificats contiennent les dates de début et de fin de validité.
Dans les autres cas, l'ICP doit procéder à la *révocation* des certificats concernés.
L'utilisateur (service ou personne) d'un certificat doit donc avoir un moyen de vérifier que ce certificat est valide à un instant donné. Pour cela, une ICP doit fournir un service de validation permettant à tout instant de vérifier la validité des certificats qu'elles délivrent.
La méthode la plus employée jusqu'à maintenant est la publication d'une liste appelée *Certificate Revocation List* (CRL) qui est la liste des numéros de série des certificats révoqués.
Un CRL a un format normalisé et est signé à l'aide de la clé privée du CA émetteur. Il peut être subdivisé en plusieurs CRLs, ceci pour des raisons de performances. Il peut être publié via le même annuaire de publication que les certificats.
La technique des CRL s'avère peu optimisée que ce soit pour des raisons de performances (volumes des certificats à gérer) ou d'efficacité. En effet les CRLs ne permettent pas de diffuser rapidement l'information de révocation d'un certificat, ce qui peut-être très préjudiciable dans le cas d'applications nécessitant un haut niveau de sécurité.

Cette technique devrait être progressivement remplacée par un nouveau protocole : **OCSP** *(Online Certificate Status Protocol)* [rfc2560]. Un client OCSP pourra vérifier la validité d'un certificat donné en interrogeant un serveur OCSP. Cela permettra la diffusion quasi instantanée de l'information concernant la révocation d'un certificat.

2.4.5 Politiques de Certification

De même que la sécurité se met en place en suivant une politique de sécurité définie préalablement, la mise en place d'une ICP oblige à une ***définition de politiques de certification*** : « un ensemble de règles indiquant, ce pour quoi le certificat est applicable et par qui, et quelles sont les conditions de leur mise en œuvre au sens juridique, administratif et technique ».
La règle de base étant avant tout que les certificats et les moyens de mise en œuvre soient définis en fonction de l'utilisation que l'on veut en faire. Les facteurs principaux à prendre en compte sont :

- Étude de la population/des utilisateurs à qui est destiné le certificat en tenant compte à la fois des caractéristiques des utilisateurs, de l'utilisation qui sera faite du certificat (signature, chiffrement entre entités morales et/ou physiques, accès à des applications sécurisées) et de la mise en place des critères d'attribution.
- Étude des moyens de collecte des informations, de leur validation et de la création des certificats.
- Définition de la durée de vie des clefs (privée, publique et/ou de session), des certificats, de la consolidation de ceux-ci, de la gestion des listes de révocation.
- Étude des moyens de distribution des certificats via des communications sécurisées de type «VPN» ou sur un support style « carte de crédit » avec récupération en main propre ou par un agent de sécurité sur site.
- Sécurité des ICP au sens implantation physique, et sécurité des annuaires supports des informations publiques en tenant compte de l'infrastructure, de l'administration et du coût de gestion.
- Définition des services nécessitant une haute disponibilité (exemple : gestion des listes de révocation).
- Prise en compte de la nécessité d'un recouvrement des clefs privées et de l'interaction avec l'autorité suprême et/ou avec d'autres communautés (interopérabilité pour certifications croisées).
- Étude du support matériel/logiciel du certificat chez l'utilisateur en tenant compte de la vétusté des postes de travail et en prévoyant des évolutions aisées.
- Prise en compte de l'impact sur les structures existantes : physiques et organisationnelles.
- Définition de la formation/information des acteurs.

Tout ceci sera transcrit dans un document qui constituera la Politique de Certification (PC) du CA. C'est la PC qui définira le niveau de sécurité associé à un certificat. Par exemple, si la PC spécifie que pour obtenir un certificat, une personne devra se présenter avec une pièce d'identité, la valeur du certificat obtenu sera plus grande que si on se contentait de vérifier l'adresse électronique de la personne.

La Déclaration des Pratiques de Certification (DPC) a pour but de décrire les détails des processus techniques mis en œuvre au sein des différentes composantes de l'ICP (CA, RA,...), conformément à la PC.
En résumé, la PC spécifie des objectifs de sécurité qu'il est nécessaire d'atteindre pour la sécurisation de l'application utilisatrice des services de l'ICP, alors que la DPC spécifie les moyens mis en œuvre pour atteindre ces objectifs.
Les politiques de certifications peuvent permettre d'établir des normes communes d'interopérabilité et des critères d'assurance communs entre plusieurs organismes.

2.4.6 Protection des clés privées

La mise en place d'une ICP dans le but de déployer des applications utilisant les certificats X.509 n'a de sens que si un niveau de confiance suffisant peut être établi entre les différents protagoniste.
Vis à vis de l'ICP elle-même, la confiance est apportée par sa politique de certification et les moyens mis en œuvre pour la faire respecter.
Le point faible concerne les possesseurs de certificats, car ce sont eux qui ont la responsabilité de la protection de leur certificat et de la clé privée associée.
En effet si on veut pouvoir faire confiance à un certificat, il est nécessaire que la clé privée associée soit correctement protégée. Ceci est vrai aussi bien pour les certificats de personnes que pour ceux de serveurs.
Concernant les certificats de personnes, il est absolument indispensable de mettre en place une formation dans le but de faire comprendre aux utilisateurs ce qu'est un certificat, sa fonction et pourquoi il faut protéger la clé privée associée.
Le choix du support physique de la clé privée associée dépendra du niveau de sécurité que l'on veut atteindre :

- un fichier sur disque dur

Cette solution a l'avantage d'être facile à mettre en œuvre. Mais elle implique que le niveau de sécurité du certificat et de la clé privée sera directement dépendant du niveau de sécurité de l'ordinateur (poste de travail ou serveur) sur lequel ils seront installés. De plus le certificat et la clé privée peuvent facilement être installés sur différents postes de travail.

- une carte à puce

Cette solution, plus chère, matérialise le certificat et la clé privée sous la forme d'une carte. Elle facilite l'"éducation" des utilisateurs qui associe mieux le certificat à une pièce d'identité personnelle.
Il est clair que dans le cas d'applications sensibles, seule une solution à base de cartes à puce pourra apporter un niveau de sécurité raisonnable.

3 LDAPv3

Ce chapitre décrit succinctement le protocole d'annuaire standardisé par l'IETF.

3.1 Historique du protocole

La forme des annuaires électroniques a fortement évolué depuis leur apparition au début de l'ère informatique. Citons-en quelques uns :

- Unix : /etc/passwd (années 70-80)
- DNS (1984)
- X.500 (1988-1993-1997)
- LDAP (1993)

3.1.1 X.500

Ce standard a été conçu par des opérateurs de télécommunications. Son objectif : fédérer les annuaires de chaque opérateur, et ainsi offrir un annuaire global. Ses principaux défauts sont une certaine lourdeur (il offre malencontreusement beaucoup trop de fonctionnalités), et l'obligation de s'appuyer sur une pile de protocoles OSI. Il est en effet obligatoire d'utiliser conjointement avec le protocole d'annuaire les protocoles suivants : ACSE, X.226, X.225 et X.224.
Le protocole de base de X.500 s'appelle DAP (*Directory Access Protocol* ; protocole d'accès à l'annuaire).

3.1.2 LDAP

LDAP signifie *Light-weight Directory Access Protocol* (protocole allégé d'accès à l'annuaire). Le protocole LDAP est né de l'adaptation de X.500 ; il est plus simple ; il est orienté TCP/IP.
LDAP a été initialement conçu comme frontal X.500. L'objectif était de permettre l'accès à des annuaires X.500 à partir d'équipements n'intégrant pas la totalité des couches ISO, ou ne les mettant pas du tout en œuvre.
Le protocole devient annuaire natif (Université Michigan) en 1995.
En 1996 commence l'apparition des premiers serveurs commerciaux.

3.2 Protocole LDAP

LDAP n'est pas seulement un protocole standardisé, mais c'est aussi un ensemble de concepts, de modèles, de définitions, de services. Parmi les principaux modèles que l'on peut associer à LDAP, nous avons le modèle de nommage, le modèle d'information, le modèle fonctionnel.

3.2.1 Modèle de nommage (le distinguished name)

Le modèle de nommage définit comment sont organisées et identifiées les entrées de l'annuaire. Le plus simple pour comprendre comment cela fonctionne, est de partir d'un exemple simple. Nous allons essayer de concevoir l'annuaire interne de l'INT. Nous ne nous intéresserons dans un premier temps qu'aux personnes physiques et aux entités administratives de l'institut.
Prenons un cas particulier : l'un des auteurs de ce document est Michel Gardie et il travaille au sein du département Logiciel-Réseaux de l'INT. Il a un numéro de téléphone, une adresse électronique, et son bureau porte le numéro A107-01. Toutes ces informations (nom, prénom, identifiant du département, numéro de téléphone, etc.) sont regroupées dans une des entrées de l'annuaire. Le département Logiciel-Réseaux, quant à lui, possède un nom, un numéro de téléphone (celui du secrétariat), un numéro de fax, etc. Ces informations sont également regroupées dans une des entrées de l'annuaire. Le

département est également dépendant d'une entité administrative plus importante, en l'occurrence l'INT lui-même.
Si un utilisateur de l'annuaire souhaite joindre Michel Gardie par téléphone, il lui faudra simplement connaître son nom. L'annuaire devra ensuite accéder à l'entrée le concernant pour retourner le numéro de téléphone permettant de le joindre. Pour que l'annuaire puisse accéder à une entrée, il faut donc :

- un nom pour identifier l'entrée ;
- un chemin pour la trouver dans la base de données.

L'identification d'une entrée se fait à l'aide de son nom distingué ; en anglais : *distinguished name* (DN).
Le chemin pour trouver l'entrée dépend de l'organisation de la base. La base de données est organisée de manière arborescente. L'arbre ainsi formé de toutes les entrées porte le nom de *Directory Information Tree* (DIT). Au sommet de l'arbre se trouve le suffixe ou la racine de la base (les deux termes sont équivalents).
Dans le cas de notre exemple, le schéma arborescent des données (figure 2) indique que l'entrée « Michel Gardie » est rattachée à l'entrée « LOR » qui est elle-même dépendante de la racine (ici, l'entrée « INT »).
Le DN représente le nom de l'entrée sous la forme du chemin d'accès à celle-ci depuis le sommet de l'arbre, mais présenté de manière inverse.
Dans le cas de notre exemple, le *distinguished name* correspondant à l'entrée « Michel Gardie » pourra être :
cn=Michel Gardie, ou=LOR, o=INT, c=FR
L'entrée « LOR » aura, quant à elle, le DN suivant : *ou=LOR, o=INT, c=FR*
Et l'entrée « INT », correspondant à la racine de l'arbre, aura pour DN : *o=INT, c=FR*
Il est important de noter que la création de l'entrée « Michel Gardie » nécessite la création préalable de l'entrée « LOR », ainsi que bien entendu la création de la racine (c'est-à-dire l'entrée « INT »).

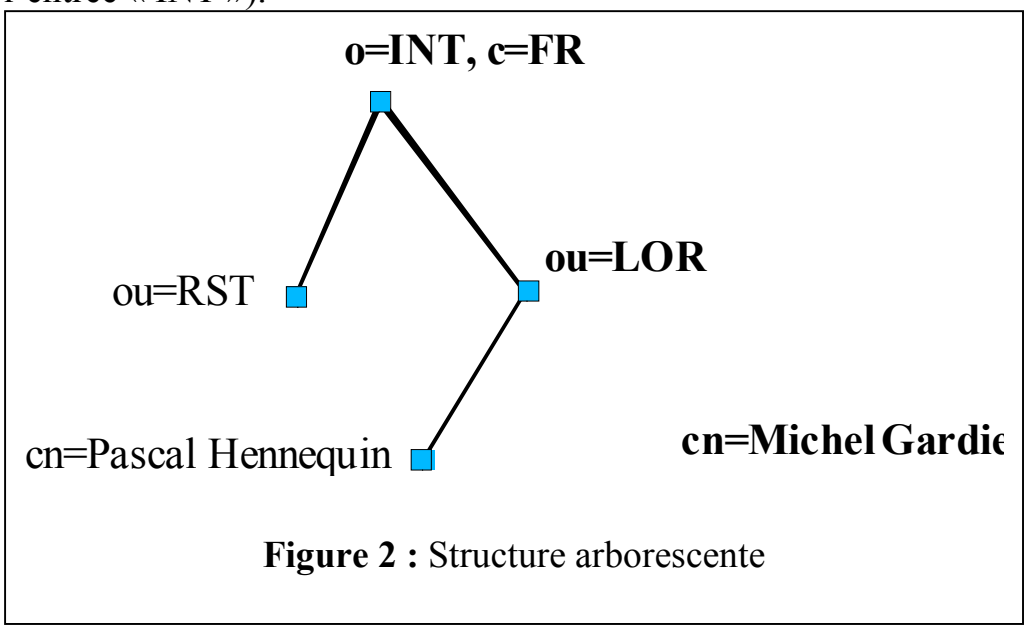

Figure 2 : Structure arborescente

3.2.2 Modèle d'information (les entrées)

Le modèle d'information définit le type de données pouvant être stockées dans la base d'information. Les données sont stockées sous forme d'*entrées*. Les entrées correspondent à des objets abstraits ou issus du monde réel, comme une personne, une imprimante, ou des paramètres de configuration. Elles contiennent un certain nombre de champs appelés *attributs* dans lesquelles sont stockées des valeurs.
Chaque entrée possède au moins une classe. Une classe LDAP permet de préciser les attributs qui sont obligatoires et/ou ceux qui sont optionnels pour une entrée particulière.

3.2.2.1 Entrées opérationnelles

Chaque serveur possède une entrée opérationnelle, appelée *root directory specific entry* (root DSE), qui contient la description de l'arbre et de son contenu. Cette entrée se situe au niveau de la racine du serveur. Voici un exemple de contenu de cette entrée :

```
namingContexts                o=INT, c=FR
                              dc=enic, dc=fr
supportedControl              2.16.840.1.113730.3.4.2
supportedExtension            1.3.6.1.4.1.4203.1.11.1
supportedSASLMechanisms       1.3.6.1.4.1.4203.1.5.1
supportedLDAPVersion          2
                              3
subschemaSubentry             cn=Subschema
```

Le premier attribut de cette entrée spéciale indique quels sont les suffixes des bases gérées par le serveur. Dans notre exemple, le serveur gère deux bases identifiées l'une par « o=INT, c=FR » et l'autre par « dc=enic, dc=fr ».
Les autres attributs indique quelles sont les fonctionnalités diverses que le serveur est capable de gérer. On indique ainsi la liste des contrôles, celle des extensions et celle des mécanismes SASL supportés par le serveur.
Le dernier attribut donne le nom d'une autre entrée opérationnelle située également au niveau de la racine du serveur. C'est le nom de l'entrée qui contient la description du schéma utilisé par le serveur pour l'ensemble des bases qu'il gère.

3.2.2.2 Entrées normales

Les entrées normales sont modélisées par au moins une classe d'objet.
Les classes d'objets décrivent des objets réels ou abstraits en les caractérisant par une liste d'attributs optionnels et obligatoires. Exemple :
La classe « person » implique que toute entrée de cette classe doit posséder les deux attributs suivants : cn et sn. L'attribut cn (common name) permet de spécifier le nom de l'entrée. Ici, cela peut correspondre au nom complet d'une personne. L'attribut sn (surname) permet de spécifier le nom de famille d'une personne. La classe « person » autorise l'usage d'autres paramètres. Il est ainsi possible d'ajouter les attributs suivants : userPassword, telephoneNumber, seeAlso et description.

Attributs

Un attribut est un type avec une ou plusieurs valeurs associées. Le type de l'attribut est défini par au moins un nom descriptif et un OID (Object Identifier).

Le type de l'attribut régit la possibilité pour une entrée de posséder plusieurs valeurs de ce type, la syntaxe à laquelle les valeurs doivent se conformer, les sortes de comparaison qui peuvent être effectuées sur les valeurs de cet attribut et d'autres fonctions.
Un exemple d'attribut est « mail ». Il peut y avoir une ou plusieurs valeurs pour cet attribut, celles-ci sont des chaînes de caractères ASCII, et la casse des caractères est non significative (e.g. michel@mesange est équivalent à MICHEL@MESANGE).

Classes d'objet

L'attribut objectClass est présent dans chaque entrée. Il spécifie les classes d'une entrée, lesquelles avec le schéma système et utilisateur déterminent les attributs permis pour une entrée.
Les valeurs de cet attribut peuvent être modifiées par les clients, mais il ne peut être supprimé. Les serveurs peuvent restreindre les modifications de cet attribut pour éviter les modifications de la classe structurelle de base d'une entrée (par exemple, on ne peut changer une personne en un pays).
Lors de la création ou de l'ajout d'une classe d'objet à une entrée, toutes les classes parents de cette classe sont implicitement ajoutées, et le client doit fournir des valeurs pour les attributs obligatoires des classes parents.

3.2.3 Modèle fonctionnel

Le modèle fonctionnel décrit le moyen d'accéder aux données et les opérations qu'on peut leur appliquer. Les opérations de base sont résumées dans le tableau suivant.

Opération LDAP	***Description***
Search	recherche dans l'annuaire d'objets à partir de critères
Compare	Comparaison du contenu de deux objets
Add	ajout d'une entrée dans l'arbre
Modify	Modification du contenu d'une entrée
Delete	Suppression d'une entrée
Rename (modifyDN)	Modification du DN d'une entrée
Bind	connexion au serveur (authentification)
Unbind	Déconnexion du serveur (fin de session)
Abandon	abandon d'une opération en cours
Extended	opérations étendues (version3)

Tableau 1 : Opérations de base

Les commandes *search* et *compare* se font sous la forme d'une requête composée de 8 paramètres comme indiqué par le tableau suivant.

Paramètre	***Description***
base object	l'endroit de l'arbre où doit commencer la recherche
Scope	la profondeur de la recherche
DerefAliases	indique si le serveur doit suivre ou non les liens
size limit	nombre de réponses limite
time limit	temps maximum alloué pour la recherche
AttrOnly	renvoie ou non la valeur des attributs en plus de leur type
Search filter	le filtre de recherche
list of attributes	la liste des attributs que l'on souhaite connaître

Tableau 2 : Paramètres d'une requête (searchRequest)

Le paramètre « base object » indique à partir de quel endroit dans l'arbre il faut commencer la recherche. On n'est pas obligé de commencer à partir de la racine, il est possible de commencer ailleurs ; ce qui suppose une connaissance préalable de la structure de l'arbre. Le paramètre « base object » se présente sous la forme d'un DN identifiant l'entrée à partir de laquelle commence la recherche.
Le *scope* définit la profondeur de la recherche dans l'arbre des données. La figure 3 montre la portée d'une recherche ou d'une comparaison en fonction du paramètre *scope*. Notons que dans le cas de la figure 3, « base object » vaut « ou=LOR, o=INT, c=FR ».

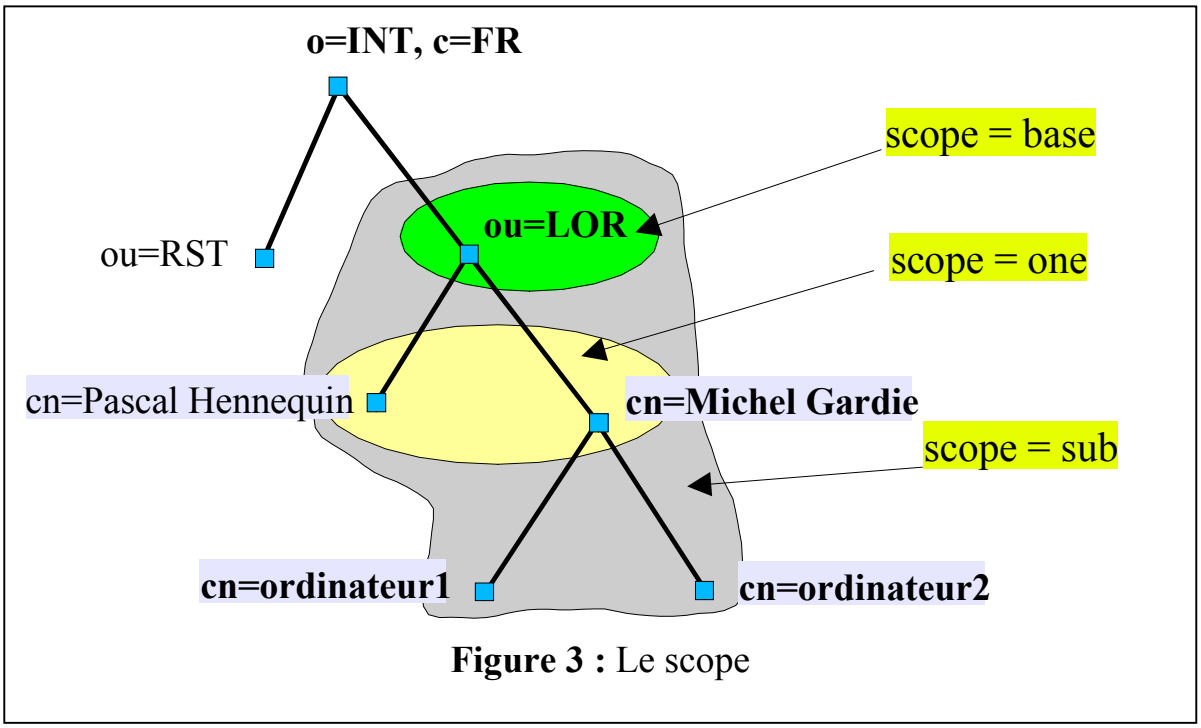

Figure 3 : Le scope

3.2.4 Protocole

Le protocole est normalisé par l'IETF (RFC 2252). La structure des PDU utilise le format BER (cf. ASN.1). Les valeurs des champs sont codées en BER pour les valeurs simples

(entiers, booléens, etc.). Les autres valeurs sont codées en ASCII, en UTF8, ou en base-64 selon la nature des données.
Par exemple, les OID *(Object Identifier)* identifiant les attributs ou classes d'objets ne sont pas transmis selon le codage BER classique des OID. Ils sont transmis sous la forme d'une chaîne ASCII qui est la représentation habituelle « humaine » des OID.

3.2.4.1 Communication client – serveur.

La figure 4 présente un exemple simplifié de communication entre un client et un serveur. La session commence par une ouverture de connexion TCP, suivie immédiatement d'une opération BIND permettant l'authentification du client auprès du serveur.
Une fois l'opération de BIND acceptée, le client transmet une opération de recherche (searchRequest). Le serveur exécute cette opération, et dans le cas où la recherche a abouti à des résultats positifs, transmet les résultats sous forme de retour d'entrées (searchResultEntry), suivi par le résultat global de la recherche (searchResultDone).
Le client émet ensuite une requête UNBIND pour terminer la session. La fermeture de la connexion TCP peut avoir lieu.

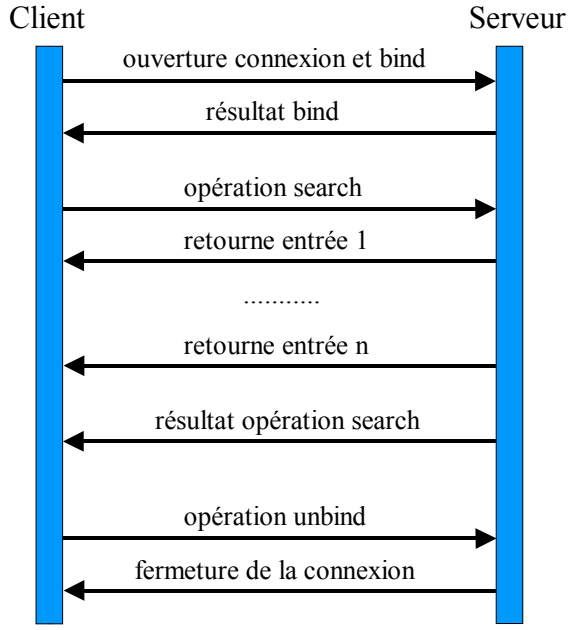

Figure 4 : Exemple de communication

Fonctionnalités offertes dans la communication client-serveur :
- recherche
- ajout
- modification
- suppression
- comparaison

3.2.4.2 Communication serveur – serveur

Il n'existe pas spécifiquement de communications serveur à serveur. Ce type de communication est traduit en réalité par une communication client-serveur dans laquelle un des deux serveurs joue le rôle d'un client.

service de duplication

Le service de duplication permet de disposer d'un serveur maître et d'un nombre indéterminé de serveurs esclaves. Les serveurs sont mis à jour automatiquement en cas de modification du serveur maître.

L'intérêt d'un tel service est d'offrir :

- une meilleure résistance aux pannes ; si le serveur maître est inactif ou en panne, les serveurs esclaves prennent le relais. Toutefois, il convient de configurer correctement les clients pour que ceux-ci puissent se rediriger vers les serveurs esclaves en cas d'indisponibilité du serveur maître.
- la possibilité de gérer facilement plusieurs sites miroir. Ceci est très intéressant dans le cas de sites miroir géographiquement très éloignés les uns des autres.

referral service

Ce service permet de lier plusieurs serveurs. Par exemple, le serveur A gère une base de données B_a ; dans cette base, existe une entrée pointant sur une base B_b gérée par un serveur B. Lorsqu'un utilisateur génère une requête vers le serveur A, celui-ci peut re-diriger automatiquement la requête vers le serveur B.

3.2.4.3 Format de transport des données.

La manière dont sont transportées les données dépend de la nature de celles-ci. Plusieurs cas sont envisageables. Il y a tout d'abord le cas de certaines données liées au protocole, comme par exemple le transfert d'OID ; il y a également le transfert de données de type texte, et enfin, se pose le problème des données binaires.

Dans les deux premiers cas, on utilise le format *BER-like* ou *light-weight BER*. Ce format, hérité du protocole DAP, utilise une variante de la syntaxe de transfert normalisée BER. Lorsque les données à transférer sont relativement simples, comme par exemple des entiers, des booléens, celles-ci sont transférées en utilisant le codage pur BER. Dans les autres cas, comme par exemple le transfert d'OID, on utilise un format mixte BER et ASCII. Le type de la donnée est au format BER, la valeur de la donnée est au format ASCII. Toutefois, le type BER utilisé est toujours le même: c'est le type OCTET STRING.

Dans le cas du transfert de données texte, deux cas sont également envisageables. Le texte transmis ne comporte aucun caractère diacritique ni caractère spécial. Dans ce cas, le transfert se fait en utilisant de l'ASCII pur. Sinon, on utilise le format UTF-8 pour le transfert de telles chaînes de caractères.

Enfin, lorsque l'on souhaite transférer des données binaires, comme par exemple une image JPEG, le format utilisé est le codage base-64.

4 Sécurité des échanges et authentification dans LDAPv3

Le protocole LDAPv3 intègre des mécanismes de sécurité qui permettent de :

- Authentifier un client LDAP et l'autoriser à réaliser des opérations sur le serveur LDAP. Les mécanismes d'authentification de LDAP reposent sur la méthode SASL *(Simple Authentication and Security Layer)* [rfc2222] (cf. chapitre 4.3).
- Protéger les informations échangées, aussi bien le résultat d'une requête LDAP, que le mot de passe fourni par un client LDAP lors de la phase d'authentification. La protection est offerte par le protocole TLS *(Transport Layer Security)* [rfc2446] (cf. chapitre 4.1).
- Protéger en intégrité/authentification les informations contenues dans la réponse LDAPResult. La protection est assurée par le protocole LDAP (cf. chapitre 4.2).

Les deux premiers aspects sont intimement liés dans LDAP. D'une part, la méthode SASL a parfois besoin de TLS par exemple pour protéger des mots de passe en confidentialité. D'autre part, SASL peut déléguer à TLS l'authentification du client, avec obligation pour le client de présenter un certificat et de s'authentifier à l'aide de son certificat (cf. chapitre 4.6).

A noter, le client LDAP considéré dans ce chapitre peut-être un utilisateur disposant d'un browser LDAP ou bien un serveur LDAP effectuant une opération LDAP sur un serveur LDAP distant.

4.1 Protection des échanges LDAPv3 par TLS [rfc2830]

Le protocole TLS 1.0 *(Transport Layer Security)* [rfc2246] est une couche de sécurité qui permet de sécuriser une session TCP. Il se déroule en deux phases :

- une négociation de services de sécurité (confidentialité, intégrité, authentification des données), et mécanismes de sécurité (algorithmes, fonctions) avec mise en place de clés partagées ;
- la protection des échanges par la mise en œuvre des services préalablement négociés.

Pour LDAPv3, TLS est vu comme un moyen de facilement assurer la protection des échanges (confidentialité et intégrité) et, parfois comme le moyen d'authentifier un client par le biais de son certificat TLS (cf. chapitre 4.6). Afin de permettre l'établissement d'une session TCP, une opération appelée *Start Transport Layer Security* (StartTLS) a été définie dans [rfc2830]. Il s'agit d'une requête étendue de LDAP (ExtendedRequest) qui se présente sous la forme :

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

Après émission de cette requête StartTLS, le client suspend toute émission jusqu'à réception de la réponse StartTLS du serveur :

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

L'OID considéré dans ces deux messages est celui de l'opération StartTLS, à savoir : 1.3.6.1.4.1.1466.20037
Ces requête/réponse sont encapsulées dans un PDU LDAP. La négociation TLS ne peut commencer que si la réponse contient le code success.

4.2 Authenticité des informations fournies par le serveur dans *LDAPResult [rfc2649]*

Un client LDAPv3 peut exiger lors d'une opération LDAPSearch que le serveur retourne une signature portant sur le résultat de la recherche. Pour cela, le client doit émettre dans le message LDAPSearch le contrôle DemandSignedResult (OID = 1.2.840.113549.6.0.1) qui correspond à un booléen :

```
DemandSignedResult ::= LDAPSigType
LDAPSigType ::= BOOLEAN
```

Le serveur devra alors retourner dans sa réponse LDAPResult le contrôle SignedResult (OID = 1.2.840.113549.6.0.2) qui contient une signature sur la réponse au format S/MIME pkcs-7/signature [rfc2311].

```
SignedResult ::= CHOICE {
        signature OCTET STRING }
```

Le serveur peut être configuré pour authentifier tout message LDAPResult émis. Le contrôle SignedResult ne doit pas être marqué CRITICAL ; c'est le client qui décidera de vérifier ou pas la signature [rfc2222].

4.3 Méthodes d'authentification : SASL (Simple Authentication and Security Layer) [rfc2222] et simple [rfc2829]

Deux catégories de méthodes d'authentification sont définies. La première type dite « simple » repose sur l'envoi d'un mot de passe. La seconde appelée « SASL » est définie comme une méthode permettant d'ajouter à des protocoles orientés connexion un mécanisme d'authentification. Plus exactement, SASL introduit un mécanisme d'identification/authentification des clients, et optionnellement la possibilité de négocier une couche de sécurité pour protéger les échanges ultérieurs.
Les protocoles pouvant faire appel à SASL sont nombreux et comptent IMAP, ACAP, POP3, LDAPv3... SASL est indépendant des méthodes d'authentification utilisées. En fait, chaque méthode exploitée par SASL est identifiée par un nom qui est composé d'une chaîne de caractères en majuscules approuvée par l'IANA. Elle doit faire l'objet d'un rfc, comme OTP (rfc 2444), SECURID (rfc 2808), DIGEST-MD5 [rfc2831].
Dans LDAPv3, les méthodes d'authentification préconisées [rfc2829] sont les suivantes :

- PLAIN (authentification simple) : le client LDAP s'authentifie en émettant un mot de passe en clair sur le réseau. Sachant que le mot de passe risque d'être espionné sur le réseau, il y a tout intérêt à le protéger par exemple par TLS.
- ANONYMOUS (authentification simple) : cette méthode n'est pas définie mais est présentée comme telle ici à seule fin de clarifier les explications ci-dessous. Une connexion anonyme à un serveur LDAP (cf. chapitre 4.4) n'est en fait rien d'autre que la méthode SASL PLAIN pour laquelle on ne précise pas de mot de passe (mot de passe de

longueur nulle). Une connexion anonyme s'avère surtout intéressante pour la consultation de bases LDAP en lecture seule. Elle n'est pas recommandée pour un accès en lecture/écriture au serveur du fait de la facilité qu'on aurait à réaliser des dénis de service (e.g. remplissage de tout l'espace disponible avec des informations inutiles).

- DIGEST-MD5 (authentification SASL) : le client LDAP s'authentifie grâce à un mot de passe, mais le mot de passe n'est pas envoyé tel quel sur le réseau (cf. chapitre 4.5). Il subit une transformation faisant intervenir la fonction MD5.
- EXTERNAL (authentification SASL) : le serveur LDAP délègue l'authentification auprès d'un protocole d'authentification extérieure comme TLS (cf. chapitre 4.6).

4.4 Connexion LDAP anonyme (ANONYMOUS)

LDAPv3 a obligation de supporter des connexions anonymes [rfc2245].
La méthode « ANONYMOUS + TLS » [rfc2829], i.e. s'appuyant sur TLS peut optionnellement être supportée. Cette dernière est décrite à la figure 5. Elle consiste à établir une session TLS par le mécanisme StartTLS (cf. chapitre 4.1). Tous les échanges sont alors protégés par TLS. Reste alors à établir une session LDAP anonyme.

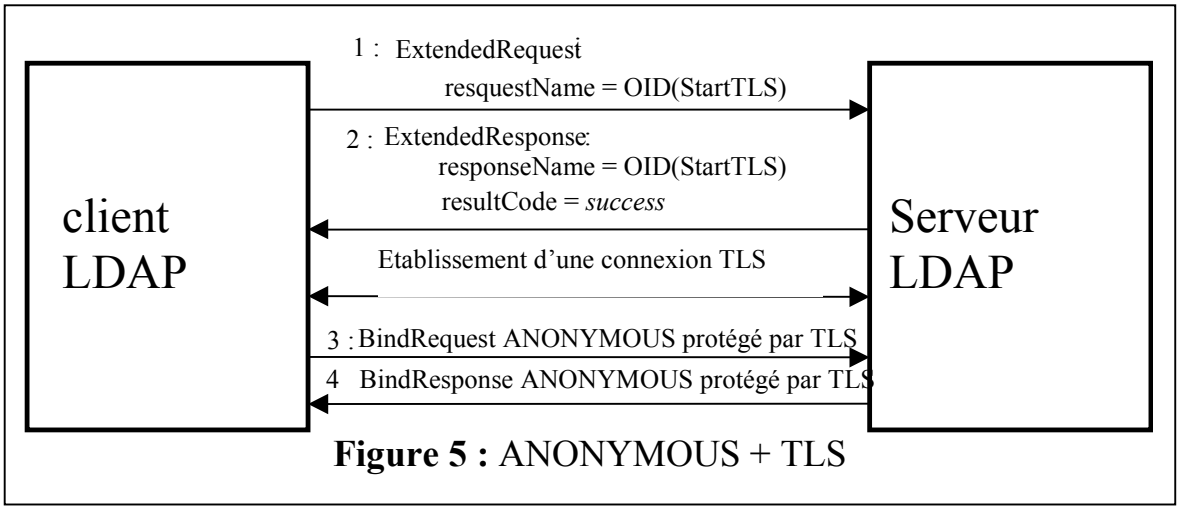

Figure 5 : ANONYMOUS + TLS

4.5 Authentification par mot de passe

Les implémentations de LDAP ont obligation de supporter l'authentification par mot de passe basée sur le mécanisme SASL DIGEST-MD5 [rfc2831]. Il est recommandé d'implémenter la méthode simple PLAIN accompagnée de TLS pour assurer la confidentialité du mot de passe durant le transfert.

4.5.1 SASL DIGEST-MD5 [rfc2829]

Le client LDAP doit d'abord vérifier que le serveur supporte la méthode SASL DIGEST-MD5. Pour cela, comme le présente la figure 6, il fait une requête de type LDAPSearch au serveur sur l'attribut opérationnel supportedSASLMechanisms, présent uniquement sur le rootDSE et doit vérifier que la valeur 'DIGEST-MD5' est bien présente dans la réponse.

Le client envoie alors une requête Bind précisant l'utilisation de SASL DIGEST-MD5. Le serveur retourne un challenge dans le champ serverSaslCreds. Ce challenge correspond au champ digest-challenge dans le document [rfc2831] décrivant la méthode DIGEST-MD5.
Le client émet un deuxième Bind contenant une réponse au challenge dans le champ Credentials (qui correspond au champ digest-response dans [rfc2831]). Si l'authentification est réussie, le serveur renvoie une réponse Bind avec une indication success.

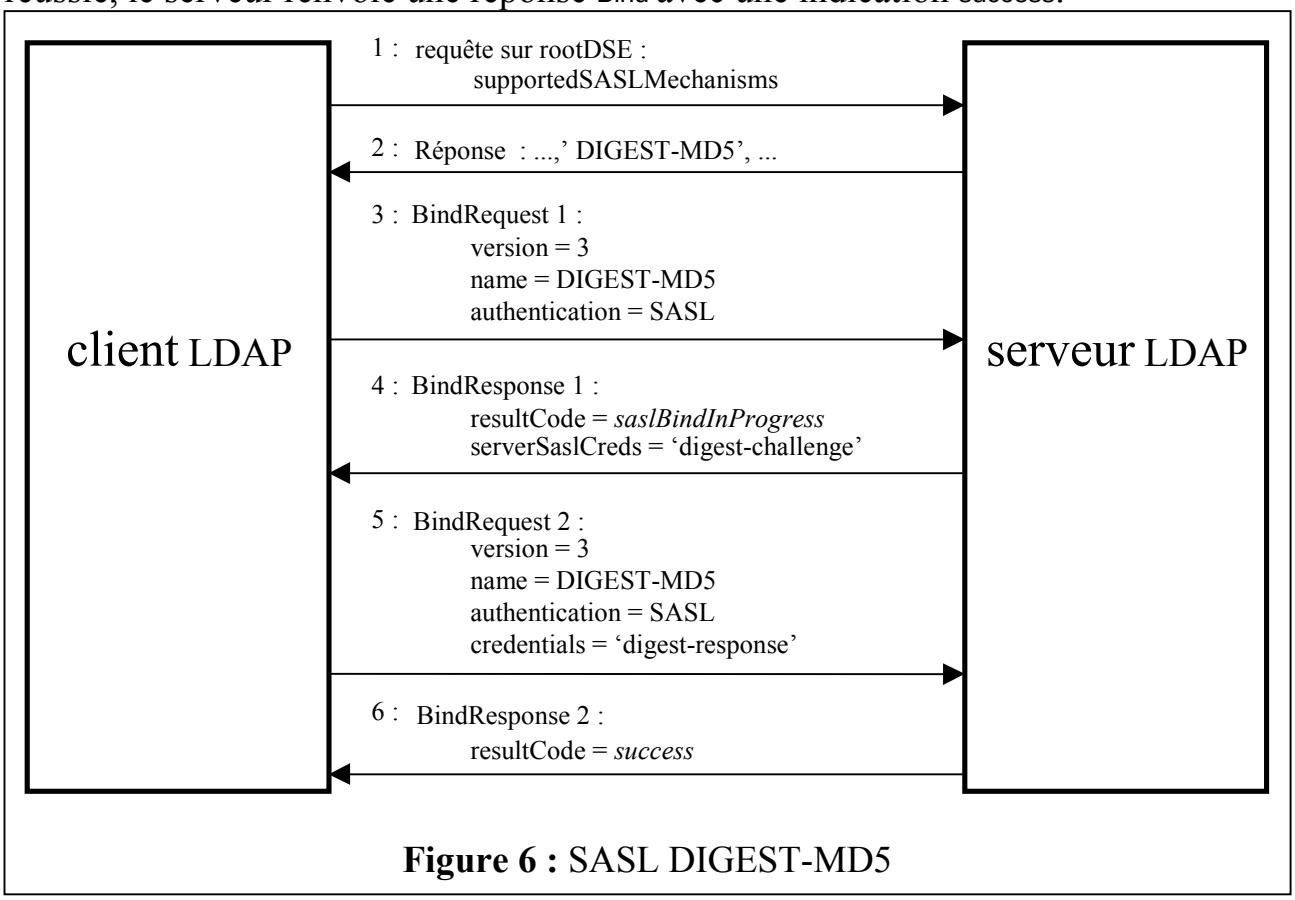

Figure 6 : SASL DIGEST-MD5

4.5.2 PLAIN + TLS

Avec la méthode PLAIN, chaque utilisateur susceptible de s'authentifier auprès du serveur doit disposer d'une entrée associée dans la base LDAP. Cette entrée doit contenir un attribut password (cf. chapitre 5.1), ce qui lui permet de s'authentifier auprès du serveur lors d'une séquence d'opérations Bind. Pour protéger le transfert du mot de passe sur le réseau, une négociation TLS a lieu au préalable grâce à l'opération StartTLS. Les échanges sont décrits à la figure 7.

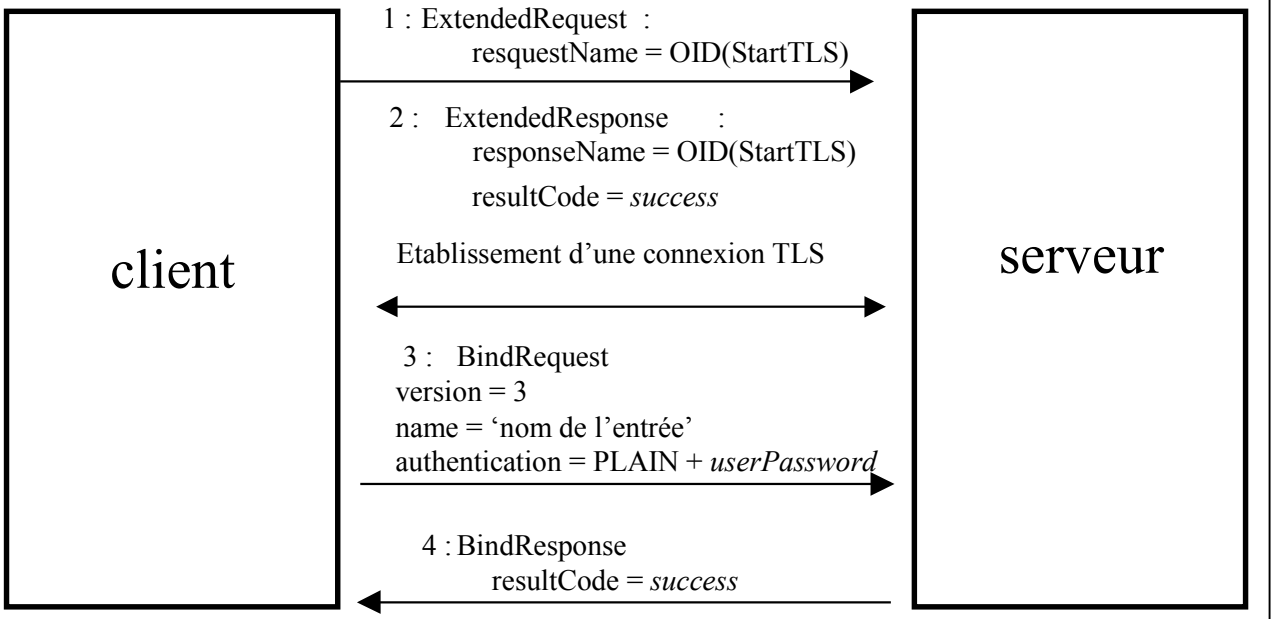

Figure 7 : SASL PLAIN + TLS

4.6 Authentification par certificat TLS [rfc2830]

Il est recommandé que LDAP supporte l'authentification d'un client par TLS [rfc2830]. L'activation de cette méthode nécessite de choisir la méthode SASL EXTERNAL. En effet, cette dernière oblige le serveur à authentifier via le protocole TLS le client en fonction du certificat et de la signature présentés. Bien entendu, il faut que l'autorité de certification émettrice du certificat du client soit reconnue du serveur et, théoriquement, il faudrait que ce certificat n'ait pas été révoqué. Si la méthode SASL EXTERNAL n'est pas activée, alors c'est la méthode ANONYMOUS qui prévaut.
Lorsque la connexion TLS est établie et le client authentifié par le serveur (cf. figure 8), le client émet une requête BindRequest au serveur en précisant le mécanisme SASL EXTERNAL. Le serveur peut vérifier la méthode et accepter ou pas la connexion. Tous les échanges LDAP sont protégés ici par TLS. La protection TLS offerte dépend des services et mécanismes négociés lors de l'établissement de la connexion TLS. La suite cryptographique suivante doit obligatoirement être supportée : TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA [rfc2829].

LDAP et la certification

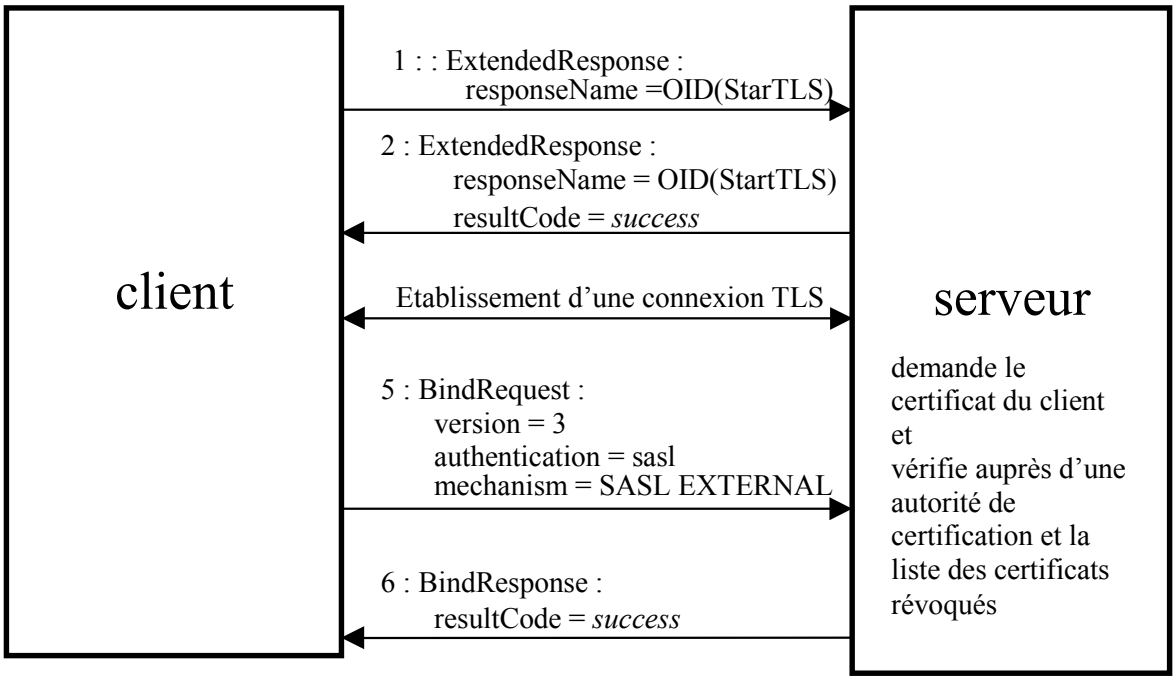

Figure 8 : SASL EXTERNAL + TLS

5 Contrôle d'accès à la base LDAPv3 et sécurité mise en œuvre pour protéger le contenu de la base

Ce chapitre traite des aspects liés à la protection de la base LDAP. Elle concerne deux volets :

- le contrôle de l'accès à la base. En fonction du client LDAP identifié et de l'éventuelle authentification réalisée (cf. chapitre 4), un client est autorisé ou pas à lire, écrire ou modifier des entrées/attributs dans la base. Les règles d'autorisation se présentent sous la forme de listes de contrôle d'accès (acl) décrites au chapitre 5.2. Si un mot de passe est utilisé au cours de la phase d'authentification, le mot de passe fourni par le client doit être confronté à un mot de passe en local qui est protégé. Les techniques de protection sont décrites dans le chapitre 5.1.
- l'intégrité de la base LDAP (cf. chapitre 5.3). Le but est de permettre à un client LDAP de vérifier la validité des informations de la base LDAP et de contrôler l'auteur de la modification de l'entrée dans la base. Ce dernier contrôle est rendu possible grâce à des mécanismes de signature électroniques.

Notez qu'un client LDAP peut prendre la forme d'un utilisateur disposant d'un browser LDAP ou bien d'un serveur LDAP effectuant une opération LDAP sur un autre serveur.

5.1 Stockage des mots de passe

Les mots de passe associés à des clients LDAP (traditionnellement des DN [rfc2253]) peuvent être stockés dans la base dans deux attributs possibles : userPassword [rfc2256] ou authPassword [rfc3112].

userPassword se présente comme suit :

userPassword (2.5.4.35

NAME 'userPassword'
EQUALITY octetStringMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.40{128})

userPassword est stocké sous la forme d'un Octet String et n'est pas chiffré, ce qui facilite le vol de mots de passe. Il est donc préférable de stocker des valeurs dérivées des mots de passe plutôt que les mots de passe eux-même. C'est pourquoi, l'attribut authPassword a été défini avec les attributs authPasswordSyntax (OID 1.3.6.1.4.1.4203.1.1.2 – valeurs associées au mot de passe), authPasswordExactMatch (OID 1.3.6.1.4.1.4203.1.2.2 – conditions devant être strictement remplies pour valider un mot de passe présenté), authPasswordMatch (OID 1.3.6.1.4.1.4203.1.2.3 – conditions plus souples devant être remplies pour valider un mot de passe présenté), supportedAuthPasswordSchemes (OID 1.3.6.1.4.1.4203.1.3.3 – ensemble des techniques supportées), authPasswordObject (OID 1.3.6.1.4.1.4203.1.4.7 – entrée principale pouvant inclure un ou plusieurs mots de passe).

authPassword (OID 1.3.6.1.4.1.4203.1.3.4) :

(1.3.6.1.4.1.4203.1.3.4

NAME 'authPassword'
DESC 'password authentication information'
EQUALITY 1.3.6.1.4.1.4203.1.2.2
SYNTAX 1.3.6.1.4.1.4203.1.1.2)

Deux techniques de stockage de mot de passe à base de fonctions de hachage [rfc3112] sont disponibles : MD5 [rfc1321] et SHA1 [SHA1]. Elles consistent à concaténer le mot de passe avec une valeur *salt* (contenue dans authPasswordSyntax) de longueur comprise entre 64 et 128 bits et à appliquer une fonction de hachage à ce résultat. La valeur obtenue est codée en base64 et enregistrée dans authPasswordSyntax.

5.2 Listes d'accès (acl)

Une liste de contrôle d'accès va permettre de spécifier quelles entrées et/ou quels attributs peuvent être lus, modifiés, non vus, et par qui (utilisateurs anonymes ou authentifiés). La manière dont sont spécifiées les listes est dépendant de l'implémentation. Mais les listes doivent normalement respecter les principes définis dans [rfc2820].
La présentation des listes d'accès sera faite ici dans le cas de l'outil OpenLDAP. Dans cet outil, une liste d'accès est constituée de règles (directives) dont la syntaxe générale est la suivante :

```
access to <what>
     [ by <who> <access> [ <control> ] ]+
```

La clause *what* représente l'objet qui est soumis au contrôle. Cela peut être une entrée, un attribut, etc.
La clause *who* précise quel type d'utilisateur est concerné par le contrôle.
Et enfin, la clause *access* précise le type d'accès qui est accordé à *who* pour l'objet *what*.
La clause *control* est optionnel. Il permet de régir la manière dont sont lues les règles d'accès.

5.2.1 Clause <what>

Il peut prendre les formes suivantes :

```
*
[dn[.<dnstyle>]=<pattern>]
[filter=<ldapfilter>]
[attrs=<attrlist>]
```

Le caractère * spécifie toutes les entrées de l'annuaire.
La forme *dn=pattern* permet de spécifier une entrée particulière de la base. Par exemple : `dn="cn=liste machines,ou=Centre de Calcul,o=Banque XYZ,c=FR"` permet de contrôler l'accès à cette entrée.
La forme *filter=pattern* permet de sélectionner des entrées répondant à un filtre de recherche. Exemple : `filter="(&(objectClass=inetorgperson)(ou=LOR))"` sélectionne toutes les entrées dont la classe est *inetorgperson* et l'attribut *ou* vaut LOR.
La forme *attrs=<attrlist>* permet de spécifier une liste d'attributs dont on veut contrôler l'accès. Exemple : `attrs=cn,mail,telephoneNumber` permet de spécifier que les attributs *cn*, *mail* et *telephoneNumber* vont être soumis aux mêmes règles de contrôle.

5.2.2 Clause <who>

Cette clause précise qui peut (ou non) accéder aux entités spécifiées dans la clause *what* précédente. Il peut prendre une des nombreuses formes suivantes :

```
*
anonymous
users
self

dn[.<dnstyle>[,<modifier>]]=<pattern>
dnattr=<attrname>
group[/<objectclass>[/<attrname>]]
     [.<style>]=<pattern>
peername[.<style>]=<pattern>
sockname[.<style>]=<pattern>
domain[.<domainstyle>[,<modifier>]]=<pattern>
sockurl[.<style>]=<pattern>
set[.<style>]=<pattern>

ssf=<n>
transport_ssf=<n>
tls_ssf=<n>
sasl_ssf=<n>

aci=<attrname>
```

Pour simplifier, nous nous intéresserons ici aux cinq premières formes.
Le caractère * représente n'importe qui. Autrement dit, tout le monde.
Le mot-clé **anonymous** signifie que l'accès est accordé aux utilisateurs non authentifiés. Ce mot-clé est souvent utilisé pour limiter l'accès aux ressources d'authentification (comme par exemple, les mots de passe) aux utilisateurs non authentifés pour des besoins justement d'authentification.

Le mot-clé **users** signifie que l'accès est accordé aux utilisateurs authentifiés, c'est-à-dire qui se sont connectés par exemple à l'aide d'un mot de passe.
Le mot-clé **self** signifie que l'accès est accordé au propriétaire de l'entrée, c'est-à-dire que le nom distingué de l'utilisateur et celui de l'entrée sont les mêmes. De plus, l'utilisateur se sera préalablement authentifié.
Enfin, la forme **dn=pattern** permet de spécifier un utilisateur particulier qui ne rentre dans aucune des catégories précédentes.

5.2.3 Clause <access>

Cette clause spécifie le niveau d'accès auquel peut prétendre l'utilisateur spécifié dans la clause *who* précédente. Sa structure est la suivante :

```
<access> ::= [self]{<level>|<priv>}
<level> ::= none|auth|compare|search|read|write
<priv> ::= {=|+|-}{w|r|s|c|x}+
```

Le modificateur **self** permet des opérations spéciales comme d'avoir un certain niveau d'accès ou de privilège uniquement dans le cas où l'opération implique le nom de l'utilisateur qui effectue la requête. Cela implique que l'utilisateur qui effectue la requête soit authentifié (*bound*). Un exemple est l'accès **selfwrite** à l'attribut *member* d'un groupe, ce qui autorise les utilisateurs à ajouter ou retirer leur propre DN d'une liste de membres d'un groupe, sans affecter les autres membres.

Le modèle d'accès <**level**> est basé sur une interprétation incrémentale des privilèges d'accès. Chaque niveau implique les niveaux précédents. Ainsi par exemple, le niveau **write** implique tous les autres niveaux.
Alors que **none** est trivial (aucun accès accordé), **auth** signifie que l'on est autorisé à accéder à un attribut uniquement pour effectuer des opérations d'authentification (c'est-à-dire un *bind*). Ceci est utile pour accorder aux utilisateurs non autorisés le moins possible d'accès aux ressources critiques comme par exemple les mots de passe.
Le niveau **compare** permet de comparer des attributs, sans que les valeurs soient retournées au client.
Le niveau **search** permet d'appliquer des filtres de recherche sur les entrées et/ou les attributs sans que les valeurs des attributs soient retournées au client.
Le niveau **read** permet de lire les attributs d'une entrées, et enfin, le niveau **write** permet de les modifier.

Le modèle d'accès <**priv**> repose sur le positionnement explicite de privilèges d'accès pour chaque clause. Le symbole = remet à zéro les accès préalablement définis ; en conséquence, les privilèges finaux seront seulement ceux définis par la clause. Les signes + et – ajoutent et retirent respectivement les privilèges d'accès à ceux existants. Les privilèges sont **w** pour l'écriture, **r** pour la lecture, **s** pour la recherche, **c** pour la comparaison et **x** pour l'authentification. Plus d'un privilège peuvent être ajoutés en une seule définition.

5.2.4 Clause <control>

La clause control permet de régir la manière dont sont lues les règles d'accès. Il peut prendre une des formes suivantes :

```
stop
continue
break
```

dans laquelle **stop**, la valeur par défaut, signifie que l'analyse des règles d'accès s'arrête en cas d'application d'une règle.
Les deux autres formes sont utilisées pour continuer le traitement des autres clauses. De manière plus détaillée, **continue** permet aux autres clauses <**who**> dans la même clause <**access**> d'être considérées. Par contre, **break** autorise les autres clauses <**access**> (qui correspondent à la même cible) d'être traitées.
Considérons l'exemple (stupide) suivant :

```
access to dn.subtree="dc=example,dc=com"
          attrs=cn
         by * =cs break

access to
          dn.subtree="ou=People,dc=example,dc=com"
        by * +r
```

qui attribue des privilèges de recherche et de comparaison à tout le monde sous l'arbre « dc=example, dc=com », et grâce à la second règle, permet également la lecture dans le sous-arbre « ou=people ».
L'exemple ci-dessous (encore plus stupide) accorde à tout le monde des droits de recherche et de comparaison d'une part, et des droits de lecture aux utilisateurs privilégiés d'autre part :

```
access to dn.subtree="dc=example,dc=com"
          attrs=cn
        by * =cs continue
        by users +r
```

5.2.5 Exemple de liste d'accès

Voici un exemple succinct de liste :

```
access to attrs=employeeType
      by dn="cn=DRH,o=INT,c=FR" write
      by dn="cn=Chef,ou=LOR,o=INT,c=FR" read
      by self read
      by users none
      by anonymous none

access to attrs=seeAlso,businessCategory,homePostalAddress,homePhone
      by users read
      by anonymous none
```

Cette liste contient deux règles de contrôle. La première contrôle l'accès à l'attribut « employeeType ». Un seul utilisateur authentifié ("cn=DRH,o=INT,c=FR") possède les droits d'écriture. Le responsable hiérarchique et le propriétaire sont limités à un droit de lecture ; tous les autres utilisateurs ne peuvent y accéder.

La deuxième règle régit les accès aux attributs « seeAlso », « businessCategory », etc. de la façon suivante : les utilisateurs authentifiés ont le droit de lire, les autres ignorent l'existence de tels attributs.

5.3 Intégrité et authenticité des informations contenues dans la base LDAP [rfc2649]

Lors d'une modification d'entrées dans la base LDAP, il est important de garder la trace du client LDAP responsable de cette modification. A cette fin, différents aspects doivent être considérés :

- L'ajout éventuel d'une signature à la demande de modification/création/destruction (opération de modification) émanant du client LDAP.
- L'enregistrement de ces opérations signées dans un journal des opérations.
- Des vérifications de signature sont recommandées soit pour valider l'opération (au niveau du serveur LDAP) soit pour valider le contenu d'une entrée (au niveau d'un client ou serveur LDAP).

5.3.1 Authentification des demandes de modification/création/destruction d'entrées de la base à des fins de journalisation [rfc2649]

Le chapitre 4 décrit les méthodes SASL disponibles permettant d'authentifier les clients LDAP. Cette authentification couplée au acl du chapitre 5.2 permettent d'autoriser certains clients à accéder en écriture à la base LDAP. Ici, sont présentées des techniques propres à LDAP permettant de garantir l'intégrité et l'authenticité des opérations LDAP. Elles ont pour intérêt de permettre une journalisation aisée des opérations de création/modification/destruction.
Pour rendre ces services de sécurité dans LDAP, le client LDAP doit en général signer les opérations aboutissant à la modification de la base LDAP. Plus exactement, le client construit tout d'abord l'opération LDAPMessage ; la signature est ensuite calculée sur l'opération LDAPMessage ; elle respecte le format Multipart/Signed MIME [rfc1847], c'est-à-dire, elle comprend deux parties, une première contenant les informations à signer, et une seconde contenant tous les éléments permettant de vérifier la signature ; la signature est ensuite ajoutée dans le contrôle LDAP SignedOperation (OID = 1.2.840.113549.6.0.0) de l'attribut signatureIncluded.
Le contrôle SignedOperation permettant d'adjoindre une signature à une opération LDAP se présente sous la forme suivante :

```
SignedOperation ::= CHOICE {
        signbyServer NULL,
        signatureIncluded OCTET STRING
        }
```

Si le client n'est pas à même de générer une signature, il peut déléguer cette tâche au serveur lui-même. Dans ce cas là, le client doit choisir l'attribut signbyServer dans le contrôle SignedOperation. Généralement, après authentification du client, le serveur signera l'opération. Peu importe le signataire, une fois l'entrée de la base modifiée conformément à l'opération reçue, le contrôle SignedOperation (signé soit par le client, soit par le serveur) est ensuite enregistré dans le journal des opérations (cf. chapitre 5.3.3).

Certains serveurs LDAP obligent que toute opération de modification/création/destruction d'entrée LDAP soit signée. Dans ce cas, il doit le préciser dans son attribut signedDirectoryOperationSupport de l'entrée rootDSE.

```
attributetype (1.2.840.113549.6.2.2
NAME 'signedDirectoryOperationSupport'
DESC 'how many of the LDAP operations must be signed'
SYNTAX 'Integer' SINGLE-VALUE )
```

avec l'attribut signedDirectoryOperationSupport qui précise la politique appliquée par le serveur :

- '0' : les opération peuvent être signées
- '1' : les opérations doivent toujours être signées
- '2' : les opérations ne doivent jamais être signées

Si un serveur journalise l'ensemble des opérations de modification, il enregistrera toutes les opérations signées par les clients ; il se chargera de signer toutes les autres opérations (non signées par le client) avant de les enregistrer dans son journal.

5.3.2 Vérification

Lorsque le serveur reçoit une opération de modification/création/destruction signée par le client, le serveur peut sans obligation vérifier la signature incluse dans le contrôle SignedOperation. Cette vérification peut s'avérer gourmande en temps de calcul en particulier avec la vérification de la chaîne de certificats du client. Si le serveur est incapable de vérifier la signature, alors il retourne un code d'erreur unwillingToPerform dans l'opération LDAPResult. Si l'opération SignedOperation est marquée 'CRITICAL', alors le serveur doit journaliser l'opération signée.

Lorsqu'un client LDAP récupère les attributs Changes (cf. chapitre 5.3.3) contenant des opérations signées, il est recommandé qu'il vérifie leur authenticité par le biais de la signature.

5.3.3 Journal des opérations (enregistrement de l'opération dans la base LDAP)

Le journal des opérations maintient l'historique des opérations de modifications opérées sur chaque entrée de la base LDAP. Cela permet après modification de l'entrée de savoir qui est l'auteur de la modification et de vérifier l'intégrité de l'entrée, i.e. sa conformité vis-à-vis de la dernière opération de modification réalisée. Ce journal permet en particulier de se prémunir des attaques visant l'intégrité de la base, et ce malgré les contrôles d'authentification/autorisation réalisés.

Une information appelée signedAuditTrail (OID 1.2.840.113549.6.1.0) se présente de la façon suivante :

```
( 1.2.840.113549.6.1.0
NAME 'signedAuditTrail'
SUP top
AUXILIARY
MUST (
Changes ) )
```

avec dans l'attribut Changes (OID 1.2.840.113549.6.2.0) les informations suivantes :

```
Changes ::= SEQUENCE {
        sequenceNumber [0] INTEGER (0 .. maxInt),
        signedOperation [1] OCTET STRING }
```

Pour chaque entrée créée, un attribut Changes pour cette entrée est défini et correspond au numéro de séquence 0. Chaque modification de cette entrée est enregistrée dans un numéro de séquence qui est incrémenté à chaque fois. Notez que l'opération provoquant la création ou modification de l'entrée est enregistrée dans l'attribut signedOperation qui contient une signature portant sur l'opération elle-même (cf. chapitre 5.3.1).

5.3.4 Enregistrement de l'opération de destruction dans la base LDAP

Cette opération est spéciale car elle aboutit à la destruction d'une entrée. Il faut donc conserver en plus de la signature, l'entrée détruite. Ces informations sont conservées dans l'objet zombieObject (OID 1.2.840.113549.6.1.2) :

```
( 1.2.840.113549.6.1.2
NAME 'zombieObject'
SUP top
STRUCTURAL MUST (
Cn $ Changes $ OriginalObject ) )
```

avec dans l'attribut OriginalObject (OID 1.2.840.113549.6.2.1) des informations relatives à l'entrée détruite :

```
( 1.2.840.113549.6.2.1
NAME OriginalObject
DESC 'The LDAP URL of an object that has been deleted from the directory'
SYNTAX 'Binary' )
```

Quand une entrée de la base est détruite, cette entrée disparaît de l'objet signedAuditTrail. Par contre, toutes les modifications précédemment enregistrées dans signedAuditTrail sont recopiées dans zombieObject avec la toute dernière opération de destruction.

6 LDAP et la publication de certificats

En plus du [rfc2459] qui présente les certificats X.509 et CRL d'une façon générale, deux rfc s'intéressent à l'utilisation de LDAP pour la publication des certificats. [rfc2559] décrit les opérations pour rechercher un certificat dans une base ainsi que les opérations de modification de la base de certificats. [rfc2587] décrit le contenu de la base LDAP.

6.1 Au niveau de la base [rfc2587] [rfc2459]

[rfc2587] décrit les entrées pkiUser et pkiCA à considérer dans la base LDAP pour inclure des certificats d'utilisateur et du CA.

```
pkiUser OBJECT-CLASS ::= {
SUBCLASS OF { top}
KIND auxiliary
MAY CONTAIN {userCertificate}
ID joint-iso-ccitt(2) ds(5) objectClass(6) pkiUser(21)}


userCertificate ATTRIBUTE ::= {
WITH SYNTAX Certificate
EQUALITY MATCHING RULE certificateExactMatch
ID joint-iso-ccitt(2) ds(5) attributeType(4) userCertificate(36) }
```

```
pkiCA OBJECT-CLASS ::= {
SUBCLASS OF { top}
KIND auxiliary
MAY CONTAIN {cACertificate | certificateRevocationList | authorityRevocationList | crossCertificatePair }
ID joint-iso-ccitt(2) ds(5) objectClass(6) pkiCA(22)}


cACertificate ATTRIBUTE ::= {
WITH SYNTAX Certificate
EQUALITY MATCHING RULE certificateExactMatch
ID joint-iso-ccitt(2) ds(5) attributeType(4) cACertificate(37) }


crossCertificatePair ATTRIBUTE::={
WITH SYNTAX CertificatePair
EQUALITY MATCHING RULE certificatePairExactMatch
ID joint-iso-ccitt(2) ds(5) attributeType(4) crossCertificatePair(40)}
```

Le format de l'attribut Certificate est donnée en ASN.1 dans [2459] :

```
Certificate ::= SEQUENCE {
        tbsCertificate TBSCertificate,
        signatureAlgorithm AlgorithmIdentifier,
        signatureValue BIT STRING }
TBSCertificate ::= SEQUENCE {
        version [0] EXPLICIT
        Version DEFAULT v1,
        serialNumber CertificateSerialNumber,
        signature AlgorithmIdentifier,
        issuer Name,
        validity Validity,
        subject Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL, -- If present, version shall be v2 or v3
        subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL, -- If present, version shall be v2 or v3
        extensions [3] EXPLICIT Extensions OPTIONAL -- If present, version shall be v3 }
Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
Validity ::= SEQUENCE {
        notBefore Time,
        notAfter Time }
Time ::= CHOICE {
        utcTime UTCTime,
        generalTime GeneralizedTime }
UniqueIdentifier ::= BIT STRING
SubjectPublicKeyInfo ::= SEQUENCE {
        algorithm AlgorithmIdentifier,
        subjectPublicKey BIT STRING }
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
        extnID OBJECT IDENTIFIER,
        critical BOOLEAN DEFAULT FALSE,
        extnValue OCTET STRING }
AlgorithmIdentifier ::= SEQUENCE {
        algorithm OBJECT IDENTIFIER,
        parameters ANY DEFINED BY algorithm OPTIONAL }
```

6.2 Filtrage – le contrôle de l'accès aux certificats

Dans le projet CADDISC, nous nous intéressons à la publication de certificats sans restriction quant à leur diffusion. Le contrôle d'accès (cf. chapitre 4) et les méthodes d'authentification (cf. chapitre 5.2) à mettre en place sont présentés sur le tableau 3 :

	Consultation de certificats	Mise à jour de certificats
Restriction de l'accès (acl)	Accès en lecture pour tous	Accès en écriture pour le CA limité aux classes d'objets pkiUser et pkiCA Accès en écriture pour l'administrateur sans restriction
Méthodes d'authentification standard acceptées	Connexion anonyme acceptée	SASL DIGEST-MD5 ou authentification par certificat TLS

Tableau 3 : Restriction de l'accès à la base et méthodes d'authentification

Si le but du serveur LDAP était de publier des certificats dans un cadre privé (e.g. en interne au GET), alors les accès en lecture seraient restreints aux entités du GET.

6.3 Mise à jour de la base LDAP

Seul le CA est autorisé à mettre à jour les certificats de la base LDAP (cf. tableau 3). Cette opération de modification doit être signée et enregistrée dans le journal des opérations (cf. chapitre 5.3.3).

7 LDAP et la révocation de certificats

[rfc2587] décrit la liste de révocation de certificats (CRL) classique (tel que présenté dans [rfc2459]) sous la forme :

```
certificateRevocationListATTRIBUTE::={
        WITH SYNTAX CertificateList
        EQUALITY MATCHING RULE certificateListExactMatch
        ID joint-iso-ccitt(2) ds(5) attributeType(4) certificateRevocationList(39)}
```

Une autre liste CRL est définie pour révoquer les certificats d'autres CAs générés par le CA local.

```
authorityRevocationListATTRIBUTE::={
        WITH SYNTAX CertificateList
        EQUALITY MATCHING RULE certificateListExactMatch
```

```
        ID joint-iso-ccitt(2) ds(5) attributeType(4) authorityRevocationList(38)}


deltaRevocationList ATTRIBUTE ::= {
      WITH SYNTAX CertificateList
      EQUALITY MATCHING RULE certificateListExactMatch
      ID joint-iso-ccitt(2) ds(5) attributeType(4) deltaRevocationList(53) }


deltaCRL OBJECT-CLASS::= {
      SUBCLASS OF { top }
      KIND auxiliary
      MAY CONTAIN { deltaRevocationList }
      ID joint-iso-ccitt(2) ds(5) objectClass(6) deltaCRL(23) }


commonName ATTRIBUTE::={
      SUBTYPE OF name
      WITH SYNTAX DirectoryString
      ID joint-iso-ccitt(2) ds(5) attributeType(4) commonName(3) }


cRLDistributionPoint OBJECT-CLASS::= {
      SUBCLASS OF { top }
      KIND structural
      MUST CONTAIN { commonName }
      MAY CONTAIN { certificateRevocationList |
                        authorityRevocationList |
                        deltaRevocationList }
      ID joint-iso-ccitt(2) ds(5) objectClass(6) cRLDistributionPoint(19) }
```

Dans [rfc2459], les définitions complémentaires sont données :

```
CertificateList ::= SEQUENCE {
        tbsCertList TBSCertList,
        signatureAlgorithm AlgorithmIdentifier,
        signatureValue BIT STRING }


TBSCertList ::= SEQUENCE {
        version Version OPTIONAL, -- if present, shall be v2
        signature AlgorithmIdentifier,
        issuer Name,
        thisUpdate Time,
        nextUpdate Time OPTIONAL,
        revokedCertificates SEQUENCE OF SEQUENCE {
                userCertificate CertificateSerialNumber,
                revocationDate Time,
                crlEntryExtensions Extensions OPTIONAL -- if present, shall be v2 } OPTIONAL,
        crlExtensions [0] EXPLICIT Extensions OPTIONAL -- if present, shall be v2 }


Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
Time ::= CHOICE { utcTime UTCTime, generalTime GeneralizedTime }
Extension ::= SEQUENCE {
    extnID OBJECT IDENTIFIER,
    critical BOOLEAN DEFAULT FALSE,
    extnValue OCTET STRING }
AlgorithmIdentifier ::= SEQUENCE {
    algorithm OBJECT IDENTIFIER,
    parameters ANY DEFINED BY algorithm OPTIONAL }
id-ce-cRLDistributionPoints OBJECT IDENTIFIER ::= { id-ce 31 }
cRLDistributionPoints ::= { CRLDistPointsSyntax }
CRLDistPointsSyntax ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint
```

```
DistributionPoint ::= SEQUENCE {
      distributionPoint [0] DistributionPointName OPTIONAL,
      reasons [1] ReasonFlags OPTIONAL,
      cRLIssuer [2] GeneralNames OPTIONAL }
DistributionPointName ::= CHOICE {
      fullName [0] GeneralNames,
      nameRelativeToCRLIssuer [1] RelativeDistinguishedName }
ReasonFlags ::= BIT STRING {
      unused (0),
      keyCompromise (1),
      cACompromise (2),
      affiliationChanged (3),
      superseded (4),
      cessationOfOperation (5),
      certificateHold (6) }
```

8 Conclusion sur la place de LDAP dans une PKI

Le serveur LDAP est parfaitement adapté à la publication de certificats électroniques. C'est-à-dire il remplit parfaitement le rôle de l'autorité d'enregistrement qui renseigne les clients sur les certificats associés à des utilisateurs, serveurs web, passerelle VPN…

Etant donnée le rôle critique joué par l'autorité de certification, il vaut mieux que l'autorité de certification soit gérée indépendamment du serveur LDAP. Il est même conseillé que le logiciel de génération de certificats/CRL soit hébergé sur une machine déconnectée du réseau.

En pratique…

Dans le projet CADDISC, notre maquette repose sur les logiciels libres OpenCA_0.9.1-1, et OpenLDAP-2.1.22. Le logiciel OpenCA tient le rôle de l'autorité de certification en permettant la génération, la distribution et la révocation de certificats et également le rôle de l'autorité d'enregistrement en approuvant les demandes de génération et de révocation issues des utilisateurs. En pratique, OpenCA_0.9.1-1 est administré via Apache (Apache_1.3.27) et donc chacun des rôles correspond à un répertoire différent sur le serveur OpenCA : /ca/ et /ra/. Ainsi suivant le rôle joué, l'administrateur se connectera sur l'une ou l'autre des pages web, par exemple http://pivert/ca/ et http://pivert/ra/. Les utilisateurs doivent faire des demandes de certificat en accédant au répertoire /pub/ (exemple : http://pivert/pub/). OpenCA propose une passerelle vers OpenLDAP ; il est donc possible depuis OpenCA de publier tous les certificats de la base OpenCA dans un annuaire OpenLDAP ; cela se fait via le répertoire /ldap/. Bien entendu, pour une question de cohérence il est important que le nommage (DN) retenu soit identique dans OpenCA et OpenLDAP. En effet, le but est d'avoir un certificat dont le détenteur correspond au DN de l'entrée LDAP où il sera stocké.

9 Remerciements

Je remercie Nadia Rasamoely qui, au cours de son stage de DEA, a débuté les travaux sur LDAP [Ras02].

10 Glossaire

CA *Certification Authority*
ACL *Access Control List*
CRL *Certificate Revocation List*
IGC Infrastructure de Gestion de Clés
LDAP *Lightweight Directory Access Protocol*
PKI *Public Key Infrastructure*
RA *Registration Authority*
SASL *Simple Authentication and Security Layer*
TLS *Transport Layer Security*

11 Annexe 1 : format en ASN.1 de l'opération SignedOperations

```
SIGNEDOPERATIONS DEFINITIONS ::=

BEGIN
SignedOperation ::= CHOICE {
        signbyServer NULL,
        signatureIncluded OCTET STRING
        }
Changes ::= SEQUENCE {
        sequenceNumber [0] INTEGER (0 .. maxInt),
        signedOperation [1] OCTET STRING }
DemandSignedResult ::= LDAPSigType
LDAPSigType ::= BOOLEAN
SignedResult ::= CHOICE {
        signature OCTET STRING }
END
```

12 Références

[Ras02] Rasamoely N., « Gestion des certificats par LDAP », rapport de stage, septembre 2002.
[rfc1321] Rivest, R., "The MD5 Message-Digest Algorithm", rfc 1321, April 1992.
[rfc1847] J. Galvin, S. Murphy, S. Crocker N. Freed, « Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted », rfc 1847, Oct. 1995.
[rfc2222] J. Myers, « Simple Authentication and Security Layer (SASL) », rfc 2222, Oct. 1997.

[rfc2245] C. Newman, « Anonymous SASL Mechanism », rfc 2245, Nov. 1997.
[rfc2246] T. Dierks, C. Allen, « The TLS Protocol Version 1.0 », rfc 2246, Jan. 1999.
[rfc2251] M. Wahl, T. Howes, S. Kille, « Lightweight Directory Access Protocol (v3) », rfc 2251, Dec. 1997.
[rfc2256] M. Wahl, « A summary of the X.500(96) User Schema for use with LDAPv3 », rfc 2256, Dec. 1997.

[rfc2311] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka, « S/MIME Version 2 Message Specification », rfc 2311, March 1998.
[rfc2459] R. Housley, W. Ford, W. Polk, D. Solo, « Internet X.509 Public Key Infrastructure Certificate and CRL Profile », rfc 2459, Jan. 1999.
[rfc2649] B. Greenblatt, P. Richard, « An LDAP Control and Schema for Holding Operation Signatures », rfc 2649, August 1999.

[rfc2820] E. Stokes, D. Byrne, B. Blakley, P. Behera, « Access Control Requirements for LDAP », rfc 2820, May 2000.
[rfc2829] M. Wahl, H. Alvestrand, J. Hodges, R. Morgan, « Authentication Methods for LDAP », rfc 2829, May 2000.
[rfc2830] Hodges, R. Morgan, M. Wahl, « Lightweight Directory Access Protocol (v3) : Extension for Transport Layer Security », rfc 2830, May 2000.
[rfc2831] P. Leach, C. Newman, « Using Digest Authentication as a SASL Mechanism », rfc 2831, March 1997.

[rfc2559] S. Boeyen, T. Howes, P. Richard, « Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2 », rfc 2559, April 1999.
[rfc2560] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, « X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP », rfc 2560, June 1999.
[rfc2587] S. Boeyen, T. Howes, P. Richard, « Internet X.509 Public Key Infrastructure LDAPv2 Schema », rfc 2587, June 1999.
[rfc3112] K. Zeilenga, « LDAP Authentication Password Schema », rfc 3112, May 2001.
[SHA1] NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.