Ecole d'Ingénieurs de Genève **laboratoire de transmission de données**

Public Key Infrastructure

Etudiant : Denis Cotte
Professeur : Mr Gérald Litzistorf
Projet réalisé en collaboration avec e-Xpert solutions

04/12/2001

e i g
Ecole d'ingénieurs - hes
Genève

4, rue de la Prairie
1202 Genève

tél. 022 338 04 00
fax 022 338 04 10
info@eig.unige.ch

Département des technologies de l'information
Session d'automne 2001

Candidat .
M. COTTE Denis
Filière informatique

En collaboration avec .
e-Xpert Solutions SA
Professeur :
M. Gérald Litzistorf

Diplôme en transmission de données

Infrastructure à clé publique

Description :

Une infrastructure à clé publique (*Public Key Infrastructure*) crée un espace de confiance qui permet de gérer tous les aspects de sécurité : authentification des utilisateurs et des entités techniques, confidentialité, intégrité des données et non répudiation des transactions.

Elle est constituée par des services de génération et de diffusion des certificats numériques (sorte de carte d'identité virtuelle) et des clés nécessaires au chiffrement et au déchiffrement.

Ce travail se décompose selon les étapes décrites ci-dessous.

Travail demandé au candidat :

Le candidat commencera par étudier les différents protocoles, procédures et formats présents dans une infrastructure à clé publique pour délivrer un certificat, le publier, le bloquer et tester sa validité :

- *Public Key Cryptography Standard*
- *Simple Certificate Enrolment Protocol*
- *Certificate Management Protocol*
- *Certificate Revocation List*
- *Online Certificate Status Protocol*

La mise en œuvre sera basée sur l'autorité de certification *Keon Certificate Authority 5.7* de RSA (compatible NT4 serveur) et divers composants du laboratoire : serveur SSL, modules IPSec (Checkpoint - Cisco) et cartes à puce.

Le travail, proposé par la société e-Xpert, consiste à développer un module pour serveur *web Apache* capable de fonctionner comme client OCSP *(Online Certificate Status Protocol)* afin de pouvoir valider le certificat de l'utilisateur lors de son authentification.

Plate-forme imposée pour le serveur *web Apache 1.3.20* : système Linux avec distribution SuSE

Sommaire

1 Avant propos

1.1 Conventions typographiques

Afin de faciliter la lecture de ce document et de lui donner autant de cohérence et d'homogénéité que possible nous avons adopté les règles typographiques suivantes :

- Le texte courant est écrit en Times New Roman
- Les termes spécifiques à un domaine particulier ainsi que les termes en anglais sont écrits en *italique*
- Les extraits de *Request for comment*, les commandes ainsi que le code, sont écrits en `Courier`
- Les références Littéraires [L], RFC [RFC] sont exprimées [entre crochets], leur définition apparaît dans les références.

Tout au long de ce document une attention particulière est à porter aux éléments **en gras**, ils sont utilisés pour désigner les points les plus importants.

1.2 Remerciements

Au terme de ce travail, je tiens à remercier toutes les personnes qui se sont intéressées à ce projet et qui, par leur aide ou leur soutien, m'ont encouragé et motivé tout au long de ce travail.

- Sylvain Maret de la société e-Xpert Solutions SA pour sa collaboration, le temps qu'il m'a consacré dans le cadre de ce projet ainsi que les diverses sources d'informations qu'il m'a fournit.
- Gérald Litzistorf, pour sa rigueur et l'intérêt qu'il a porté à ce travail.
- Docteur Andrea Giacobazzi, pour son aide précieuse et le code de son patch OCSP qu'il m'a transmit.
- Sébastien Borsa pour ces informations sur les certificats numériques et sur le protocole SCEP.
- Mes collègues diplômants pour l'ambiance conviviale qu'ils ont fait régné au sein du laboratoire.

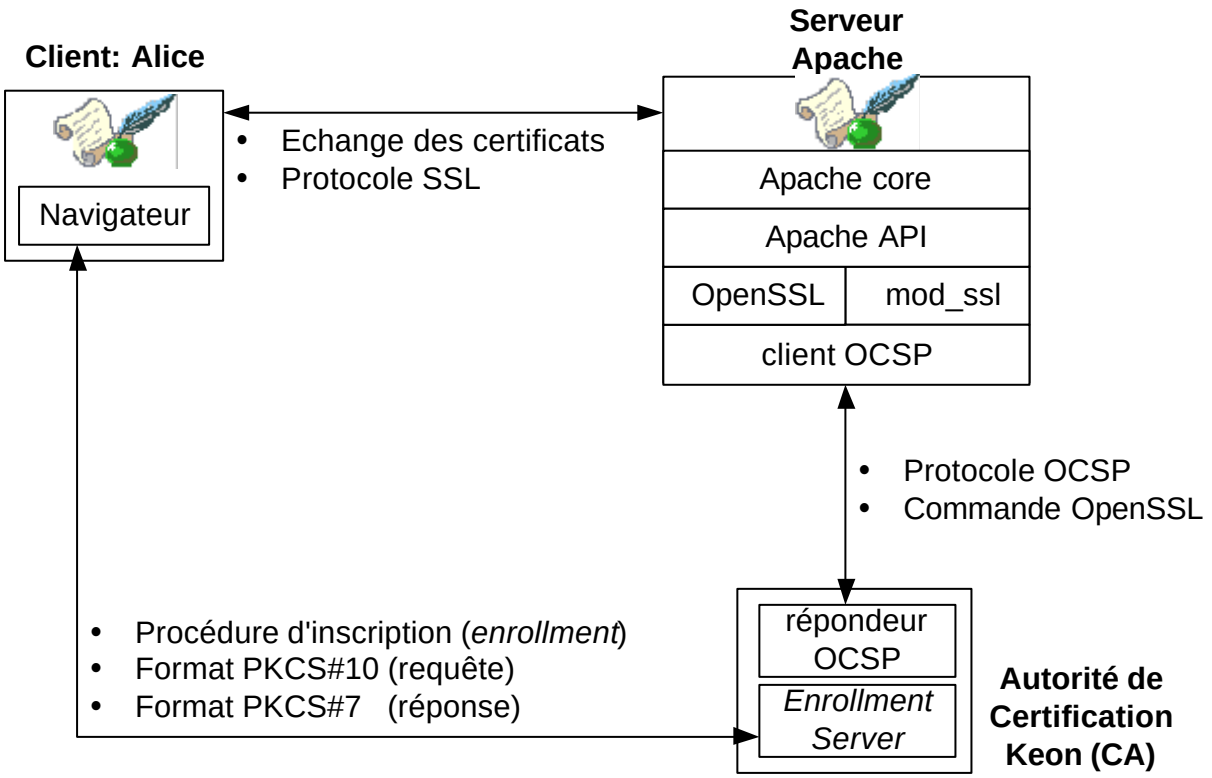
2 Introduction

La montée en puissance du commerce électronique impose la mise en place d'infrastructures à clé publique (PKI), et l'interopérabilité entre celles ci. De plus en plus d'entreprises utilisent le *e-business* et le nombre d'applications qui utilisent les clés publiques et les certificats numériques pour sécuriser des transactions grandis rapidement.

Dans ce mémoire nous allons tout d'abord nous intéresser à la théorie des infrastructures à clé publique. Puis nous rentrerons plus en détails sur la phase qui consiste à se faire publier un certificat numérique par une autorité de certification, on l'appelle la procédure d'inscription (en anglais *enrollment procedure*). Nous analyserons les différents protocoles et formats normalisés pour effectuer cette transaction.
Ensuite, nous installerons et configurerons un serveur Apache avec un module qui permet l'utilisation du protocole SSL pour effectuer des échanges sécurisés. Enfin nous implémenterons sur ce module la fonction OCSP chargée de vérifier la validité des certificats que le serveur reçoit de la part de ses clients.

La figure suivante résume l'architecture du travail. Le client est communément appelé Alice.
L'autorité de certification est une CA Keon de RSA Security et le serveur Apache est la dernière version disponible 1.3.22. Plus d'informations sur la configuration sont disponibles en ***Annexe A.***

Architecture du travail de diplôme



3 PKI : Infrastructure à Clé Publique

3.1 Définition

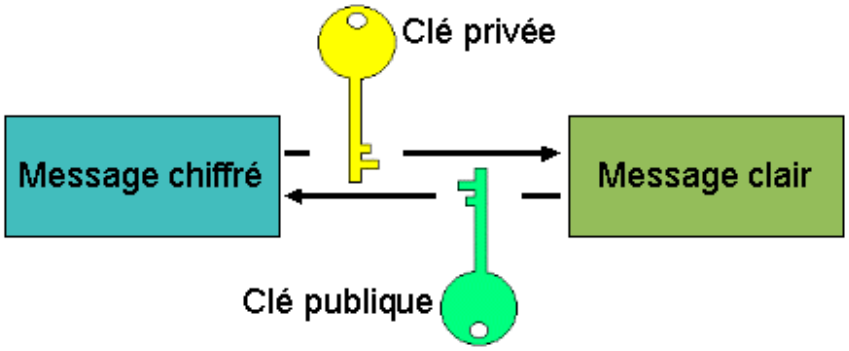
On appelle PKI (en français ICP pour Infrastructure à Clé Publique) l'ensemble des solutions techniques basées sur la cryptographie à clés publiques.

Les systèmes de cryptographie à clé publique résolvent le difficile problème de la transmission de la clé secrète d'un algorithme symétrique; ils ne nécessitent pas la connaissance d'une information secrète pour déchiffrer un message codé. Un utilisateur possèdera donc une paire de clés, l'une privée qu'il conserve en lieu sûr, et l'autre publique qui sera accessible à quiconque devra l'utiliser. Ces deux clés sont liées mathématiquement de telle sorte qu'il soit extrêmement difficile de retrouver par calcul la clé privée à partir de la clé publique.

3.1.1 Confidentialité

On obtiendra la **confidentialité** d'un message en utilisant la clé publique du destinataire comme clé de chiffrement. En effet, seul le destinataire possède la clé privée correspondante autorisant le déchiffrement du message.

Figure 3.1 Chiffrement/Déchiffrement d'un message

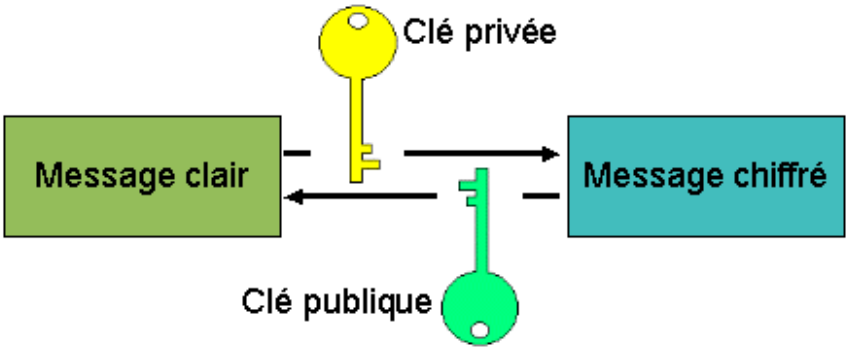


Le chiffrement du message ne nécessite que la connaissance de la clé publique du destinataire qui pourra décoder le message chiffré à l'aide de sa clé privée.

3.1.2 Authentification

L'authentification d'un message est également possible dès lors que l'expéditeur utilise sa clé privée pour chiffrer le message qu'il désire envoyer. Le destinataire utilisera la clé publique correspondante pour déchiffrer le message qui sera ainsi authentifié. Remarquons que bien que le message soit chiffré, il n'est pas confidentiel car n'importe qui peut utiliser la clé publique de l'expéditeur pour en prendre connaissance; il n'est donc que signé par le propriétaire de la clé privée. Le destinataire est alors confronté au problème qui consiste à se convaincre de l'identité de l'utilisateur de la clé privée qui a servi à signer le message. Les certificats numériques y apportent une réponse acceptable en juxtaposant une clé publique et des informations permettant de déterminer l'identité d'une personne.

Figure 3.2 Chiffrement/Déchiffrement de la signature



Une signature réalisée en chiffrant le message à l'aide de la clé privée de l'expéditeur pourra être vérifiée par le destinataire en employant la clé publique de l'expéditeur.

Notons que l'expéditeur d'un message signé de la sorte peut le rendre confidentiel en chiffrant le résultat une seconde fois à l'aide de la clé publique du destinataire. Celui-ci le déchiffrera en deux temps: en utilisant d'abord sa clé privée, puis en utilisant la clé publique de l'expéditeur.

L'infrastructure PKI permet de sécuriser un système d'information grâce à un précepte de cryptographie asymétrique (clé publique/clé privée). Elle permet notamment de mettre en place un système de gestion centralisé des clés pour tous les utilisateurs, celles-ci devant être certifiées par des autorités.

On défini un tiers de confiance comme une entité (appelée communément autorité de certification**,** en anglais *Certification Authority* abrégé *CA*) chargée d'assurer la véracité des informations contenues dans le certificat de clé publique et de sa validité. Pour ce faire, l'autorité signe le certificat avec sa propre clé privée.

Les crypto systèmes à clés publiques permettent de s'affranchir de la nécessité d'avoir recours systématiquement à un canal sécurisé pour s'échanger les clés. En revanche, la publication de la clé publique à grande échelle doit se faire en toute confiance pour assurer que :

- La clé publique est bien celle de son propriétaire
- Le propriétaire de la clé est digne de confiance
- La clé est toujours valide

Ainsi, il est nécessaire d'associer au bi-clé (ensemble clé publique/clé privée) un certificat délivré par un tiers de confiance : l'infrastructure de gestion de clés.

3.2 La notion de certificat

Les algorithmes de chiffrement asymétrique sont basés sur le partage entre les différents utilisateurs d'une clé publique. Généralement le partage de cette clé se fait au travers d'un annuaire électronique généralement au format LDAP.

Toutefois ce mode de partage a une grande lacune : rien ne garantit que la clé est bien celle de l'utilisateur à qui elle est associée. En effet un pirate peut corrompre la clé publique présente dans l'annuaire en la remplaçant par sa clé publique. Ce pirate sera en mesure de déchiffrer tous les messages ayant été chiffrés avec la clé présente dans l'annuaire.

Ainsi un certificat permet d'associer une clé publique à une entité (une personne, une machine) afin d'en assurer la validité. Le certificat est en quelque sorte la carte d'identité de cette entité, délivré par l'organisme appelé autorité de certification.

L'autorité de certification est chargée de délivrer les certificats, de leur assigner une date de validité (équivalent à la date limite de péremption des produits alimentaires), ainsi que de révoquer éventuellement des certificats avant cette date en cas de compromission de la clé (ou du propriétaire).

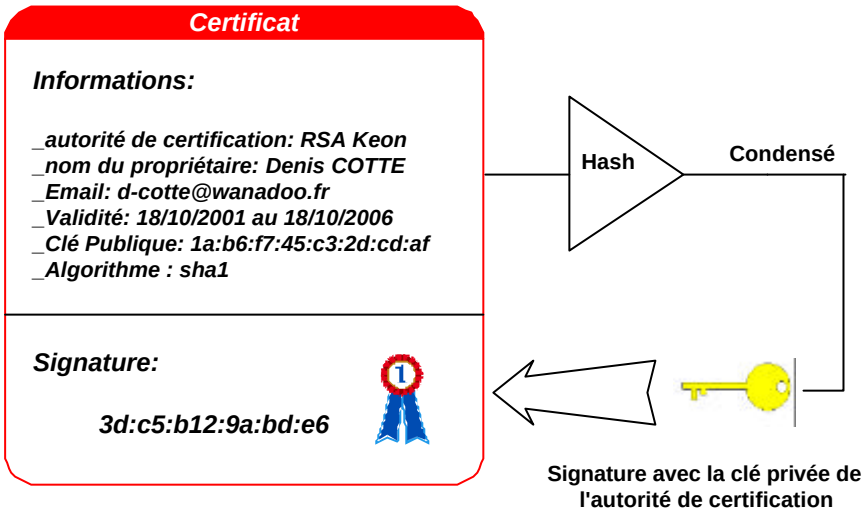
Les certificats sont des petits fichiers divisés en deux parties :

- Une partie contenant des informations
- Une partie contenant la signature de l'autorité de certification

La structure des certificats est normalisée par le standard **X.509** de l'UIT, qui définit les informations contenues dans le certificat :

- Le nom de l'autorité de certification
- Le nom du propriétaire du certificat
- La date de validité du certificat
- L'algorithme de chiffrement utilisé
- La clé publique du propriétaire

Figure 3.3 Structure d'un certificat X.509



Certificat Numérique contenant des informations et une signature chiffrée avec la clé privée de la CA.

L'ensemble de ces informations est signé par l'autorité de certification, cela signifie qu'une fonction de hachage crée une empreinte de ces informations, puis ce condensé est chiffré à l'aide de la clé privée de l'autorité de certification.

Lorsqu'un utilisateur désire communiquer avec une autre personne, il lui suffit de se procurer le certificat du destinataire. Ce certificat contient le nom du destinataire, ainsi que sa clé publique et est signé par l'autorité de certification. Il est donc possible de vérifier la validité du message en appliquant d'une part la fonction de hachage aux informations contenues dans le certificat, en déchiffrant d'autre part la signature de l'autorité de certification avec la clé publique de cette dernière et en comparant ces deux résultats.

Figure 3.4 Vérification de la validité du certificat.

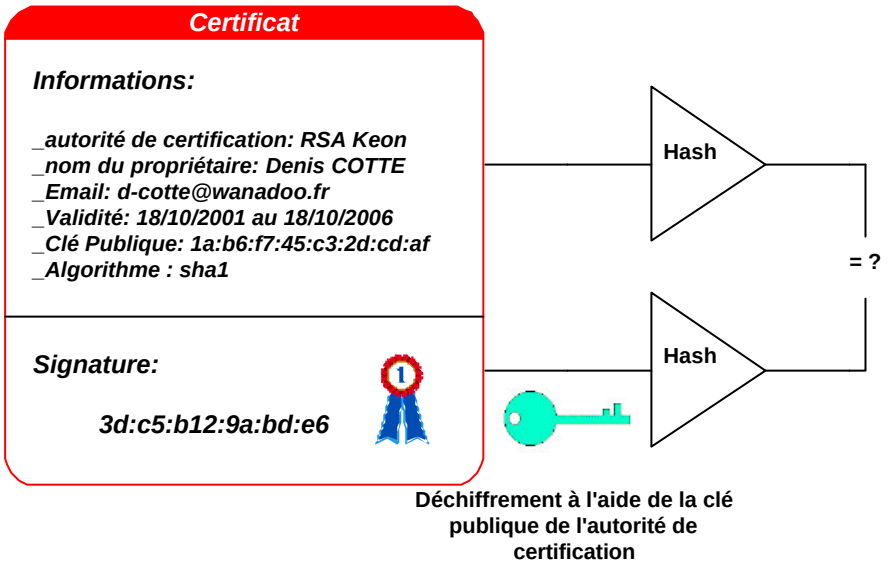


Schéma de principe pour vérifier un Certificat Numérique.

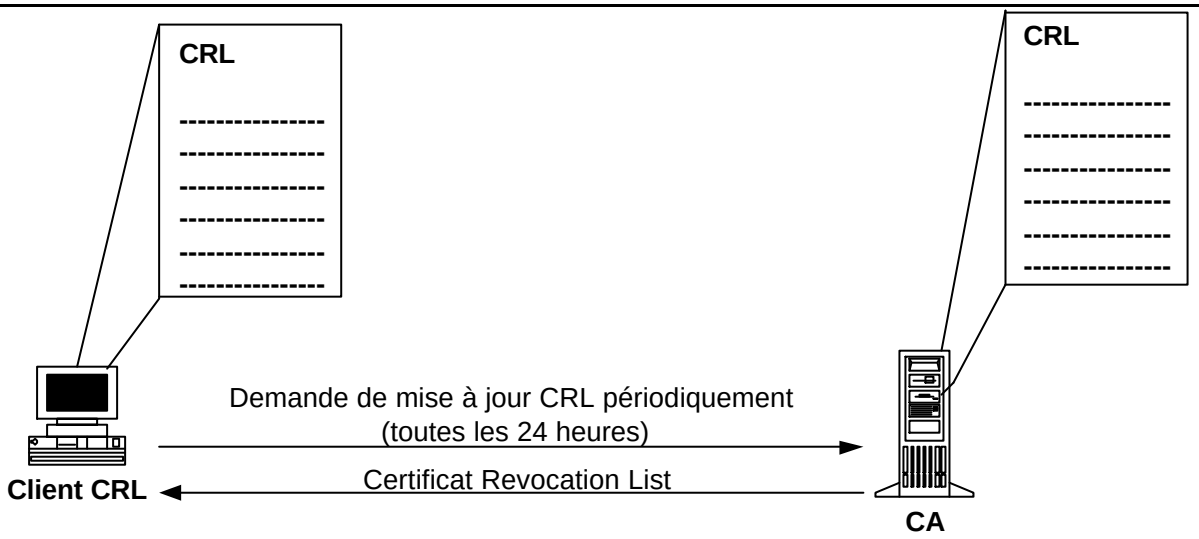
Il y a actuellement deux principaux algorithmes de calcul d'empreintes numériques (*hash*) parfois appelées "*fingerprint*" ou "*thumbprint*". Le premier SHA-1 (utilisé par Internet Explorer) est appelé « *thumbprint* » et le deuxième MD5 (utilisé par PGP et Netscape) est appelé « *fingerprint* ».

3.3 Validité des certificats numériques

La validité d'un certificat est généralement désignée par **une intervalle de temps défini** entre une date de publication (naissance du certificat) et une date d'expiration paramétrable par la CA.

L'autorité de certification est chargée de délivrer les certificats, de leur assigner leur date de validité, ainsi que d'éventuellement révoquer des certificats avant cette date en cas de **compromission de la clé ou du propriétaire**.
Les certificats corrompus sont stockés dans une liste de certificat révoqué CRL (*Certificat Revocation List)*. Le certificat devient invalide et inutilisable (un peu comme on peut faire opposition sur une carte bancaire). Faire confiance à une autorité de certification impose de charger régulièrement les listes de révocation émises par cette autorité. Ces listes de révocation sont actualisées dès qu'un certificat a été révoqué et au moins chaque mois (passé ce délais, les CRLs sont paramétrées pour expirer) et votre navigateur doit vous prévenir.

Figure 3.5 Principe d'une liste de certificats révoqués



Vérification de la validité d'un certificat avec les CRLs.

Deux options d'Internet Explorer, permettent de mettre à jour les CRLs. Pour cela il faut dans la fenêtre d'Internet Explorer choisir « *Tools* »-« *Internet Options* »-« *Advanced* » puis descendre jusqu'au paragraphe « *Security* » et enfin cocher les deux options :

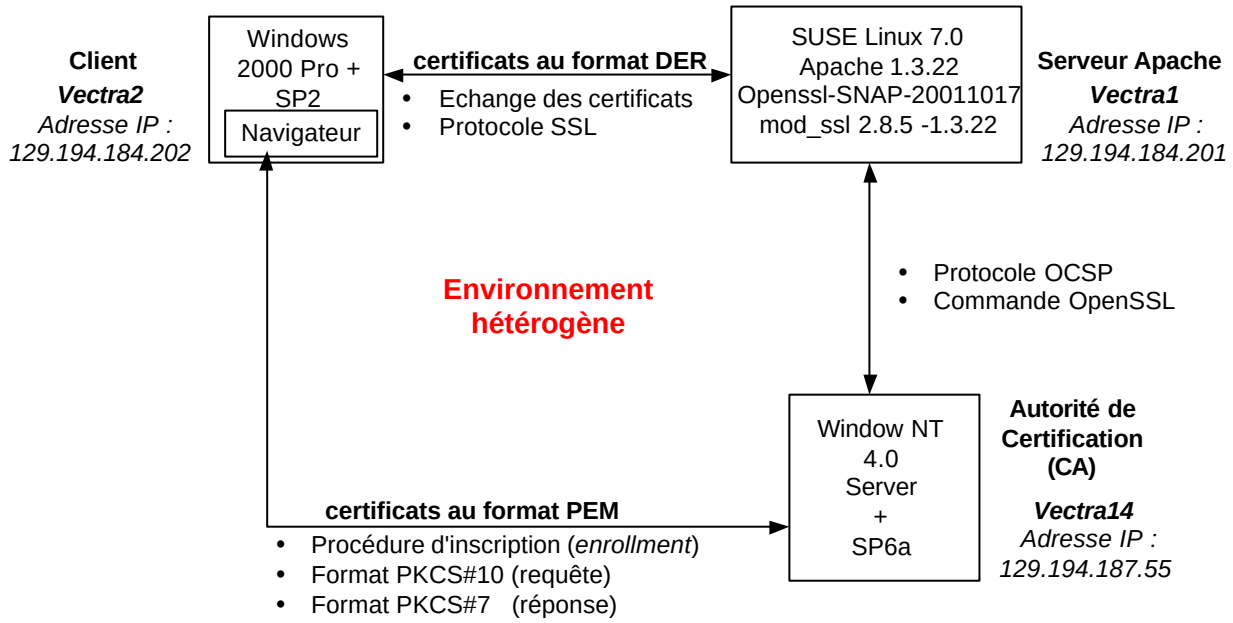
- *Check for publisher's certificat revocation*
- *Check for server certificat revocation (requires restart)*

3.4 Formats ASN1, PEM et DER

3.4.1 Architecture générale

La figure suivante représente l'architecture générale du travail de diplôme. Nous avons trois machines qui fonctionnent sous des environnements différents, il s'agit d'un système hétérogène. Nous allons tout d'abord nous intéresser aux différents formats disponibles pour représenter un certificat numérique lors d'échanges entre les différentes entités.

Figure 3.6 Architecture générale



On distingue de suite deux formats différents, le **format DER** et le **format PEM.** Le format DER est utilisé pour transmettre le certificat entre le Client et le serveur Apache alors que le format PEM est utilisé entre le Client et l'autorité de certification. Nous allons maintenant étudier ces différents formats.

3.4.2 Format ASN1

La **notation formelle** ASN1 (*Abstract Syntax Notation 1*) permet de **spécifier les données transmises par les protocoles** de télécommunications indépendamment des langages informatiques et de la représentation physique de ces données, pour toutes sortes d'applications communicantes et de données aussi complexes (ou aussi simples) soient-elles. Cette représentation est souvent utilisée dans les RFC (*Request For Comment)* pour décrire une structure de données.

La notation offre un certain nombre de types prédéfinis tels que :

- les entiers (`INTEGER`),
- les booléens (`BOOLEAN`),
- les chaînes de caractères (`IA5String`, `UniversalString`...),
- les chaînes de bits (`BIT STRING`),
- etc,

et permet de définir des types composés tels que :

- des structures (`SEQUENCE`),
- des listes (`SEQUENCE OF`),
- un choix de types (`CHOICE`),
- etc.

Exemple pour un certificat:

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signature            BIT STRING
}
```

Pour chaque type, des contraintes de sous typage permettent de restreindre l'ensemble des valeurs qui constitue son domaine.

La notation ASN.1 couvre uniquement les aspects de structuration de l'information (il n'existe pas d'opérateurs pour manipuler les valeurs ou faire des calculs sur celles-ci). Ce n'est donc **pas** un **langage de programmation**.

3.4.3 Format DER

Le passage de ASN.1 à un certificat numérique se fait en appliquant les règles d'encodage ***DER*** (*Distinguished Encoding Rules*) qui sont un sous-ensemble des règles BER (*Basic Encoding Rules*) décrites dans les recommandations ITU-T X.208. Chaque valeur ASN est représentée comme une chaîne unique d'octet au format DER. Le codage DER est utilisé notamment pour le commerce électronique ce qui implique de la cryptographie et surtout le fait qu'il y ai une et seulement une façon de coder et décoder un message.

Les éléments ASN.1 encodés avec ***DER*** sont des triplets *tag/longueur/valeur* représentés par une suite d'octets, le premier, appelé *identifier octet*, comporte trois champs:

Bit	8 7	6	5 4 3 2 1
	Classe	Composé	Numéro de tag

où les différentes classes (bits 8 et 7) sont:

00	Universel	Identique pour toutes les applications; défini dans X.208
01	Application	Spécifique à une application
10	Privé	Spécifique à une entreprise
11	Dépendant du contexte	Spécifique à un type structuré

voici quelques numéros de tag:

`BOOLEAN`	1
`INTEGER`	2
`BIT STRING`	3
`OCTET STRING`	4
`NULL`	5
`OBJECT IDENTIFIER`	6
`SEQUENCE`	16
`SET`	17

L'octet suivant indique la longueur (en octets) de la suite d'octets à interpréter en tant que valeur. Si la longueur est supérieure à 127, le bit 8 est mis à *1*, les bits 1 à 7 indiquent le nombre d'octets à lire pour déterminer la longueur. Le dernier octet de longueur est suivi des octets représentant la valeur encodée.

Voyons le contenu du certificat numérique dont un extrait de la **représentation hexadécimale** est *30 82 03 77 :*

30_{16} (Certificate ::= SEQUENCE {

- l'identifier octet est $30_{16} = 0011\ 0000_2$ ce qui se décode comme classe universel (00) composé (1) SEQUENCE ($10000_2 = 16_{10}$).
- la longueur est $82_{16} = 1000\ 0010_2$, comme le bit 8 est à 1, et que les bits 1..7 donnent la valeur 2, les 2 prochains octets $0377_{16} = 887_{10}$ indiquent la longueur de la suite d'octets représentant la valeur de « Certificate ».

3.4.4 Format PEM

"*Privacy Enhanced Mail*" (en français messagerie à confidentialité ajoutée) est un système sécurisé pour le courrier électronique, garantissant le caractère confidentiel ou privé des données transmises par courrier électronique. Les techniques "habituelles" de courrier électronique ne sont pas sécurisées.
Le format DER des certificats numériques se prête mal aux applications de type courrier électronique car il contient des octets qui pourraient être "malmenés" par certains logiciels de transfert de messages.
Le format PEM y remédie en utilisant **l'encodage base 64**. Un exemple de certificat au format PEM est composé de caractères ASCII :

```
-----BEGIN CERTIFICATE-----
MIIDdzCCAuCgAwIBAgIBBzANBgkqhkiG9w0BAQQFADCBpDELMAkGA1UEBhMCQ0gx
CzAJBgNVBAgTAkdFMQ8wDQYDVQQHEwZHZW5ldmExHTAbBgNVBAoTFFVuaXZlcnNp
dHkgb2YgR2VuZXZhMSQwIgYDVQQLExtVTklHRSBDZXJ0aWZpY2F0ZSBBdXRob3Jp
dHkxETAPBgNVBAMTCFVuaUdlIENBMR8wHQYJKoZIhvcNAQkBFhB1bmlnZWNhQHVu
aWdlLmNoMB4XDTk5MTAwNDE2MjI1N1oXDTAwMTAwMzE2MjI1N1owgbExCzAJBgNV
BAYTAkNIMQswCQYDVQQIEwJHRTEPMA0GA1UEBxMGR2VuZXZhMR0wGwYDVQQKExRV
bml2ZXJzaXR5IG9mIEdlbmV2YTEeMBwGA1UECxMVRGl2aXNpb24gSW5mb3JtYXRp
cXVlMRowGAYDVQQDExFBbGFpbiBIdWdlbnRvYmxlcjEpMCcGCSqGSIb3DQEJARYa
QWxhaW4uSHVnZW50b2JsZXJAdW5pZ2UuY2gwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBALIL5oX/FR9ioQHM0aXxfDELkhPKkw8jc6I7BtSYJk4sfqvQYqvOMt1u
ugQGkyIuGhP2djLj6Ju4+KyKKQVvDJIu/R1zFX1kkqOPt/A2pCLkisuH7nDsMbWb
ep0hDTVNELoKVoVIAazwWMFlno2JuHJgUcs5hWskg/azqI4d9zy5AgMBAAGjgakw
gaYwJQYDVR0RBB4wHIEaQWxhaW4uSHVnZW50b2JsZXJAdW5pZ2UuY2gwDAYDVR0T
BAUwAwIBADBcBglghkgBhvhCAQ0ETxZNVU5JR0VDQSBjbGllbnQgY2VydGlmaWNh
dGUsIHNlZSBodHRwOi8vdW5pZ2VjYS51bmlnZS5jaCBmb3IgbW9yZSBpbmZvcm1h
dGlvbnMwEQYJYIZIAYb4QgEBBAQDAgSwMA0GCSqGSIb3DQEBBAUAA4GBACQ9Eo67
A3UUa6QBBNJYbGhC7zSjXiWySvj6k4az2UqTOCT9mCNnmPR5I3Kxr1GpWToH68Lv
A30inskP9rkZAksPyaZzjT7aL//phV3ViJfreGbVs5tiT/cmigwFLeUWFRvNyT9V
UPUov9hGVbCc9x+v05uY7t3UMeZejj8zHHM+
-----END CERTIFICATE-----
```

Un certificat numérique au format PEM possède la forme suivante:
-----BEGIN CERTIFICATE-----
données encodées base64
-----END CERTIFICATE-----

La longueur des lignes (à l'exception de la dernière) est de 64 caractères ; lesquelles n'utilisent que les 64 caractères [A-Za-z0-9/+] qui devraient résister aux manipulations des logiciels de courrier électronique.

A titre d'exemple, un convertisseur base64 est disponible à l'adresse suivante :
http://www.wc.cc.va .us/dtod/base64
Le lien suivant donne la table ASCII qui peut être utile pour comprendre le codage:
http://www.asciitable.com/

La représentation base64 est basée sur un alphabet de 64 caractères. Cet alphabet est représenté dans le tableau ci-dessous :

SEQUENCE	CHARACTERS
0 .. 25	'A' .. 'Z'
26 .. 51	'a' .. 'z'
52 .. 61	'0' .. '9'
62	'+'
63	'/'

Pour transcrire un fichier binaire, il faut regrouper les bits par groupe de 6. On retrouve un nombre décimal entre 0 et 63 pour chaque groupe. On convertit ce nombre en utilisant la table d'équivalence de l'alphabet base64. Le résultat est une suite de caractères lisibles. Voici un exemple de conversion d'une chaîne de bits en notation base64.

'001100110011'

Cette chaîne de bits peut être divisée en deux parties.

'001100' et '110011'

En convertissant ces nombres binaires de 6 bits en notation décimale (base-10) on obtient les nombres 12 et 51. Maintenant, en regardant la correspondance avec l'alphabet base64, on peut donner le résultat suivant :

Base64_encode(*'001100110011'*) = *'Mz'*

Prenons un autre exemple, la représentation suivante :

'01010101' '00100100' '00010001'

En hexadécimal ce nombre est *'552411'*. En regroupant ces 3 bytes en série de 6-bits, on a alors exactement quatre caractères base64.

'010101' '010010' '010000' '010001'

En base-64, cette représentation signifie *'VSQR'*.

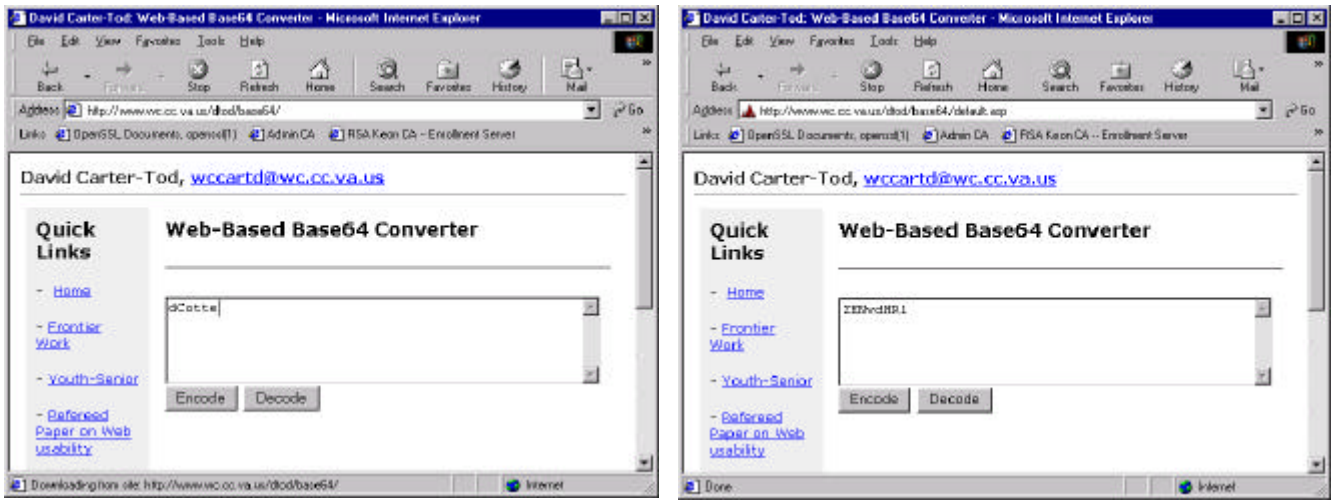
Dans le cas où la représentation binaire n'a pas une taille intégrale de 3 octets à la fin d'un message, il se présente un problème de représentation. Après regroupement par série de 3 octets équivalent à 4 caractères base64, il faut combler le message afin de finir les 3 derniers octets. Pour cela, on utilise le caractère de remplissage base64 qui est *'='.*
C'est pourquoi, un message au format PEM se termine parfois par un ou deux caractères *'='.*

Terminons par un exemple simple, codons la chaîne de caractère « dCotte »

caractère	D	**C**	O	**t**	t	**E**		
ASCII	100	**67**	111	**116**	116	**101**		
Hexa	64	**43**	6F	**74**	74	**65**		
Binaire	01100100	**01000011**	01101111	**01110100**	01110100	**01100101**		
Regroupement (par 6 bits)	011001	000**100**	**011**01	101111	**011101**	**000**111	0100**01**	**100101**
Décodage	25	4	13	47	29	7	17	37
Base64	Z	E	N	V	D	H	R	l

La chaîne de caractère « dCotte » se code en base64 par la chaîne de caractère « ZENVdHRl»
Vérifions à l'aide du site http://www.wc.cc.va.us/dtod/base64

Figure 3.7 Vérification du codage base64



3.5 Générer un certifcat avec OpenSSL

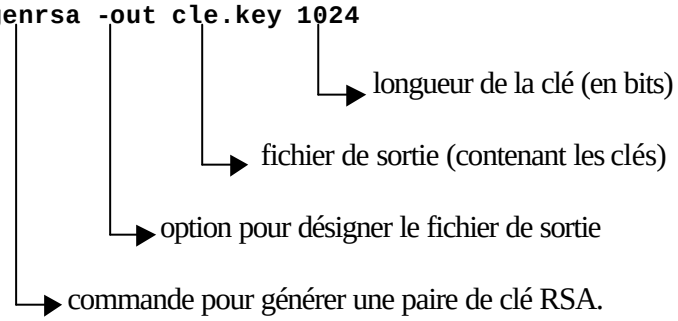
OpenSSL est un outil pédagogique permettant d'effectuer différentes d'opérations en matière de sécurité (SSL, cryptage, génération de certificat ….) OpenSSL est géré par une communauté mondiale de volontaires qui utilisent l'Internet pour communiquer et développer ce « *toolkit* OpenSSL » et sa documentation. OpenSSL est basé sur la bibliothèque SSLeay développée par Eric A. Jeune et Tim J. Hudson. Nous illustrerons souvent les différents points de ce mémoire avec des commandes de type OpenSSL.

Cet outil est disponible sur le lien suivant :
http://www.openvalidation.org/openssldownload.htm
Deux versions sont disponibles: pour Windows et une autre pour Linux.

Voici les commandes OpenSSL pour générer un certificat :

Tout d'abord, il nous faut une clé privée (secrète), la commande pour générer celle ci est la suivante :

```
> openssl genrsa -out cle.key 1024
```

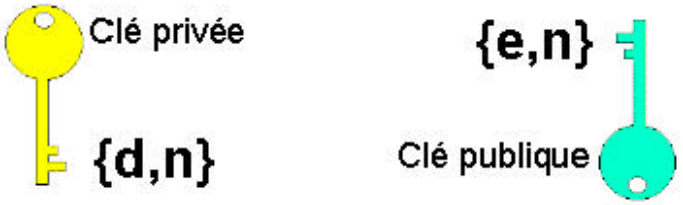


Reprenons la théorie sur le crypto système RSA, les étapes de la génération d'une paire de clés sont les suivantes:

- Prendre deux grands nombres premiers p et q aléatoire
- Calculer le "modulus" $n=p\cdot q$
- Calculer $phi(n)=(p-1)\cdot(q-1)$
- Choisir un e aléatoire ($1<e<phi(n)$) tel que $pgcd(e,phi(n))=1$ c'est à dire un e premier avec $phi(n)$
- Calculer $d=e^{-1} mod\ phi(n)$ l'inverse de e en utilisant l'algorithme d'Euclide.

- Publier $\{e,n\}$ qui fera office de clé publique, conserver **la clé privée** $\{d,n\}$ en lieu sûr et détruire $p, q, phi(n)$

Figure 3.6 Equivalence clé publique/clé privée



Equivalence entre les clés et les variables de l'algorithme.

- Une fois la **clé publique** $\{e,n\}$ transmise au partenaire, celui ci pourra chiffrer des messages à destination du détenteur de la clé privée $\{d,n\}$ en calculant:

$$C = M^e \bmod n$$

où M est le message en clair et C le message chiffré. Le déchiffrage de C se fera en calculant:

$$M = C^d \bmod n$$

Prenons un exemple:

choisissons 2 nombres premiers

$p=7$ et $q=13$

calculons

$n = p \cdot q = 7 \cdot 13 = 91$ et
$phi(n)=(p-1)\cdot(q-1)= 6 \cdot 12 = 72$

choisissons

$e=5$ qui satisfait à: $1<e<72$ et $pgcd(72,5)=1$

calculons

$d = e^{-1} \bmod 72 = 29$ car $29 \cdot 5 = 145 = 2 \cdot 72 + 1$
$\{5, 91\}$ est la clé publique et
$\{29, 91\}$ est la clé privée

essayons maintenant de chiffrer le message $M=17$ à l'aide de la clé publique

$$C = M^e \bmod n = 17^5 \bmod 91 = 75$$

que nous pouvons déchiffrer à l'aide de la clé privée

$$M = C^d \bmod n = 75^{29} \bmod 91 = 17$$

On génère donc une paire de clé avec la commande OpenSSL précédentes. On peut visualiser les informations du fichier de sortie nommé `cle.key` avec la commande suivante :

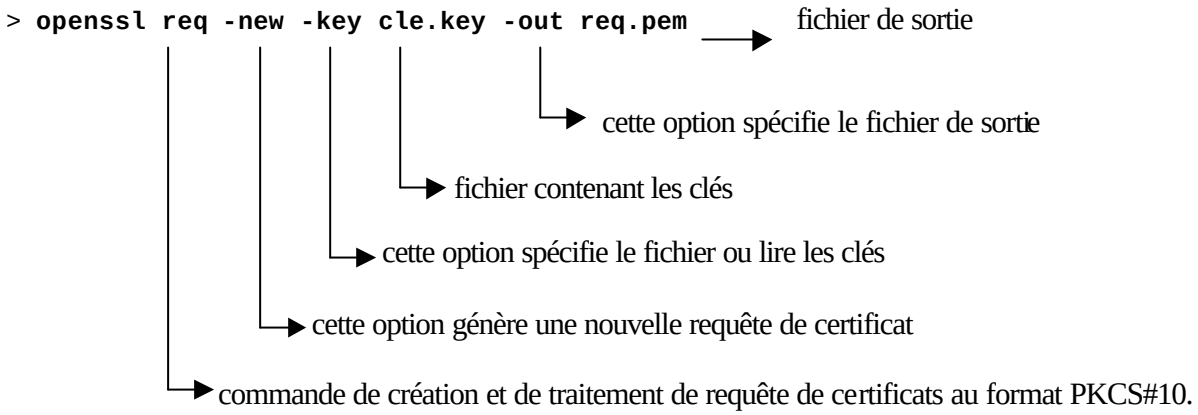
```
> openssl rsa -in cle.key –text –noout
```

- `openssl` : commande de traitement des clés RSA
- `rsa` : commande de traitement des clés RSA
- `-in` : option désignant un fichier en entrée
- `cle.key` : fichier d'entrée
- `-text` : option de visualisation des informations du fichier d'entrée
- `-noout` : option signifiant qu'il n'y a de fichier de sortie

Le résultat est le suivant :

```
read RSA key
Private-Key: (1024 bit)
modulus:
    00:c1:d8:8d:25:a3:b7:31:b8:20:d1:74:6e:e9:db:
    79:1e:72:37:a8:dc:8c:e1:fc:e8:09:f9:c1:d3:bc:
    60:24:82:39:12:f1:b4:cb:ec:0f:9a:d6:2a:35:fb:
    aa:0f:44:65:6f:df:30:72:1e:2c:20:7f:43:6d:3b:     //Valeur de n=p*q
    db:e2:40:2f:65:06:bc:db:1c:5d:86:78:32:72:50:
    15:39:b5:f4:b6:3f:22:31:f0:6a:9f:28:24:9d:c5:
    3d:3b:ef:a9:8f:5f:8d:d9:a9:ca:61:85:a5:ff:97:
    3b:1a:97:45:85:82:63:4b:f4:6c:12:0a:e7:13:08:
    fd:17:83:cc:cb:22:3d:f9:83
publicExponent: 65537 (0x10001)                    //Valeur de e
privateExponent:
    2b:9a:a0:b5:74:cc:42:9c:de:94:ff:11:eb:fc:f8:
    93:c6:b1:8a:84:82:14:5b:a5:7e:88:f5:f6:c1:0b:
    07:6b:5b:97:4d:53:94:03:77:c7:26:a1:bc:1e:ee:
    34:1c:f8:8c:5f:b2:30:19:65:67:b1:f8:e2:db:72:     //Valeur de d
    2c:c4:af:64:2f:be:b1:69:62:88:de:60:d7:13:0c:
    88:5a:e0:17:35:80:2f:dc:12:44:8a:ff:e2:98:c4:
    45:45:45:72:d2:88:9e:42:b7:45:66:e8:cf:5b:b2:
    ba:3d:95:0e:ef:55:1d:15:fe:2f:d7:b8:3b:b6:42:
    17:50:99:bb:be:b4:f8:e1
prime1:
    00:f1:db:3b:0d:64:56:2e:36:6e:ff:d0:d5:e5:a3:
    48:76:7b:a6:83:63:57:d3:39:70:7d:d6:98:85:e0:     //Valeur de p
    83:a4:97:cf:91:fb:bc:87:d6:40:75:e3:86:57:0e:
    ef:c5:42:16:3f:29:a0:d1:f3:13:9f:af:da:a3:5e:
    dc:d7:3d:2f:2b
prime2:
    00:cd:2e:91:81:45:e7:83:27:fd:d3:f9:08:dd:43:
    d0:1d:0e:81:35:e8:99:cb:8d:22:15:95:00:cf:e1:     //Valeur de q
    90:66:e3:45:63:ce:5d:56:c0:ce:c8:f3:73:99:3d:
    7c:c2:b8:74:69:98:0e:6a:4d:db:1b:f5:b9:4e:05:
    74:73:35:73:09
exponent1:
    00:91:93:fc:7f:9b:1d:a4:c3:6f:1c:dc:7f:63:b2:
    5d:33:b4:4a:0e:5c:05:c9:46:91:c7:ad:1c:31:b9:
    6a:83:f0:3d:29:09:f5:f9:6d:a5:6f:50:7c:d4:7a:
    51:28:d3:16:c0:fe:35:a7:2a:41:6d:a5:54:5d:72:
    04:4c:2a:af:f1
exponent2:
    00:ab:37:50:f4:2f:01:21:d1:1d:5e:e5:51:20:52:
    96:37:a9:02:e9:99:4f:bd:2b:e8:65:5a:11:73:67:
    26:b8:b4:ae:12:bb:01:e8:82:bc:0b:b4:1b:a2:a4:
    4c:97:b0:94:74:09:0e:fe:66:39:90:fb:5b:c6:5f:
    86:ed:1c:8d:01
coefficient:
    03:35:72:37:6f:9f:fb:4e:c6:75:8f:bc:a3:bf:56:
    87:8e:86:43:c5:72:2d:f7:62:1e:02:ae:a4:f7:7d:
    3b:4a:53:fe:13:64:d2:e7:2e:af:bb:55:5b:57:28:
    f1:52:fb:83:00:d8:f5:ea:9b:49:01:53:46:d7:27:
    74:77:59:d9
```

La clé privée et la clé publique sont maintenant implicitement présentes dans le fichier `cle.key`, puisque le fichier contient les deux paires de nombre {d,n} et {e,n} représentant la clé privée et la clé publique.
On utilise ce fichier pour générer la requête de certificat.

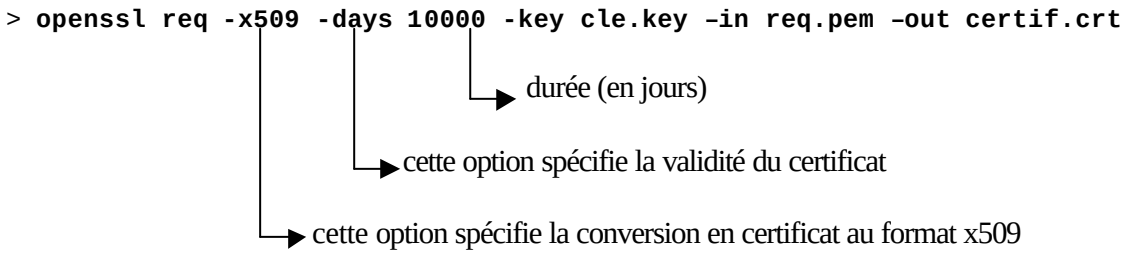


Cette commande se sert du fichier de configuration par défaut de OpenSSL qui défini les différents champs d'informations du certificat.
L'utilisateur répond à une série de questions (entrer un **.** pour laisser un champ vierge) de type:

- Country Name (2 letter code): CH
- State or Province Name : GVE
- Locality Name: GENEVE
- Organization name: eig
- Organization Unit Name: labotdd
- Common name: vectra1.td.unige.ch
- Email Address: d-cotte@wanadoo.fr
- A challenge password:.
- An optional company name:.

On possède maintenant une requête de certificat dans le fichier `req.pem` au format PKCS#10 que l'on développera plus tard.

On peut maintenant valider la requête pour générer un certificat numérique en entrant:



Notre certificat est disponible c'est le fichier `certif.crt`.

3.6 Conclusion

Un certificat, est un document électronique, qui contient divers renseignements sur une personne ainsi que la clé publique de celle ci. Ces informations sont signées à l'aide de la clé privée d'une autorité de certification. La clé publique de l'autorité est supposée connue de tous les correspondants, mais si ce n'est pas le cas, cette autorité peut disposer elle-même d'un certificat signé par une autorité de plus haut niveau. Un tel certificat indiquera de façon certaine la clé publique de la personne au nom et prénom spécifiés. Donc si on reçoit un message dont la signature est déchiffrable à l'aide de cette clé publique, c'est nécessairement cette personne qui a envoyé le message. Il est donc possible de rendre publique ces certificats dans des annuaires de clés. On voit tout de suite l'importance de la clé secrète de l'autorité de certification. Si elle est compromise, il peut y avoir émission de faux certificats, ce qui peut avoir des conséquences extrêmement graves. Pour éviter au maximum ce problème, la clé secrète n'est connue de personne, elle est seulement contenue dans une machine appelée CSU (*Certificates Signing Unit*).

L'outil OpenSSL possède toutes les fonctions d'une autorité de certification, c'est en plus un formidable outil de cryptographie. Outre le fait d'effectuer du cryptage, des conversions de format, signer des requêtes de certificat, nous l'avons utilisé pour créer un certificat numérique. Cet outil permet de visualiser les différentes informations comprises dans les fichiers au format PEM et DER.

3.7 Références

Voici la liste des sites qui m'ont aidé tout au long de ce chapitre. Ils sont classés par ordre d'importance :

http://cui.unige.ch/eao/www/memoires/Hugentobler/crypto.html

- Excellent site sur la crytographie, SSL, les certificats, le tout illustré avec OpenSSL. (clair et très utile)

http://www.commentcamarche.com/crypto/pki.php3

- Théorie d'une architecture à clé publique (très clair, bonne vulgarisation)

http://www.openssl.org/docs/

- Historique d'OpenSSL.
- Téléchargement des dernières versions.
- Documentation sur les commandes OpenSSL et sur la syntaxe à utiliser.

http://www.linux-sottises.net/apache_install.php

- Installation d'Apache avec SSL. (simple et clair, pas trop de détail)

http://enligne.infonet.fundp.ac.be/coursenligne/cours/00-01/IHDC2211/IHDC2211-5bis.big/sld001.htm

- Théorie sur les PKI. (9 illustrations : schémas, blocs, dessins)

http://www.cuefa.inpg.fr/%7Eplisson/reseaux/exposes/chiffrement/chiffrement.htm

- Théorie sur le Chiffrement et la cryptographie. (orienté historique)

http://www.pourlascience.com/numeros/pls-260/internet/crypto/encadre2.htm

- Scénario d'une demande certification. (assortie d'un dessin explicite)

http://www.finances.gouv.fr/DGI/tva/telepro/securisation.htm#5

- Question/Réponse sur la sécurité, chiffrement, certificat…(toujours utile)

http://slwww.epfl.ch/SIC/SL/Securite/CA/

- Explication utile pour créer soit même une CA avec des logiciels gratuits.

http://www.mail-archive.com/openssl-users@openssl.org/msg18486.html

- Comment extraire la clé publique et la clé privée d'un fichier « .key » d'OpenSSL. (histoire de bien comprendre que l'outil OpenSSL génère en faite une paire de clé dans le fichier .key)

http://asn1.elibel.tm.fr/fr/introduction/index.htm

- Définition du format ASN1 (exemple explicite)

http://www.kbcafe.com/articles/base64.html

- Définition du format PEM (exemple explicite)

4 Protocoles de Gestion des PKI

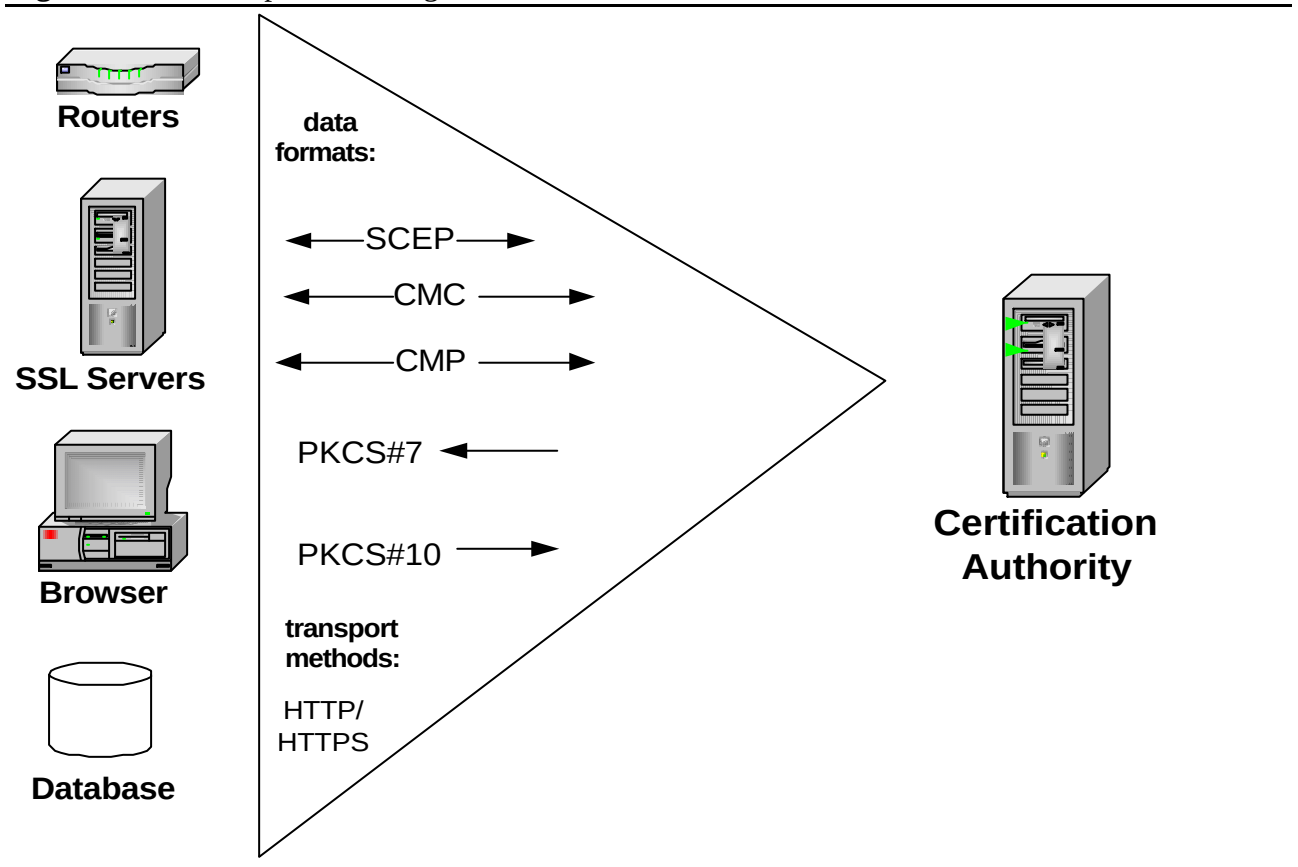
4.1 Fonctionnalités

La technologie PKI, intègre des outils comme les *e-mail*, les routeurs, les *firewalls*, il y a un besoin toujours croissant d'une interface standard pour les services à clé publique et les produits de certification afin de sécuriser les communications. De plus cela favorise l'interopérabilité possible entre différentes PKI.

Des protocoles de gestion *PKI Management Protocols* sont nécessaires pour mettre en œuvre les interactions en ligne entre les participants d'une PKI. Les autorités de certification les utilisent pour collecter les informations nécessaires à la publication de certificats et de CRLs. Il existe plusieurs protocoles différents qui effectuent ce travail avec plus ou moins de complexité et de sécurité.

Cinq protocoles sont couramment utilisés pour implémenter des transactions PKI. PKCS#10 pour *Public Key Cryptography Standard 10* est le plus répandu en combinaison avec SSL et PKCS#7. Le troisième protocole est le « *Certificate Management Protocol* » (CMP), puis vient le « *Certificate Management Using CMS* » (CMC), et enfin, le « *Simple Certificate Enrollment Protocol* » (SCEP).

Figure 4.1 Format et protocoles de gestion des PKI



4.2 Format PKCS#10 [RFC 2986 - 14pages]

Le format PKCS#10 défini la syntaxe d'un simple **message de requête**, et non un protocole complet. Cette requête est une demande de certificat. Le format et le contenu de la réponse à cette requête ne sont pas définis dans PKCS#10. C'est pourquoi, la plupart des implémentations utilisant PKCS#10 pour une requête, emploient l'implémentation PKCS#7 pour retourner le certificat.

Le format PKCS#10 doit toujours être utilisé avec un autre format de message (par exemple PKCS#7) ou un protocole différent pour fournir une fonctionnalité complète de *PKI Management Protocol*.

4.2.1 Définition

La spécification PKCS10 décrit la syntaxe des **demandes de certificat**. Une requête de certification consiste en :

- un nom distinctif
- une clé publique
- éventuellement une série d'attributs signés collectivement par l'entité qui demande le certificat

Les attributs optionnels sont utilisés pour transporter des informations à inclure dans le certificat et pour fournir des renseignements supplémentaires à l'autorité de certification (ex : une adresse postale). Une partie de ces attributs est donnée dans PKCS#9.

Les demandes de certificats sont envoyées à une autorité de certification qui transforme les requêtes en certificats à clé publique X 509.

Syntaxe des demandes de certification.

Une demande de certification se compose de trois parties:

- l'information de demande de certification
- l'algorithme de signature,
- une signature numérique sur l'information de demande de certification

L'information de demande de certification comprend le nom distingué de l'entité, la clef publique de l'entité, et un ensemble d'attributs fournissant d'autres informations au sujet de l'entité.

Une demande de certification est construite en trois étapes :

1. Une valeur « CertificationRequestInfo » contenant un nom distingué, une clef publique, et en option un ensemble d'attributs est construite par une entité.
2. La valeur de « CertificationRequestInfo » est signée avec la clef privée cette même entité.
3. La valeur de « CertificationRequestInfo », l'algorithme de signature, et la signature de l'entité sont rassemblés dans une valeur « CertificationRequest ». [Extrait de RFC 2986]

Décomposons le type « **CertificationRequestInfo** » utilisé dans les deux premières étapes au format ASN1 :

```
CertificationRequestInfo ::= SEQUENCE {
     version Version,
     subject Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     attributes [0] IMPLICIT Attributes
}
```

Les champs du type « **CertificationRequestInfo** » ont les significations suivantes:

_**Version** est le numéro de la version, pour la compatibilité avec de futures révisions des documents. Elle sera 0 pour cette version du document. (en ASN1: « Version ::= INTEGER »)

_**subject** Name, est le nom distingué du sujet du certificat (l'entité dont la clef publique doit être certifiée).

_**subjectPublicKeyInfo** contient des informations sur la clef publique qui sera certifiée. L'information identifie l'algorithme utilisé pour la clef publique de l'entité (et les paramètres associés); un exemple d'algorithme de clef publique inclue le rsa de X.509 et le rsaEncryption de PKCS #1. L'information inclut également une représentation « bit-string » de la clef publique de l'entité.

_**attributes** est un ensemble d'attributs fournissant des informations additionnelles sur le sujet du certificat. Quelques types d'attributs qui pourraient être utiles sont définis dans PKCS#9. Un exemple est l'attribut « challenge-password », qui indique un mot de passe avec lequel l'entité peut demander la révocation du certificat. Un autre exemple est l'attribut « extended-certificate-attribute », pour définir des attributs d'un certificat dit étendu ou prolongé au format PKCS#6.
(en ASN1 : « Attributes ::= SET OF Attribute »)

Décomposons maintenant de la même manière le type « **`CertificationRequest`** » créé dans la dernière étape:

```
CertificationRequest ::= SEQUENCE {
        certificationRequestInfo CertificationRequestInfo,
        signatureAlgorithm SignatureAlgorithmIdentifier,
        signature Signature }
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
```

Signature ::= BIT STRING

Les champs du type « **`CertificationRequest`** » ont les significations suivantes:

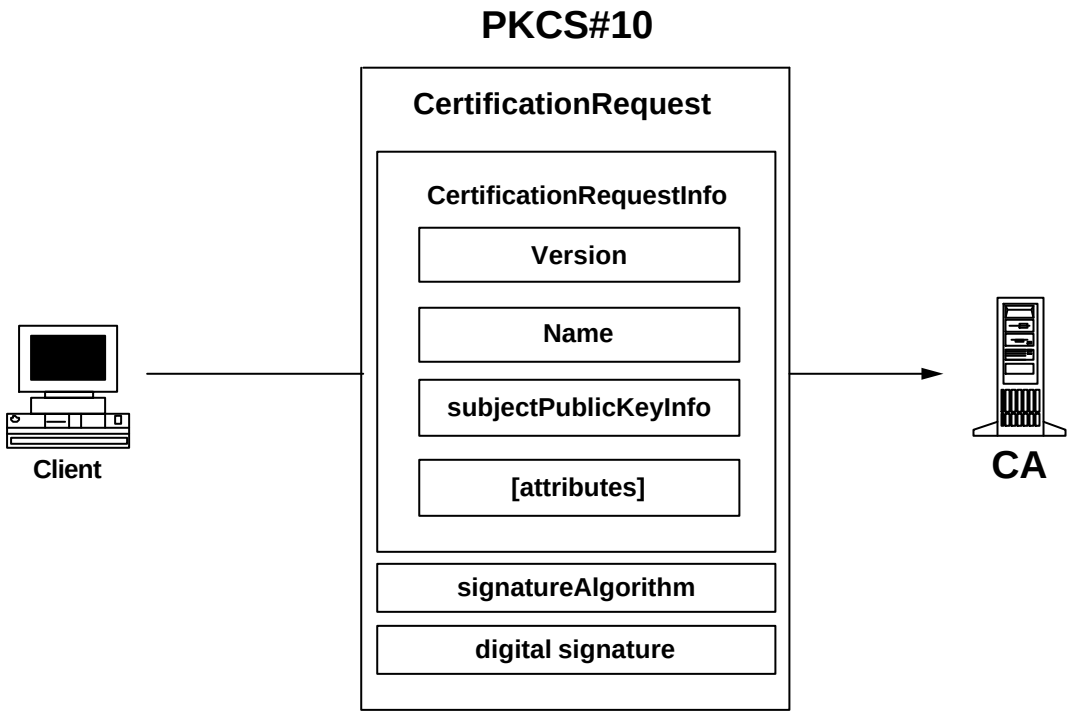
`_certificateRequestInfo` est «l'information de demande de certification vue précédemment». Cette valeur est signée.

`_signatureAlgorithm` identifie l'algorithme de signature (et tous paramètres associés) sous lesquels l'information de demande de certification est signée. Les exemples incluent `md2WithRSAEncryption` et `md5WithRSAEncryption` du format `PKCS #1.`

`_signature` est le résultat de la signature de l'information de demande de certification avec la clef privée du sujet qui demande le certificat.

La Figure 4.2 permet de mieux visualiser ces différents champs.

Figure 4.2 Représentation d'une requête de certification au format PKCS#10



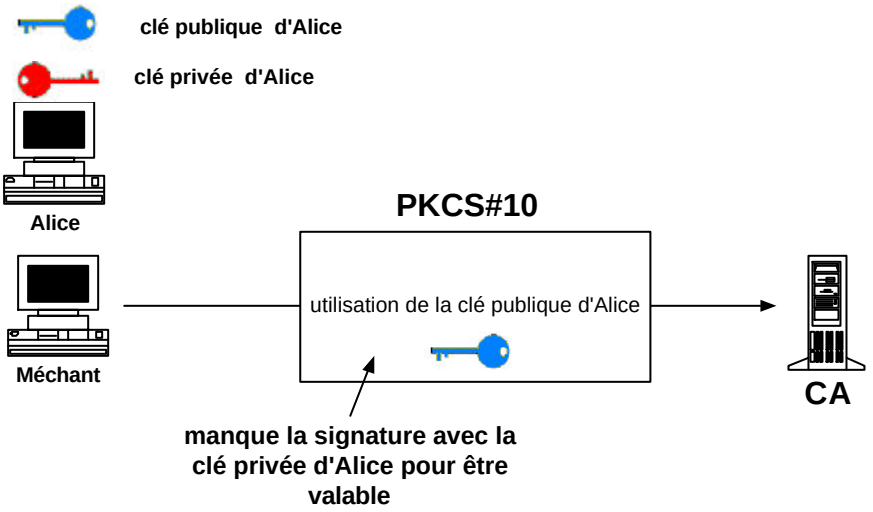
Une fois la demande effectuée, l'autorité de certification traite cette requête en effectuant les actions suivantes :

- vérification de la signature de l'entité,
- si la signature est valide, elle construit un certificat X.509 avec le nom, la clef publique et en ajoutant un nom d'émetteur, un numéro de série, une période de validité, et un algorithme de signature du choix de l'autorité de certification.
- si la demande de certificat contient un ou plusieurs attributs définis dans PKCS #9, elle construit également un certificat étendu PKCS#6 à partir du certificat X.509.
- elle renvoie le nouveau certificat. Celui-ci est souvent un message cryptographique de type PKCS #7.

Remarque sur la Sécurité :

- Une entité envoie typiquement une demande de certification après avoir produit une paire de clé : public-key/private-key, cependant si elle a besoin de changer de nom distingué elle doit à nouveau effectuer cette requête afin d'obtenir un nouveau certificat.
- La signature de la demande de certification est importante. Comme les clés publiques sont accessibles via le certificat publié par la CA une entité peut demander un certificat avec la clef publique d'une autre partie. La signature empêche cette attaque. Ce cas est illustré Figure 4.3. Une telle attaque donnerait à l'entité la capacité de feindre d'être à l'origine de n'importe quel message signé par l'autre partie. Cette attaque est significative seulement si l'entité ne sait pas que le message est signé, et la partie signée du message n'identifie pas le signataire. L'entité ne pourrait néanmoins pas déchiffrer des messages destinés à l'autre partie car il lui faudrait la clé privée.

Figure 4.3 Attaque possible en utilisant la clé publique d'Alice

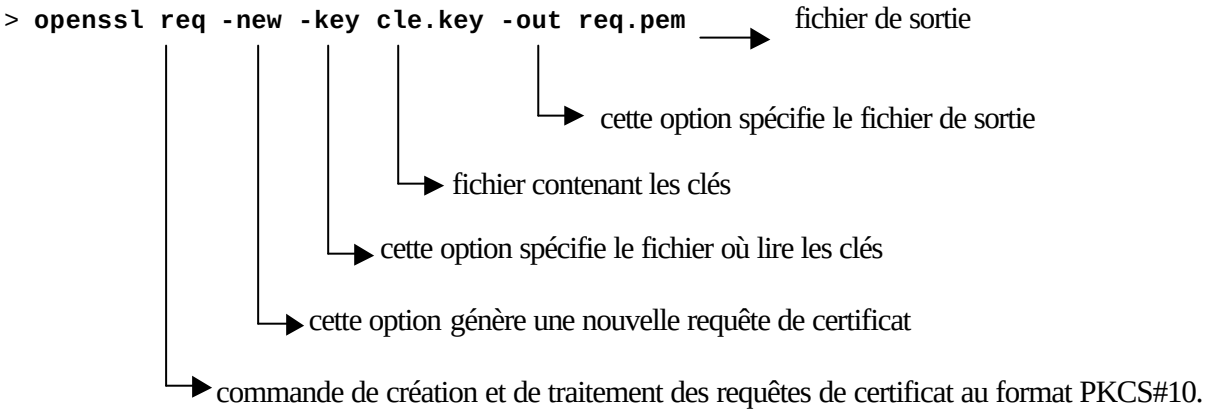


Attaque dans laquelle le méchant utiliserait la clé publique d'une autre partie. (La signature avec la clé privée empêche cette attaque)

- La signature digitale prouve que l'utilisateur a effectué la requête avec la clé privée correspondante. Cependant, une CA qui reçoit une requête PKCS#10 a besoin d'autres informations pour authentifier l'identité du requérant et vérifier que la requête a été reçue sans altération.
- Une CA qui reçoit une requête au format PKCS#10 sur HTTP ne peut pas authentifier l'identité du demandeur ni savoir si la requête n'a pas été altérée par une attaque de type «*man-in-the-middle* ». Pour cette raison, la requête est transmise via une connexion cryptée de type SSL. Le client et le serveur établissent une clé privée partagée.

4.2.2 Exemple avec OpenSSL

La commande `req` d'OpenSSL crée cette **requête de certificat** au format PKCS#10 :



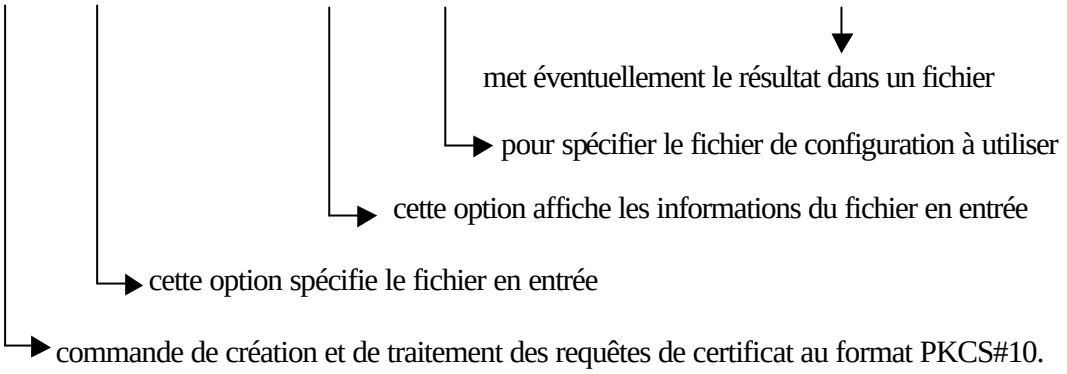
Diverses informations sont nécessaires pour créer cette requête, elles sont demandées à l'utilisateur et configurable dans le fichier de configuration « `openssl.cnf` » (ex : nom, email, organisation, localisation….) Voici l'extrait du fichier de configuration correspondant :

```
[ req_distinguished_name ]

countryName             = Country Name (2 letter code)
countryName_default             = AU
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Some-State

localityName                    = Locality Name (eg, city)

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
```

```
#1.organizationName           = Second Organization Name (eg, company)
#1.organizationName_default   = World Wide Web Pty Ltd

organizationalUnitName        = Organizational Unit Name (eg, section)
#organizationalUnitName_default    =

commonName                    = Common Name (eg, YOUR name)
commonName_max                = 64

emailAddress                  = Email Address
emailAddress_max              = 40

# SET-ex3                     = SET extension number 3

[ req_attributes ]

challengePassword             = A challenge password
challengePassword_min         = 4
challengePassword_max         = 20

unstructuredName              = An optional company name
```

On peut avec OpenSSL demander et vérifier les différentes données contenues dans le fichier « cot2req.pem » généré. Pour cela on effectue la commande suivante :

> **openssl req -in cot2req.pem -text -config openssl.cnf [-out result.txt]**




```
Fichier result.txt:
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=CH, ST=GVA, L=Geneve, O=eig, CN=cot2/Email=d-cotte@wanadoo.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:aa:f0:7a:56:4b:01:89:d3:12:9f:a0:05:70:30:
                    66:46:72:30:9d:ac:44:52:6d:1d:e7:0a:41:a7:2c:
                    52:60:e4:2e:36:1a:6d:77:f7:e5:ca:85:d8:2e:db:
                    fa:3f:c4:7c:83:5e:f2:4f:ae:fc:18:bf:71:64:e7:
                    8c:36:0b:dc:37
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        40:f3:47:7a:90:9d:f6:66:35:3e:0b:2a:22:1f:a4:b3:8b:33:
        1e:d2:aa:11:02:89:70:3a:59:39:0e:87:bf:04:e3:e5:14:fe:
        05:6d:dc:03:f3:ba:65:73:01:2e:20:c8:4c:c6:4f:fc:ed:8a:
        e7:22:ae:96:51:eb:1e:0e:d4:96

-----BEGIN CERTIFICATE REQUEST-----
MIIBJjCB0QIBADBsMQswCQYDVQQGEwJDSDEMMAoGA1UECBMDR1ZBMQ8wDQYDVQQH
EwZHZW5ldmUxDDAKBgNVBAoTA2VpZzENMAsGA1UEAxMEY290MjEhMB8GCSqGSIb3
DQEJARYSZC1jb3R0ZUB3YW5hZG9vLmZyMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJB
AKrwelZLAYnTEp+gBXAwZkZyMJ2sRFJtHecKQacsUmDkLjYabXf35cqF2C7b+j/E
fINe8k+u/Bi/cWTnjDYL3DcCAwEAAaAAMA0GCSqGSIb3DQEBBAUAA0EAQPNHepCd
9mY1PgsqIh+ks4szHtKqEQKJcDpZOQ6HvwTj5RT+BW3cA/O6ZXMBLiDITMZP/O2K
5yKullHrHg7Ulg==
-----END CERTIFICATE REQUEST-----
```

On retrouve les informations, la clé publique, l'algorithme de signature, la signature, et le certificat au format «.pem » crypté.

En changent l'extension « .pem » du fichier encodé base64 en extension « .der », il est possible avec le fichier `cot2req.der` contenant la demande de certificat au format PKCS#10 de retrouver sa structure ASN.1 vue précédemment et défini dans la [RFC 2986]. La commande OpenSSL est alors la suivante :

Conversion du fichier « .pem » en « .der » :
> **`openssl req –inform PEM –in cot2req.pem –outform DER –out cot2req.der –config openssl.cnf`**

Affichage en ASN1:
> **`openssl asn1parse –inform DER -in cot2req.der`**

Résultat :

```
  0:d=0  hl=4 l= 294 cons: SEQUENCE
  4:d=1  hl=3 l= 209 cons: SEQUENCE
  7:d=2  hl=2 l=   1 prim: INTEGER           :00                //Version
 10:d=2  hl=2 l= 108 cons: SEQUENCE
 12:d=3  hl=2 l=  11 cons: SET
 14:d=4  hl=2 l=   9 cons: SEQUENCE
 16:d=5  hl=2 l=   3 prim: OBJECT            :countryName       //DN
 21:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :CH
 25:d=3  hl=2 l=  12 cons: SET
 27:d=4  hl=2 l=  10 cons: SEQUENCE
 29:d=5  hl=2 l=   3 prim: OBJECT            :stateOrProvinceName
 34:d=5  hl=2 l=   3 prim: PRINTABLESTRING   :GVA
 39:d=3  hl=2 l=  15 cons: SET
 41:d=4  hl=2 l=  13 cons: SEQUENCE
 43:d=5  hl=2 l=   3 prim: OBJECT            :localityName
 48:d=5  hl=2 l=   6 prim: PRINTABLESTRING   :Geneve
 56:d=3  hl=2 l=  12 cons: SET
 58:d=4  hl=2 l=  10 cons: SEQUENCE
 60:d=5  hl=2 l=   3 prim: OBJECT            :organizationName
 65:d=5  hl=2 l=   3 prim: PRINTABLESTRING   :eig
 70:d=3  hl=2 l=  13 cons: SET
 72:d=4  hl=2 l=  11 cons: SEQUENCE
 74:d=5  hl=2 l=   3 prim: OBJECT            :commonName
 79:d=5  hl=2 l=   4 prim: PRINTABLESTRING   :cot2
 85:d=3  hl=2 l=  33 cons: SET
 87:d=4  hl=2 l=  31 cons: SEQUENCE
 89:d=5  hl=2 l=   9 prim: OBJECT            :emailAddress
100:d=5  hl=2 l=  18 prim: IA5STRING         :d-cotte@wanadoo.fr
120:d=2  hl=2 l=  92 cons: SEQUENCE
122:d=3  hl=2 l=  13 cons: SEQUENCE
124:d=4  hl=2 l=   9 prim: OBJECT    :rsaEncryption         //SubjectPublicKeyInfo
135:d=4  hl=2 l=   0 prim: NULL
137:d=3  hl=2 l=  75 prim: BIT STRING                       //Public Key
214:d=2  hl=2 l=   0 cons: cont [ 0 ]
216:d=1  hl=2 l=  13 cons: SEQUENCE
218:d=2  hl=2 l=   9 prim: OBJECT    :md5WithRSAEncryption  //SignatureAlgorithmId
229:d=2  hl=2 l=   0 prim: NULL
231:d=1  hl=2 l=  65 prim: BIT STRING                       //Signature
```

Le premier nombre représente l'offset dans le fichier source «`cot2req.der` ». La valeur «`d` » désigne la profondeur dans la normalisation ASN, on peut voir cette valeur comme le degré d'imbrication dans les structures ASN. La valeur « `hl` » représente la longueur de l'entête de chaque commande en octet, alors que « `l` » est la longueur totale du contenu. Il est possible d'aller encore plus loin dans cette étude avec l'option « `-strparse` offset» en lui donnant l'*offset* de la ligne à développer. Par exemple, pour développer le `BIT STRING` de l'*offset* 137 qui représente la clé publique, on utilise la commande suivante :

> **`openssl asn1parse –inform DER –in cot2req.der –strparse 137`**

Résultat :

```
  0:d=0  hl=2 l=  72 cons: SEQUENCE
  2:d=1  hl=2 l=  65 prim: INTEGER :AAF07A564B0189D3129FA005703066467230 9DAC44526D
1DE70A41A72C5260E42E361A6D77F7E5CA85D82EDBFA3FC47C835EF24FAEFC18BF7164E78C360BDC37
  69:d=1  hl=2 l=   3 prim: INTEGER           :010001
```

On peut voir que la valeur de la clé publique est correcte par rapport à celle précédemment trouvée en vérifiant la requête de certificat.

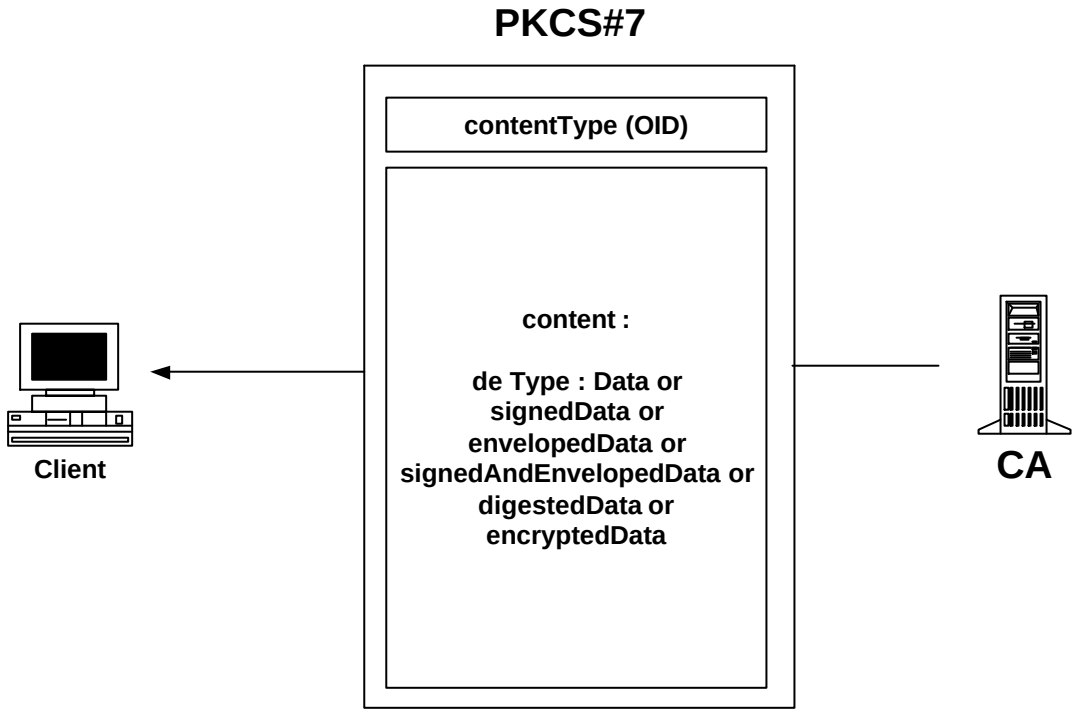
4.3 PKCS#7 [RFC 2315 -32 pages]

PKCS#7 est une spécification standard pour la protection de messages à l'aide de la cryptographie. Ce format est la base de la sécurité employée dans de nombreux protocoles, y compris S/MIME v2. Un message PKCS#7 de base possède seulement deux champs: un type et le contenu lui-même.

La syntaxe au format ASN.1 du contenu échangé entre les deux entités est le suivant :

```
ContentInfo ::= SEQUENCE {
        contentType ContentType,
        content
           [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL
}

ContentType ::= OBJECT IDENTIFIER
```

Figure 4.4 Réponse au format PKCS#7



Les champs du type « **`ContentInfo`** » ont les significations suivantes:

_ **`contentType`** indique le type de contenu. Il s'agit d'un identificateur d'objet (OID), une suite unique de nombres entiers assignés par l'autorité qui définie le type. Il y a six types de contenu défini: `Data`, `signedData`, `envelopedData`, `signedAndEnvelopedData`, `digestedData`, et `encryptedData`.

_ **`content`** représente le contenu, ce champ est facultatif. Son type est défini avec un identificateur d'objet dans `contentType`.

Les différents types de contenu sont définis par la [RFC 2315]. On va développer l'un d'entre eux, par exemple le type « **`signedData`** ».

Le type de contenu « **`signedData`** » se compose de plusieurs champs dont un condensé du message chiffré. Le message condensé chiffré est "une signature numérique" sur le contenu. N'importe quel type de contenu peut être signé par des signataires. En outre, la syntaxe possède un cas dans lequel il n'y a aucun signataire sur le contenu. Ce cas fournit un moyen de disséminer des certificats et des listes de certificats révoqués.

Le processus par lequel des données sont signées implique les étapes suivantes:

1. pour chaque signataire, un condensé du message est calculé sur le contenu avec un algorithme spécifique. Si le signataire authentifie n'importe quelle information autre que le contenu, le condensé du message sur le contenu et l'autre information sont condensé avec l'algorithme de condensé du message du signataire, et le résultat devient le "*message digest*"
2. pour chaque signataire, le condensé du message et les informations associées sont chiffrés avec la clé privée du signataire.
3. pour chaque signataire, le condensé du message chiffré et les informations du signataire sont rassemblés dans une valeur « `SignerInfo` ». Les certificats, la liste de certificats révoqués pour chaque signataire, et ceux qui ne correspondent pas à leur signataire, sont collectés à ce moment.
4. Les algorithmes de condensé de message pour chaque signataire et les valeurs de « `SignerInfo` » sont rassemblés ainsi que le contenu dans une valeur de « `SignedData` »

Un destinataire vérifie les signatures en déchiffrant le condensé du message chiffré avec la clé publique du signataire, comparant alors le condensé du message récupéré avec celui calculé indépendamment à partir du message. La clé publique du signataire est contenue dans un certificat lui même inclut dans l'information du signataire. Elle est mise en référence avec un nom distingué par l'émetteur et un numéro de série spécifique qui identifient le certificat de façon unique.

Le type de contenu « **SignedData** » possède la composition ASN.1 suivante:

```
SignedData ::= SEQUENCE {
     version Version,
     digestAlgorithms DigestAlgorithmIdentifiers,
     contentInfo ContentInfo,
     certificates
        [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
     crls
        [1] IMPLICIT CertificateRevocationLists OPTIONAL,
     signerInfos SignerInfos
}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

Les champs ont les significations suivantes: [Extrait de RFC2315]

- version is the syntax version number. It shall be 1 for this version of the document.

- digestAlgorithms is a collection of message-digest algorithm identifiers. There may be any number of elements in the collection, including zero. Each element identifies the message-digest algorithm (and any associated parameters) under which the content is digested for a some signer. The collection is intended to list the message-digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification. The message-digesting process is described in Section 9.3.

- contentInfo is the content that is signed. It can have any of the defined content types.

- certificates is a set of PKCS #6 extended certificates and X.509 certificates. It is intended that the set be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the signers in the signerInfos field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level certification authorities. There may also be fewer certificates than necessary, if it is expected that those verifying the signatures have an alternate means of obtaining necessary certificates (e.g., from a previous set of certificates).

- crls is a set of certificate-revocation lists. It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are "hot listed," but such correspondence is not necessary. There may be more

```
        certificate-revocation lists than necessary, and there may
        also be fewer certificate-revocation lists than necessary.

   o    signerInfos is a collection of per-signer
        information. There may be any number of elements in the
        collection, including zero.
```

Les cinq autres types de contenu sont définis dans la [RFC 2315], pour plus d'information s'y reporter.

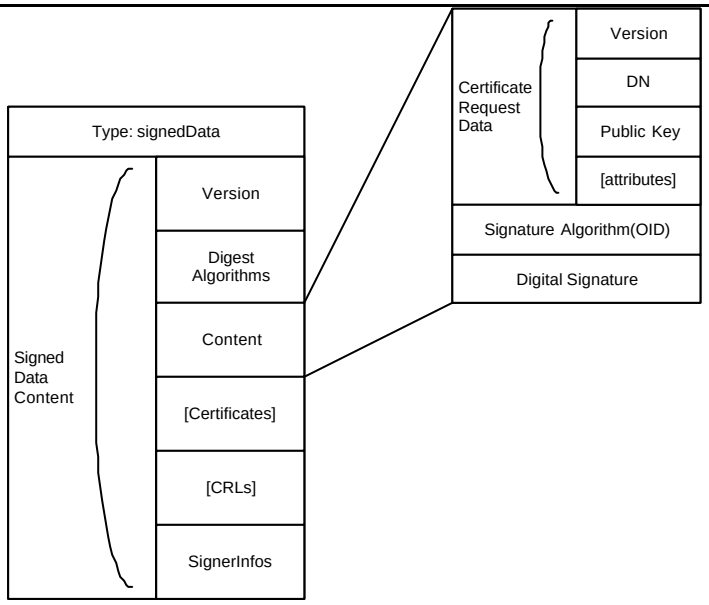
4.4 PKCS#7 et PKCS#10

PKCS#7 et #10 sont des spécifications indépendantes souvent employées ensemble pour mettre en oeuvre un protocole afin de délivrer des certificats. Il n'y a aucune spécification standard décrivant cette combinaison.

Un message signé est représenté Figure 4.5 dont le contenu est une requête de format PKCS#10. Les différents champs définis précédemment en ASN1 sont visibles. Le champ « `SignerInfos` » est composé de six informations:

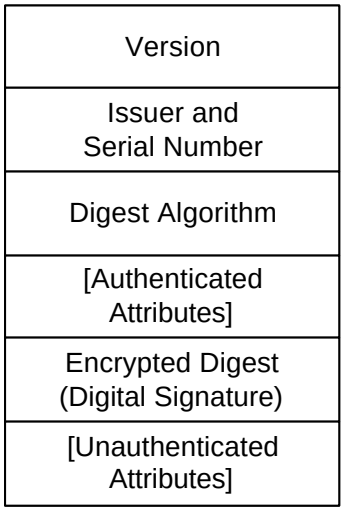
- la version
- l'émetteur suivi du numéro de série du certificat contenant la clef publique demandée afin de vérifier la signature digitale
- des attributs d'authentifications
- l'algorithme de signature digital
- la signature digitale elle-même
- des attributs non authentifiés.

Figure 4.5 Composition d'un message de donnés signées.



Les attributs authentifiés et les attributs non authentifiés sont tous les deux facultatifs. La signature digitale est produite avec le contenu de « **`signedData`** », et les attributs authentifiés du champ « **`signerInfos`** ». De cette façon, toutes les signatures incluent le contenu de «**`signedData`** ». Chaque signataire peut compléter le contenu avec des attributs authentifiés différents. Le champ « **`signerInfos`** » est représenté Figure 4.6

Figure 4.6 Composition du champ « **`signerInfos`** »



Legend :
[ ] Optional Field

- En encapsulant le format PKCS#10 dans le format PKCS#7, le message peut être signé avec une clef privée autre que celle du demandeur de certificat. Cela permet à un RA de signer la demande avec sa clef privée. Le RA est un intermédiaire entre l'utilisateur et la CA.
- De plus, la CA peut utiliser PKCS#7 pour authentifier le certificat retourné à l'abonné. Cela fournit au client ou au RA la preuve archivistique que le certificat a été publié.
- PKCS#7 peut être employé comme une réponse de la CA au demandeur. Si le certificat a été publié, la CA peut retourner le nouveau certificat d'abonné dans un message « `signedData` », comme suggéré dans la Figure 4.5 où le contenu est la requête PKCS#10. Ensemble, les deux messages forment une transaction complète.
- La CA peut conserver le message PKCS#7 signé contenant la demande de certificat sous forme PKCS#10 afin de l'archiver. L'utilisateur peut lui aussi conserver le message de réponse PKCS#7 signé. Chaque participant a un message signé pour effectuer une archive qui fera office de preuve.

La combinaison de PKCS#7 et PKCS#10 représente une amélioration significative par rapport à PKCS#10. Nous avons un protocole qui offre des messages signés qui peuvent être archivés. Une preuve de possession des clefs de signature dans le message PKCS#10 peut être préservée même si un RA a signé le message PKCS#7. Nous avons un protocole simple, avec un flux de message bien défini pour l'édition de certificat. Le protocole est facile à mettre en oeuvre, depuis la plupart des supports système PKCS#7 et des formats PKCS#10.

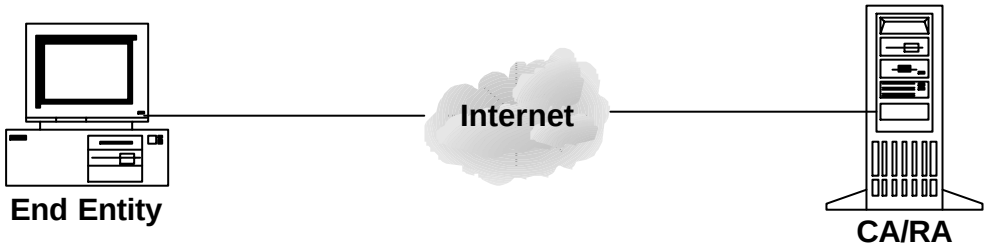
4.5 Certificate Management Protocol (CMP) [RFC 2510 -72pages]

Le groupe de travail IETF PKIX a voulu développer un protocole compréhensible supportant une grande variété de modèles.
Le protocole CMP défini d'une manière générale les différents éléments d'une PKI :

- les différentes entités (utilisateur final, RA, CA…)
- les points nécessaires à la gestion d'une PKI (conformité à ISO 9594-8, mise à jour régulière d'une des pair de clé sans affecter les autres, support de l'utilisation d'algorithme de cryptographie standard …)
- la location de la génération des clés, le premier message de l'entité au CA…
- un schéma d'authentification basic :

```
In terms of the classification above, this scheme is where:
```

- initiation occurs at the end entity;
- message authentication is REQUIRED;
- "key generation" occurs at the end entity
- a confirmation message is REQUIRED.




```
End entity                                                    RA/CA
==========                                                 =============
              out-of-band distribution of Initial Authentication
                 Key (IAK) and reference value (RA/CA -> EE)
Key generation
Creation of certification request
Protect request with IAK
                  -->>--certification request-->>--
                                                      verify request
                                                      process request
                                                      create response
                  --<<--certification response--<<--
handle response
create confirmation
                  -->>--confirmation message-->>--
                                                      verify confirmation
(Where verification of the confirmation message fails, the RA/CA MUST
 revoke the newly issued certificate if it has been published or
 otherwise made available.)
```

- un champ de Preuve de Possession (Proof of Possession POP) de la clé privée :

  Ce champ a pour but d'empêcher certaines attaques et de permettre au CA de vérifier la validité du lien qui existe entre entité et clé privée. En effet, des opérations permettent à une entité de prouver qu'elle est bien en possession de la clé privée correspondante à la clef publique pour laquelle le certificat est demandé. Le CA est libre de choisir le moyen de mettre en œuvre le POP dans ses échanges de certification. Il est clair que l'entité n'enverra jamais sa clé privée directement pour prouver qu'elle la détient. Voici quelques exemples :

  - L'entité peut signer une valeur et l'envoyer pour prouver la possession de la clé privée.
  - L'entité peut décrypter une valeur pour prouver la possession de la clé privée directement ou indirectement.

    `The direct method is for the RA/CA to issue a random challenge to which an immediate response by the EE is required.`

    `The indirect method is to issue a certificate which is encrypted for the end entity (and have the end entity demonstrate its ability to decrypt this certificate in the confirmation message). This allows a CA to issue a certificate in a form which can only be used by the intended end entity.`

    `This specification encourages use of the indirect method because this requires no extra messages to be sent (i.e., the proof can be demonstrated using the {request, response, confirmation} triple of messages). [extrait RFC 2510]`

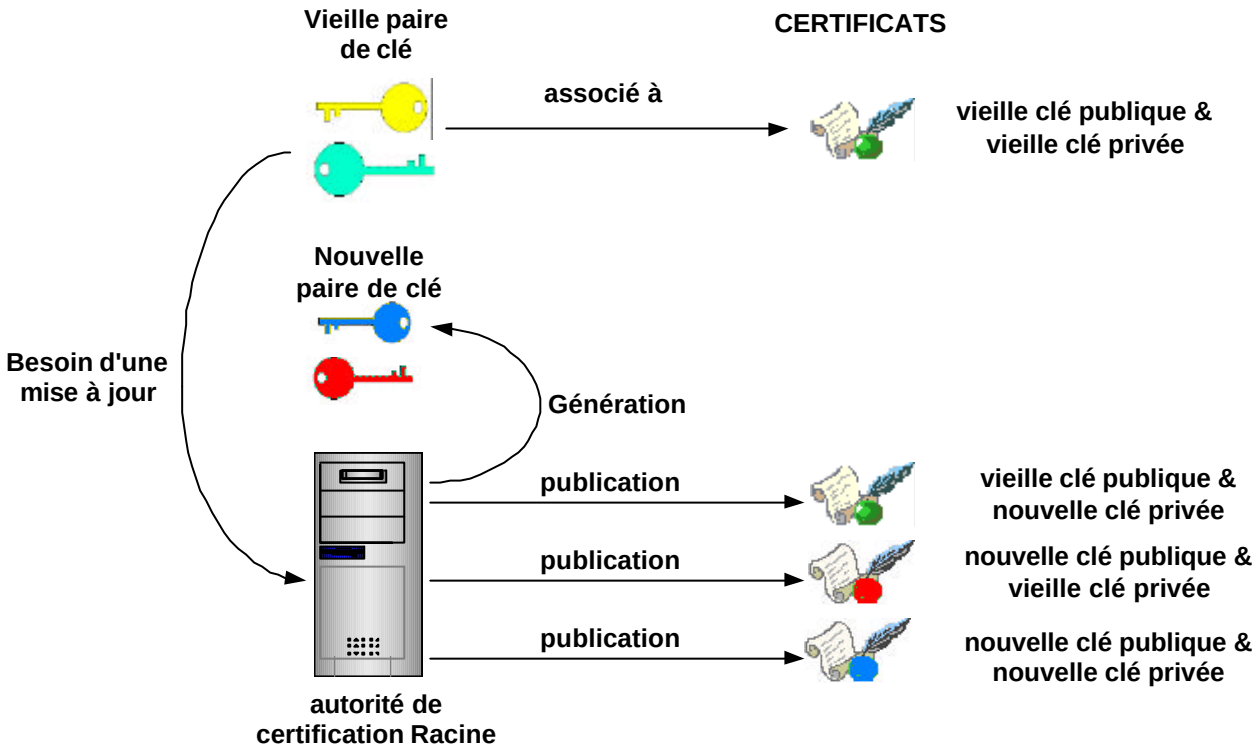
  - L'entité et la CA établissent une clé (secret partagé) pour prouver la possession de la clé privée.

- Principe de la mise à jour des clés d'une « *Root CA* »

  La fonctionnalité suivante n'est valable que pour les CA Racine (*Root CA*).
  Lorsque l'autorité de certification veut changer ses clés, il génère une nouvelle paire de clé.
  Elle combine ses nouvelles et ses anciennes clés pour générer des certificats.
  (for a total of four: *OldWithOld*; *OldWithNew*; *NewWithOld*; and *NewWithNew*)
  Les entités qui ont acquis l'ancienne clé publique de la CA sont affectées par ce changement.
  Ces entités peuvent alors accéder à la nouvelle clé publique de la CA protégée avec la vieille clé publique de la CA. Cette fonctionnalité est valable uniquement pendant une période de temps définie.

La Figure 4.7 illustre ce principe :

Figure 4.7 Mise à jour des clés d'une CA racine (*root CA*)



➢ Action de la CA

Pour changer ses clés la CA effectue les opérations suivantes :

- Génération d'une nouvelle paire de clé
- Création d'un certificat contenant l'ancienne clé publique signé avec la nouvelle clé privé (« *old with new* » *certificate*)
- Création d'un certificat contenant la nouvelle clé publique signé avec l'ancienne clé privée (« *new this old* » *certificate*)
- Création d'un certificat contenant la nouvelle clé publique signé avec la nouvelle clé privée (« *new with new* » *certificate*)
- Publication de ces nouveaux certificats.
- Exportation de la nouvelle clé publique de la CA pour que les entités de fin puissent l'acquérir.

La vieille clé privée de la CA n'est maintenant plus utilisée. La vieille clé publique reste valable le temps que toutes les entités acquièrent la nouvelle clé publique de la CA. Ce mécanisme permet donc de parer le problème d'une clé privée compromise.

La validité des différents certificats est la suivante :

```
      The "old with new" certificate must have a validity period starting at
the generation time of the old key pair and ending at the expiry date of the
old public key.

      The "new with old" certificate must have a validity period starting at
the generation time of the new key pair and ending at the time by which all
end entities of this CA will securely possess the new CA public key (at the
latest, the expiry date of the old public key).

      The "new with new" certificate must have a validity period starting at
the generation time of the new key pair and ending at the time by which the
CA will next update its key pair.
```

Le message au format *Certificat Management Protocol* (CMP) est représenté Figure 4.8. Il est composé de quatre parties, « l'entête », « le corps », « la protection », et la partie « extra certificat ». CMP défini un message utilisé dans une structure PKI comme ayant la structure ASN1 suivante :

```
PKIMessage ::= SEQUENCE {
        header           PKIHeader,
        body             PKIBody,
        protection   [0] PKIProtection OPTIONAL,
        extraCerts   [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL
}
```

La partie « extraCerts » et la partie « protection » sont optionnelles. Le champ protection est utilisé pour garantir l'intégrité de l'entête et du corps. Il contient une signature digitale, un code d'authentification ou le condensé de ce code. Le champ protection n'est pas utilisé si l'entête et le corps du message sont déjà protégés par d'autres moyens. L'entête contient le nom de l'expéditeur, celui du destinataire, un « Message Time » et l'algorithme de cryptage utilisé pour protéger le message.

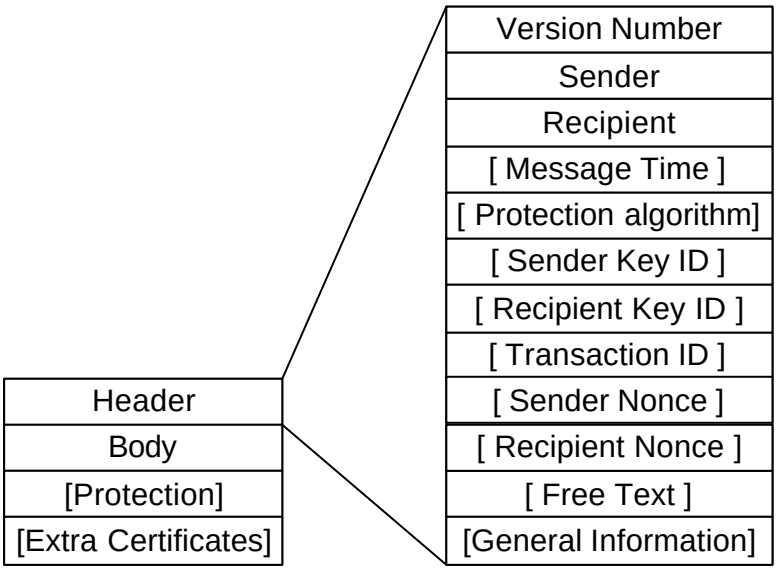
Tous les messages PKI exigent des informations placées en entête pour identifier la transaction et effectuer l'adressage. La structure de données suivante est employée pour contenir ces informations :

```
PKIHeader ::= SEQUENCE {
        pvno                INTEGER     { ietf-version2 (1) },
        sender              GeneralName,
        -- identifies the sender
        recipient           GeneralName,
        -- identifies the intended recipient
        messageTime     [0] GeneralizedTime         OPTIONAL,
        -- time of production of this message (used when sender
        -- believes that the transport will be "suitable"; i.e.,
```

```
        -- that the time will still be meaningful upon receipt)
        protectionAlg   [1] AlgorithmIdentifier     OPTIONAL,
        -- algorithm used for calculation of protection bits
        senderKID       [2] KeyIdentifier           OPTIONAL,
        recipKID        [3] KeyIdentifier           OPTIONAL,
        -- to identify specific keys used for protection
        transactionID   [4] OCTET STRING            OPTIONAL,
        -- identifies the transaction; i.e., this will be the same in
        -- corresponding request, response and confirmation messages
        senderNonce     [5] OCTET STRING            OPTIONAL,
        recipNonce      [6] OCTET STRING            OPTIONAL,
        -- nonces used to provide replay protection, senderNonce

        -- is inserted by the creator of this message; recipNonce
        -- is a nonce previously inserted in a related message by
        -- the intended recipient of this message
        freeText        [7] PKIFreeText             OPTIONAL,
        -- this may be used to indicate context-specific instructions
        -- (this field is intended for human consumption)
        generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
                                InfoTypeAndValue     OPTIONAL
        -- this may be used to convey context-specific information
        -- (this field not primarily intended for human consumption)
    }
```

Cette représentation est reprise Figure 4.8

Figure 4.8 Message au format CMP



Les champs de l'entête ont les significations suivantes : [extrait RFC2510]

The pvno field is fixed (at one) for this version of this specification.

The sender field contains the name of the sender of the PKIMessage. This name (in conjunction with senderKID, if supplied) should be usable to verify the protection on the message. If nothing about the sender is known to the sending entity (e.g., in the init. req. message, where the end entity may not know its own Distinguished Name (DN), e-mail name, IP address, etc.), then the "sender" field MUST contain a "NULL" value; that is, the SEQUENCE OF relative distinguished names is of zero length. In such a case the senderKID field MUST hold an identifier (i.e., a reference number) which indicates to the receiver the appropriate shared secret information to use to verify the message.

The recipient field contains the name of the recipient of the PKIMessage. This name (in conjunction with recipKID, if supplied) should be usable to verify the protection on the message.

The protectionAlg field specifies the algorithm used to protect the message. If no protection bits are supplied (note that PKIProtection is OPTIONAL) then this field MUST be omitted; if protection bits are supplied then this field MUST be supplied.

senderKID and recipKID are usable to indicate which keys have been used to protect the message (recipKID will normally only be required where protection of the message uses Diffie-Hellman (DH) keys).

The transactionID field within the message header MAY be used to allow the recipient of a response message to correlate this with a previously issued request. For example, in the case of an RA there may be many requests "outstanding" at a given moment.

The senderNonce and recipNonce fields protect the PKIMessage against replay attacks.

The messageTime field contains the time at which the sender created the message. This may be useful to allow end entities to correct their local time to be consistent with the time on a central system.

The freeText field may be used to send a human-readable message to the recipient (in any number of languages). The first language used in this sequence indicates the desired language for replies.

The generalInfo field may be used to send machine-processable additional data to the recipient.

Ce que contient le corps est déterminé par le type du message. CMP défini 24 types de message. Au format ASN.1 ces messages sont :

```
PKIBody ::= CHOICE {       -- message-specific body elements
       ir      [0]  CertReqMessages,        --Initialization Request
       ip      [1]  CertRepMessage,         --Initialization Response
       cr      [2]  CertReqMessages,        --Certification Request
       cp      [3]  CertRepMessage,         --Certification Response
       p10cr   [4]  CertificationRequest,   --PKCS #10 Cert. Req.
         -- the PKCS #10 certification request (see [PKCS10])
       popdecc [5]  POPODecKeyChallContent, --pop Challenge
       popdecr [6]  POPODecKeyRespContent,  --pop Response
       kur     [7]  CertReqMessages,        --Key Update Request
       kup     [8]  CertRepMessage,         --Key Update Response
       krr     [9]  CertReqMessages,        --Key Recovery Request
       krp     [10] KeyRecRepContent,       --Key Recovery Response
       rr      [11] RevReqContent,          --Revocation Request
       rp      [12] RevRepContent,          --Revocation Response
       ccr     [13] CertReqMessages,        --Cross-Cert. Request
       ccp     [14] CertRepMessage,         --Cross-Cert. Response
       ckuann  [15] CAKeyUpdAnnContent,     --CA Key Update Ann.
       cann    [16] CertAnnContent,         --Certificate Ann.
       rann    [17] RevAnnContent,          --Revocation Ann.
       crlann  [18] CRLAnnContent,          --CRL Announcement
       conf    [19] PKIConfirmContent,      --Confirmation
       nested  [20] NestedMessageContent,   --Nested Message
       genm    [21] GenMsgContent,          --General Message
       genp    [22] GenRepContent,          --General Response
       error   [23] ErrorMsgContent         --Error Message
   }
```

Les deux messages qui nous intéressent le plus sont :

```
       cr      [2]  CertReqMessages,        --Certification Request
       cp      [3]  CertRepMessage,         --Certification Response
       p10cr   [4]  CertificationRequest,   --PKCS #10 Cert. Req.
         -- the PKCS #10 certification request (see [PKCS10])
```

3.3.3. Registration/Certification Request

```
A Registration/Certification request message contains as the PKIBody
a CertReqMessages data structure which specifies the requested
certificates.  This message is intended to be used for existing PKI
entities who wish to obtain additional certificates.

See [CRMF] for CertReqMessages syntax.

Alternatively, the PKIBody MAY be a CertificationRequest (this
structure is fully specified by the ASN.1 structure
CertificationRequest given in [PKCS10]).  This structure may be
required for certificate requests for signing key pairs when
interoperation with legacy systems is desired, but its use is
strongly discouraged whenever not absolutely necessary.
```

3.3.4. Registration/Certification Response

```
A registration response message contains as the PKIBody a
CertRepMessage data structure which has a status value for each
```

```
   certificate requested, and optionally has a CA public key, failure
   information, a subject certificate, and an encrypted private key.

  CertRepMessage ::= SEQUENCE {
      caPubs          [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL,
      response            SEQUENCE OF CertResponse
  }

  CertResponse ::= SEQUENCE {
      certReqId           INTEGER,
      -- to match this response with corresponding request (a value
      -- of -1 is to be used if certReqId is not specified in the
      -- corresponding request)
      status              PKIStatusInfo,
      certifiedKeyPair    CertifiedKeyPair    OPTIONAL,
      rspInfo             OCTET STRING        OPTIONAL
      -- analogous to the id-regInfo-asciiPairs OCTET STRING defined
      -- for regInfo in CertReqMsg [CRMF]
  }

  CertifiedKeyPair ::= SEQUENCE {
      certOrEncCert       CertOrEncCert,
      privateKey      [0] EncryptedValue      OPTIONAL,
      publicationInfo [1] PKIPublicationInfo  OPTIONAL
  }

  CertOrEncCert ::= CHOICE {
      certificate     [0] Certificate,
      encryptedCert   [1] EncryptedValue
  }

   Only one of the failInfo (in PKIStatusInfo) and certificate (in
   CertifiedKeyPair) fields can be present in each CertResponse
   (depending on the status). For some status values (e.g., waiting)
   neither of the optional fields will be present.

   Given an EncryptedCert and the relevant decryption key the
   certificate may be obtained. The purpose of this is to allow a CA to
   return the value of a certificate, but with the constraint that only
   the intended recipient can obtain the actual certificate. The benefit
   of this approach is that a CA may reply with a certificate even in
   the absence of a proof that the requester is the end entity which can
   use the relevant private key (note that the proof is not obtained
   until the PKIConfirm message is received by the CA). Thus the CA will
   not have to revoke that certificate in the event that something goes
   wrong with the proof of possession.  [extrait de la RFC 2510]
```

Une description détaillée de chaque type de message est développée chapitre 3.3 de la [RFC 2510].

4.6 Certificate Management Using CMS (CMC) [RFC 2797 - 47pages]

CMC est beaucoup plus complexe que PKCS#7 et PKCS#10, mais CMC est moins compliqué que CMP. Il a été défini comme échappatoire à la complexité du protocole CMP qui comporte un nombre important de messages.

CMC permet l'utilisation d'un message PKCS#10 non protégé et permet à un message sans contenu de transmettre des certificats. On appel ceci un message «*cert-only* ». La spécification CMC emploie des données signées et enveloppées CMS pour fournir une protection. La plupart des messages CMC ne requièrent pas de confidentialité. En effet le message est construit par encapsulation de message « `PKI data` » et « `PKI response` ».

CMC ne spécifie pas entièrement la transaction de révocation. Il trouve un terrain d'entente dans la complexité. Il y a peu de messages, mais un nombre substantiel d'attributs de contrôle. Les transactions définies peuvent être effectuées en un seul aller-retour, ce qui simplifie sa mise en œuvre.

La spécification CMC défini deux nouveaux types de contenu : « `PKI Data` », et « `PKI response` ». Le type « `PKI Data` » est un message de requête, et «`PKI response` » est la réponse provenant de la CA.

La structure ASN.1 du contenu `PKIData` est la suivante :

```
PKIData ::= SEQUENCE {
        controlSequence    SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
        reqSequence        SEQUENCE SIZE(0..MAX) OF TaggedRequest,
        cmsSequence        SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
        otherMsgSequence   SEQUENCE SIZE(0..MAX) OF OtherMsg
}
```

La signification des différents champs :

```
-- controlSequence consists of a sequence of control attributes.  The
control attributes defined in this document are found in section 5.
As control sequences are defined by OIDs, other parties can define
additional control attributes. Unrecognized OIDs MUST result in no
part of the request being successfully processed.

-- reqSequence consists of a sequence of certificate requests.  The
certificate requests can be either a CertificateRequest (PKCS10
request) or a CertReqMsg.  Details on each of these request types are
found in sections 3.3.1 and 3.3.2 respectively.

-- cmsSequence consists of a sequence of [CMS] message objects.  This
protocol only uses EnvelopedData, SignedData and EncryptedData.  See
section 3.6 for more details.

-- otherMsgSequence allows for other arbitrary data items to be
placed into the enrollment protocol.  The {OID, any} pair of values
allows for arbitrary definition of material.  Data objects are placed
here while control objects are placed in the controlSequence field.
See section 3.7 for more details.
```

La structure ASN.1 du contenu « `ResponseBody` » est la suivante :

```
ResponseBody ::= SEQUENCE {
        controlSequence   SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
        cmsSequence       SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
        otherMsgSequence  SEQUENCE SIZE(0..MAX) OF OtherMsg
}
```

La signification des différents champs :

```
-- controlSequence consists of a sequence of control attributes.  The
control attributes defined in this document are found in section 3.5.
Other parties can define additional control attributes.

-- cmsSequence consists of a sequence of [CMS] message objects.  This
protocol only uses EnvelopedData, SignedData and EncryptedData.  See
section 3.6 for more details.

-- otherMsgSequence allows for other arbitrary items to be placed
into the enrollment protocol.  The {OID, any} pair of values allows
for arbitrary definition of material.  Data objects are placed here
while control objects are placed in the controlSequence field. See
section 3.7 for more details.
```

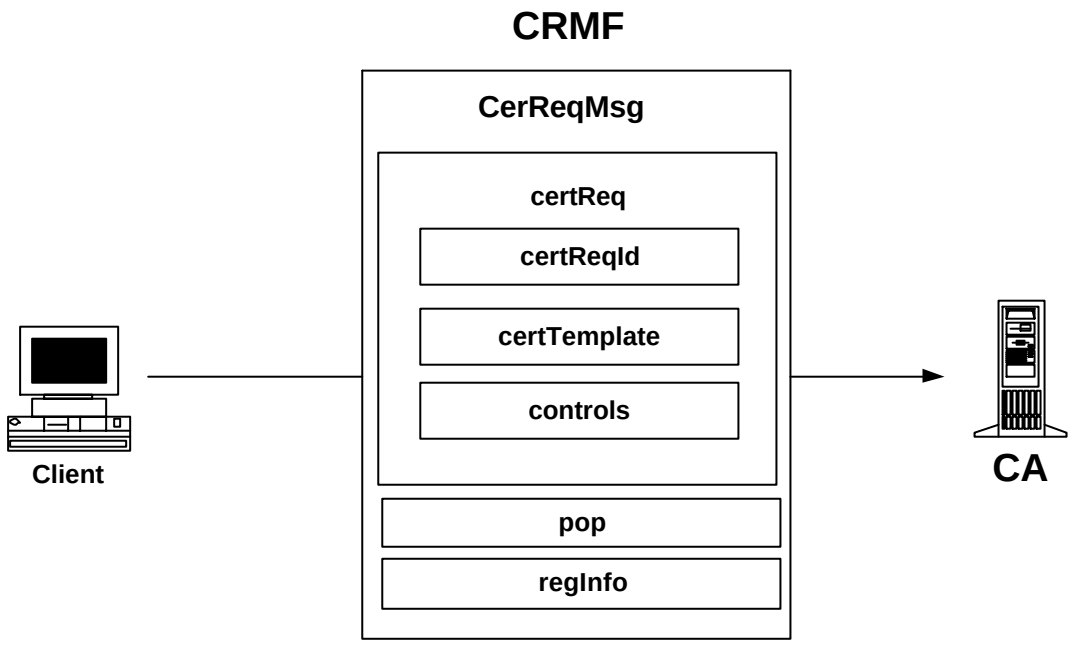
La réponse est un objet « **`signedData`** » sans objet « **`signerInfo`** ». La requête de certificat est retournée dans le « **`certificat bag`** » de l'objet « **`signedData`** ».

4.7 Certificate Request Message Format (CRMF) [RFC2511 - 25pages]

Ce message de demande de certificat « **`CertReqMessage`** » est composé de la demande de certificat « **`certReq`** », un champ facultatif de preuve de possession « **pop** : Proof Of Possession » et d'un champ information d'enregistrement « **`regInfo`** » facultatif également. Ceci se traduit au format ASN.1 :

```
CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

CertReqMsg ::= SEQUENCE {
        certReq   CertRequest,
        pop       ProofOfPossession  OPTIONAL,
        -- content depends upon key type
        regInfo   SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue OPTIONAL
}
```

Figure 4.9 Requête de certificate CRMF



Le champ « **`pop`** » est utilisé pour vérifier que l'entité associée au certificat est actuellement en possesion de sa clé privée correspondante.

Le champ « **`regInfo`**» contient des informations complémentaires liées au contexte de la demande de certificat. Ces information peuvent inclure des renseignements sur l'abonné, la facturation ou tout autre informations nécessaires à l'accomplissement de la demande.

Preuve de possession (POP) :

Pour empêcher certaines attaques et permettre au CA de vérifier la validité du lien qui existe entre entité et pair de clé.

La syntaxe du type « `CertRequest` » contenu dans « `CertReqMsg` » est décrite ci dessous.

```
CertRequest ::= SEQUENCE {
        certReqId     INTEGER,      -- ID for matching request and reply
        certTemplate  CertTemplate, -- Selected fields of cert to be issued
        controls      Controls OPTIONAL } -- Attributes affecting issuance

CertTemplate ::= SEQUENCE {
        version      [0] Version               OPTIONAL,
        serialNumber [1] INTEGER               OPTIONAL,
        signingAlg   [2] AlgorithmIdentifier   OPTIONAL,
        issuer       [3] Name                  OPTIONAL,
        validity     [4] OptionalValidity      OPTIONAL,
        subject      [5] Name                  OPTIONAL,
        publicKey    [6] SubjectPublicKeyInfo  OPTIONAL,
        issuerUID    [7] UniqueIdentifier      OPTIONAL,
        subjectUID   [8] UniqueIdentifier      OPTIONAL,
        extensions   [9] Extensions            OPTIONAL }

OptionalValidity ::= SEQUENCE {
        notBefore  [0] Time OPTIONAL,
        notAfter   [1] Time OPTIONAL } --at least one must be present

Time ::= CHOICE {
        utcTime        UTCTime,
        generalTime    GeneralizedTime }
```

Pour plus de détail sur les différents champs, la [RFC 2511] est à disposition chapitre 6.

4.8 Simple Certificate Enrollment Protocol (SCEP)

Le *Simple Certificate Enrollment Protocol* a été développé par Cisco Systems. La porté de SCEP est limitée à la publication de certificats pour des dispositifs en réseau, et aussi pour retrouver des certificats et CRLs dans une base de donnée.

SCEP définit les messages de requête et de réponse lors des demandes de certificat. Il définit deux messages. Le premier pour récupérer un certificat et le second pour récupérer une CRL. Il n'y a pas de message défini pour les demandes de révocation.

Une entité commence une transaction d'inscription en créant une demande de certificat au format PKCS#10 et l'envoie au CA. Le FQDN (*Fully Qualified Domain Name*) est exigé pour chaque client SCEP, l'adresse IP et le numéro de série sont des attributs facultatifs. Lors d'une demande de certification *(enrollment procedure)*, le message PKCS#10 contient ces attributs exigés ou facultatifs. Le certificat publié au client SCEP doit avoir un nom dans le champ « `subjectName` » et dans l'extension « `SubjectAltName` » issue du champ « `subject` » de la demande PKCS#10.

Il est important de noter qu'un client nommé Alice@cisco.com est différent d'un autre client nommé Alice@cisco.com avec en plus un attribut adresse IP (ex:171.69.1.129). Du point de vue de la CA, les noms Distingués assignés dans ces deux cas sont des noms distincts.

Une fois que la CA a reçut la demande, soit elle approuve automatiquement la demande et retourne le certificat , soit elle exige que l'entité de fin se mette en attente jusqu'à ce qu'un opérateur vienne manuellement authentifier l'identité de cette entité. Deux attributs (définis dans PKCS#6) sont inclus dans la demande de certificat PKCS#10 un attribut de « `password challenge`» et un attribut « `ExtensionReq` » facultatifs. Le mot de passe est employé pour la révocation.

SCEP est seulement défini pour l'algorithme asymétrique RSA. Il assume une paire de clé publique/privée pour la signature et la gestion des clés. C'est un protocole simple.

4.9 Conclusion

La sélection d'un protocole de gestion PKI est un des aspects les plus complexe du déploiement PKI. Il y a deux protocoles qui peuvent satisfaire aux exigences fondamentales : CMP et CMC.

Cependant, les trois protocoles basés sur PKCS#10 (PKCS#10 avec SSL, PKCS#7 et PKCS#10 et SCEP) ont une installation de base. CMP n'est pas compatible avec cette installation de base. CMC a quelques problèmes d'interopérabilité avec cette base et il y a très peu de produits actuellement disponibles.

PKCS#10 avec SSL et PKCS#10 avec #7 sont les systèmes PKI courants. Ils ont des fonctionnalités limitées et manquent d'attributs.

SCEP possède comme inconvénient une porté très limitée mais il est suffisant pour ses utilisateurs : permet des *firewalls*, des routeurs et des réseaux privés virtuels (VPN) ce qui fournit un élan significatif.

Les avantages du protocole CMP sont que :

- CMP est clairement la solution de force industrielle. Il spécifie plus de messages et de transactions que le reste des protocoles combinés.
- CMP inclut les cinq transactions fondamentales et ajoute des particularités complémentaires, y compris la gestion des clefs et le rétablissement de clefs privées.

Comme inconvénient il y a sa complexité qui a ralenti son acceptation dans le marché.

Le protocole CMC n'est pas tout à fait aussi complet que CMP, mais il spécifie suffisamment de messages et de transactions pour satisfaire à des exigences fondamentales, c'est un bon compromis.

4.10 Références

Voici la liste des documents qui m'ont aidé tout au long de ce chapitre. Ils sont classés par ordre d'importance :

Livre [L1]: Planning for PKI
- Chap 11 : PKI Management Protocols

WILEY (2001)
Russ Housley, Tim Polk

RFC2986 PKCS#10

RFC2510 CMP

RFC2797 CMS

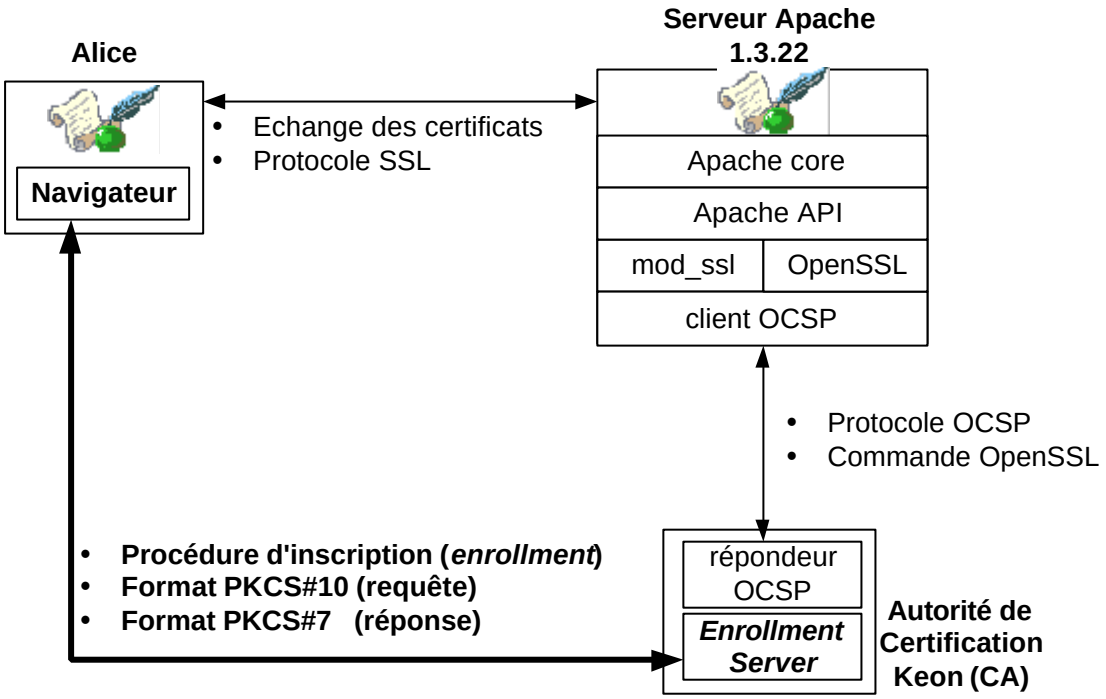
RFC2511 CRMF

http://www.ieng.com/warp/public/cc/pd/sqsw/tech/scep_wp.htm
- Excellent site sur le protocole SCEP avec des diagrammes en flèches (clair et très utile)

5 Procédure d'inscription

Pour illustrer le chapitre précédent, en pratique on effectue une procédure d'*enrollment* entre deux entités en analysant les données qui y transitent. Ici les deux entités sont l'autorité de certification Keon et le demandeur de certificat Alice. La partie concernée par ce chapitre est mise en évidence sur la Figure 5.1:

Figure 5.1 Mise en évidence de la partie : Procédure d'inscription

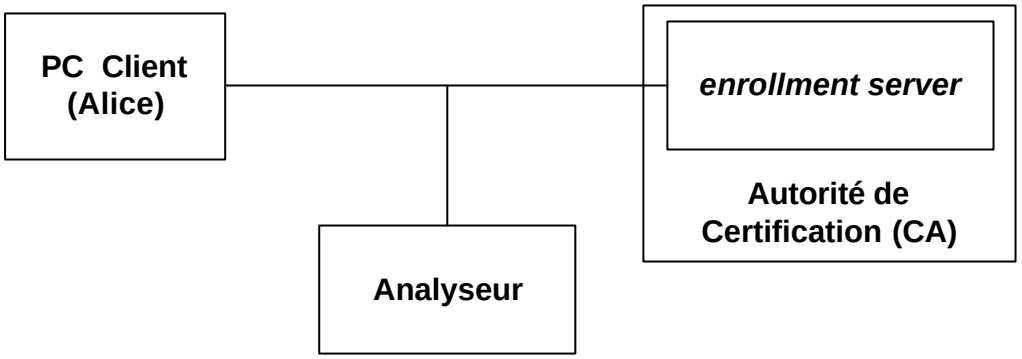


5.1 Principaux Objectifs

L'objectif de ce chapitre est de visualiser les informations échangées entre une autorité de certification et un client qui effectue une demande de certificat. On cherchera particulièrement à mettre en évidence les paquets contenant les informations au format PKCS#10 et PKCS#7. Dans cette étude, nous tirons les informations importantes durant l'échange. Les relevés des formats PKCS#10 et PKCS#7 sont disponibles en ***annexe B*** de ce mémoire**.**

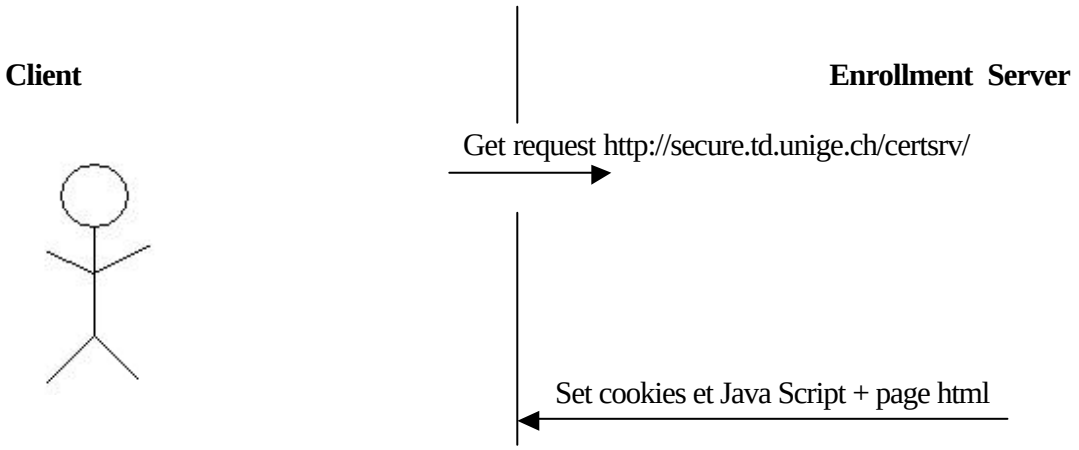
La phase de demande est appelée phase d'inscription (en anglais *enrollment phase)*, le schéma de principe est le suivant :

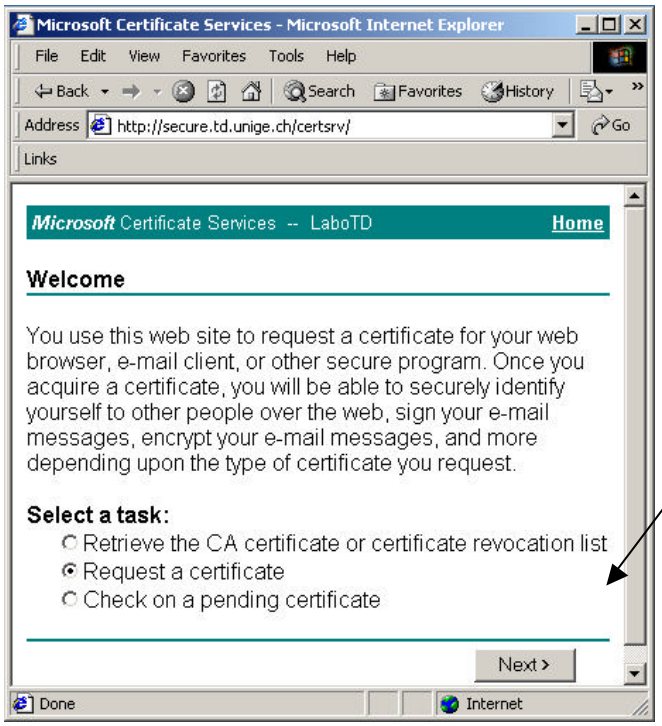
Figure 5.1 Schéma de principe



Nous allons tout d'abord effectuer cet échange avec une autorité de certification de type Microsoft sur le site du laboratoire http://secure.td.unige.ch/certsrv puis nous ferons de même pour l'autorité de certification RSA Keon sur le site https://vectra14.td.unige.ch:443 enfin, nous comparerons les résultats.

5.2 Analyse avec une CA Microsoft





Suivant le choix de requête de l'utilisateur, le serveur charge le fichier.asp correspondant :

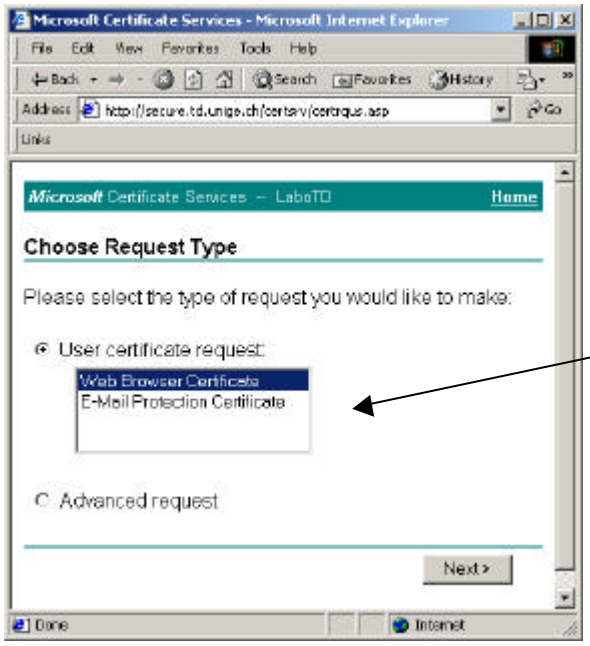
- certcarc.asp pour retrouver un certificat.
- certrqus.asp pour faire une demande de certificat
- certctpn.asp contrôler un certificat en instance

cookies

Set cookies

REQUEST certrqus.asp et cookies

Set cookies et Java Script + page html

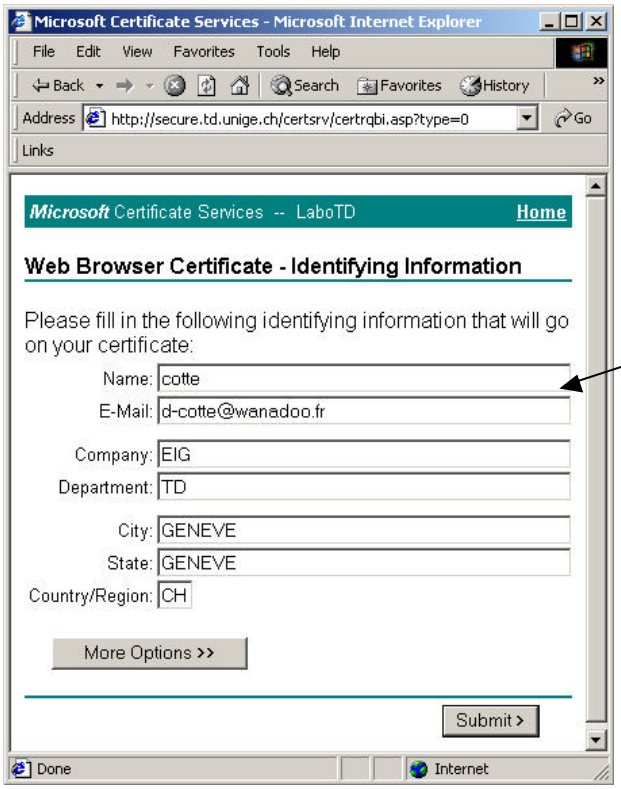


Suivant le type de requête du client, le serveur charge les fichiers.asp correspondant :

- cecrtqbi.asp pour une simple requête de certificat.
- certrqad.asp pour une requête avancée.

REQUEST certrqi.asp + cookies

Set cookies + Java Script + page html



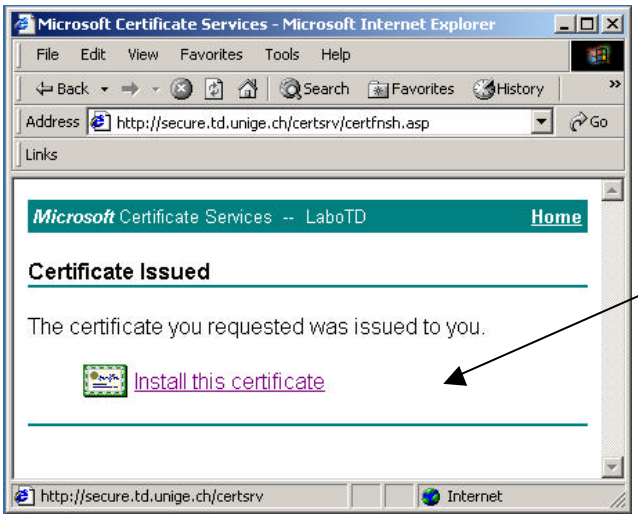
Ici, le client doit remplir les différents champs qui seront ensuite contenu dans le certificat : nom, e-mail, organisation ….

Le code Java Script renseigne sur les prochaines pages à charger suivant les boutons actionnés, il gère aussi les différentes erreurs possibles lors du remplissage des différents champs et enfin il affiche une barre de statut à l'écran.

REQUEST certfnsh.asp et cookies

certificat request codé « pem »**PKCS#10**

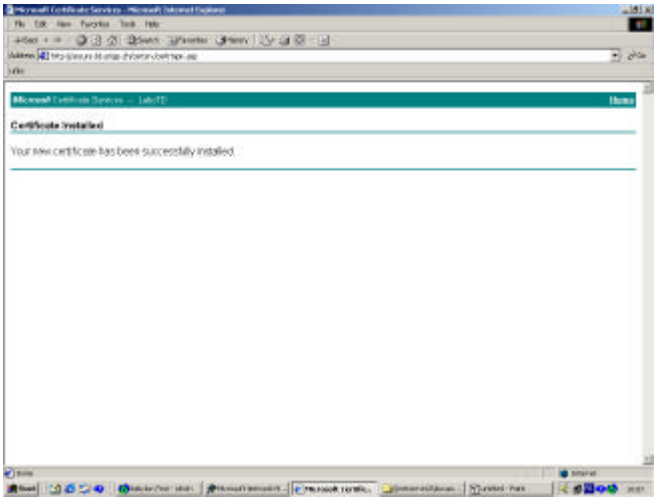
Set cookies + JavaScript + page html + **PKCS#7**



Pour terminer, le client installe son certificat.

REQUEST /certsrv/certrmpn.asp et cookies

page html



5.2.1 Utilité des cookies

Les cookies sont des fichiers comprenant des informations, ils sont produit par un serveur *Web* et stockés dans l'ordinateur de l'utilisateur dans le but d'être utilisable pour un accès futur. Les cookies sont incorporés dans l'information HTML transitant dans les deux sens entre l'ordinateur de l'utilisateur et les serveurs. Les cookies ont été mis en oeuvre pour permettre la personnalisation de côté d'utilisateur d'information du *Web*. Par exemple, les cookies permettent de stocker la liste des articles que l'utilisateur a choisis sur un site commercial. Ici les cookies sont des cookies de session, (ASPSESSIONID) ils sont codés, on ne voit pas les informations qu'ils transportent et valable uniquement pour la session. On ne les retrouve pas dans les fichiers temporaires de l'explorateur à la fin de la transaction.

5.2.2 Fichier ASP

L' **ASP** (*Active Server Pages*) est un standard Microsoft permettant de développer des applications *Web* interactives dont le contenu est dynamique. Il s'agit d'une page *Web* ASP (repérable par l'extension *asp*). Elle aura un contenu pouvant être différent selon certains paramètres (des informations stockées dans une base de données, les préférences de l'utilisateur,...) tandis que page *Web* "classique" (dont l'extension est *.htm* ou *.html*) affichera continuellement la même information.

Un script ASP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII 7 bits (des caractères non accentués) incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur. Ce fichier doit avoir l'extension "asp" pour pouvoir être interprété par le serveur!

Ainsi, lorsqu'un navigateur (le client) désire accéder à une page dynamique réalisé avec les ASP:

- le serveur reconnaît qu'il s'agit d'un fichier ASP grâce à son extension
- il lit le fichier asp
- Dès que le serveur rencontre une balise indiquant que les lignes suivantes sont du code ASP, il "passe" en mode ASP, ce qui signifie qu'il ne lit plus les instructions: il les exécute!
- Lorsque le serveur rencontre une instruction, il la transmet à l'interpréteur
- L'interpréteur exécute l'instruction puis renvoie les résultats sous forme de code HTML
- A la fin du script, le serveur transmet le résultat au client (le navigateur)

Un script ASP est interprété par le serveur, les utilisateurs ne peuvent donc pas voir le source!

Le code ASP stocké sur le serveur n'est donc **jamais** visible directement par le client puisque dès qu'il en demande l'accès, le serveur l'interprète!
De cette façon aucune modification n'est à apporter sur les navigateurs...

Afin de définir les scripts inclus dans le code HTML et interprétés par le serveur, ASP définit une nouvelle balise HTML: `<%  %>`.
A l'intérieur de ces balises, des scripts écrits dans un langage pouvant être:

- du Visual Basic Script (VBScript)
- du Java Script (JScript)
- du Perl
- du Java
- du C++
- ...

Ces scripts, une fois interprétés par le serveur auront pour effet de produire le code HTML envoyé au navigateur, ainsi que des traitements effectués au niveau du serveur et non visible dans le code résultant.

Voici un exemple de script ASP écrit en *VBScript*:

```
<%@ LANGUAGE="VBSCRIPT" %>
<HTML>
<HEAD>
<TITLE>Exemple de script ASP</TITLE>
</HEAD>
<BODY>

<% FOR i = 1 to 10 %>
Bienvenue
<% Next %>

</BODY>
</HTML>
```

Ce script a pour effet de répéter 10 fois l'affichage de la chaîne « `Bienvenue` ». Si on examine les sources HTML avec le navigateur du client, les informations entre les balises « `<%` » ni figure pas. On

peut également utiliser ces portions de code ASP à l'intérieur même d'un programme *JavaScript* pour interpréter une valeur dynamique par exemple.

Dans cette analyse de protocole, nous ne pouvons donc pas visualiser les informations de type ASP puisque le code HTML envoyé au client lors de la phase d'inscription est déjà interprété par le serveur.

5.3 Analyse avec une CA RSA Keon

Pour effectuer le même travail sur l'autorité de certification RSA Keon il faut tout d'abord désactiver le mode SSL qui nous empêche de voir les paquets http. Pour cela, il faut modifier le fichier de configuration de l'*enrollment* serveur nommé `httpd.conf`.

Premièrement remplacer le port d'*enrollment* 443 par le port http 80. Puis modifier le drapeau SSL afin d'arrêter l'échange sécurisé.

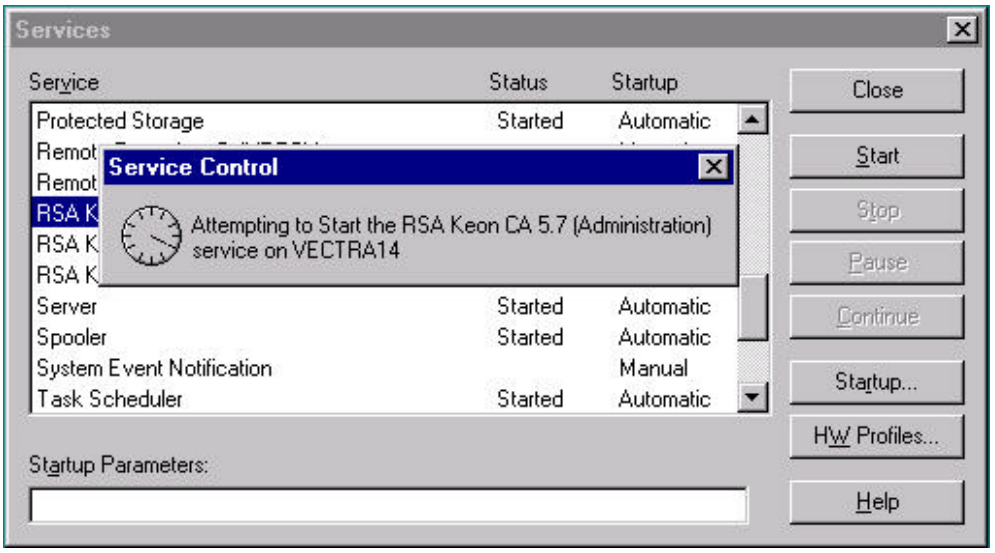
Ci dessous, un extrait du fichier de configuration :

```
###
#
# SSL Directives
#
###

###
# *** We strongly recommend against changing any of the SSL options ***
###

# passage de SSLFlag on en SSLFlag off pour analyse de protocole http
# le 2/10/01
# SSL turned on
SSLFlag off
```

Comme on vient de modifier un fichier de configuration, il est nécessaire de redémarrer les services RSA Keon. Les services sont accessibles depuis le control panel. On sélectionne alors les services RSA Keon dans la liste proposée. Il faut tout d'abord les stopper s'ils sont déjà en marche puis les redémarrer.

On effectue cette dernière opération à l'aide des boutons *Start* et *Stop* de la fenêtre service.

Figure 5.2 Fenêtre des services de Windows NT

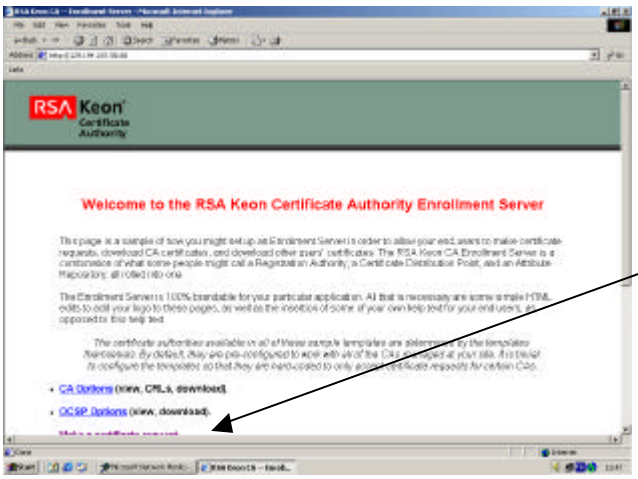


On peut maintenant effectuer l'analyse.

Client ***Enrollment Server***

Get request

Java Script + page html

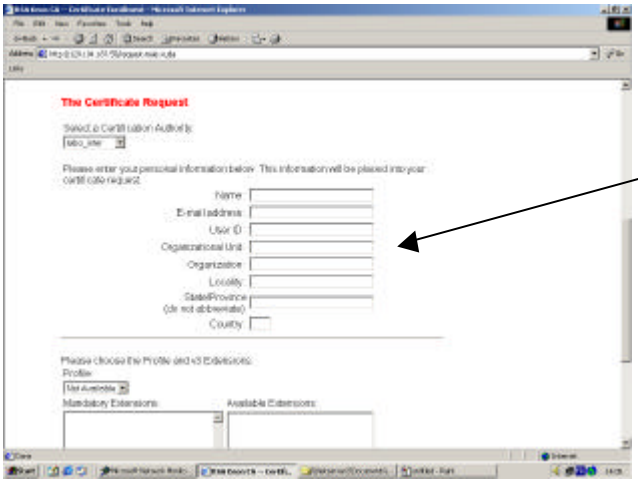


Tout comme précédemment suivant le type d'opération que veut effectuer le client le serveur chargera des fichiers.xuda :

- request-cacert.xuda pour effectuer une lecture.
- request-oscpcert.xuda pour paramétrer la fonction OCSP.
- request-msie.xuda pour faire une requête de certificat.
- etc ….

REQUEST request-msie.xuda

Java Script + page html



Comme plus haut, le client rempli les différents champs de son futur certificat.

REQUEST exterror.js

Java Script

REQUEST exttest.js

Java Script

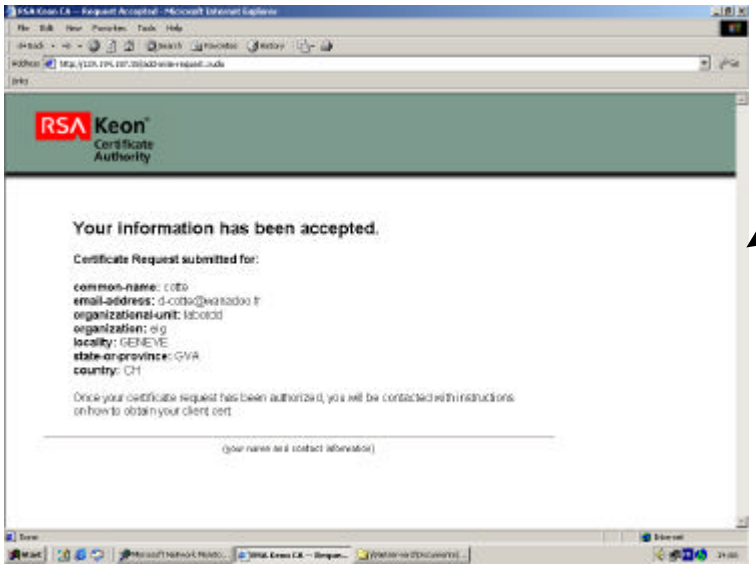
REQUEST profile.js

Java Script

REQUEST add-mise-request.xuda

PKCS#10

Page html

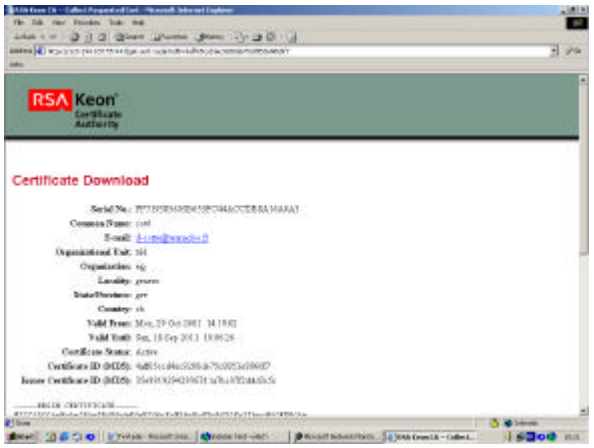


Les informations sont correctes, le client doit maintenant attendre que la requête soit acceptée par un administrateur qui le contactera.
Enfin un e-mail sera envoyé au client avec un lien pour installer son certificat.

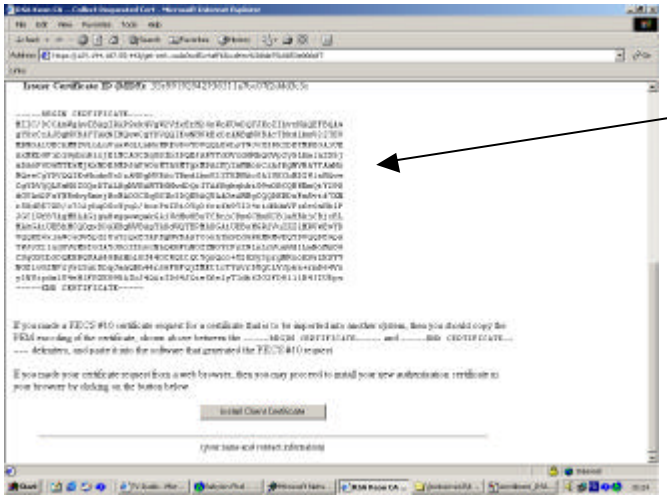
Identifie le certificat

REQUEST /get-cert.xuda?md5=9439f427d01cc24e2040d70074f6c

Page html + **PKCS#7**(+ PKCS#10)



Les informations entrées sont rappelées ici. (haut de page)

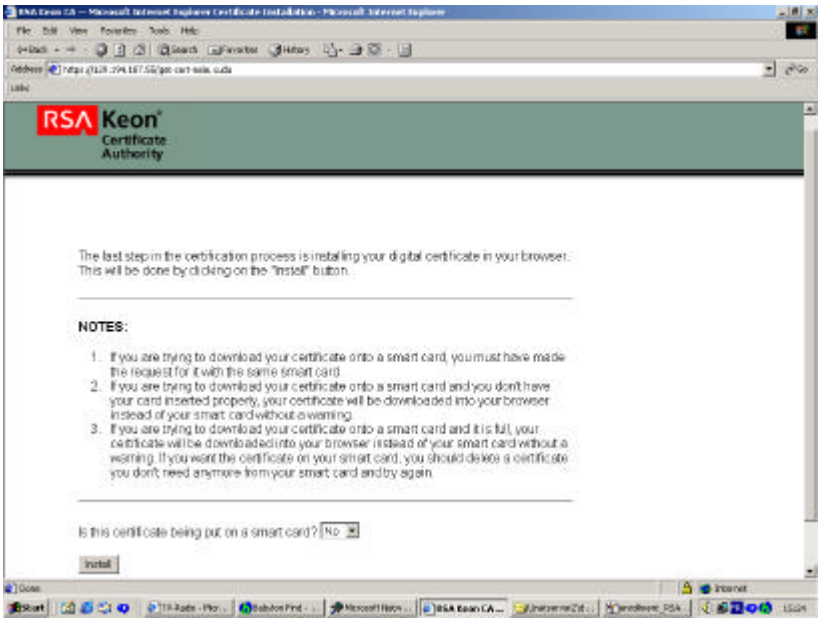


On peut voir notre requête de certification PKCS#10 crypté « pem » (bas de page)

REQUEST /get-cert-msie.xuda

PKCS#7

Page html + **pkcs7Chain**



5.4 Conclusion

Le mécanisme d'*enrollment* de l'autorité de certification Keon est entièrement sécurisé, du début à la fin. Il utilise principalement des fichiers *Java Script* & *VBScript* ainsi que des fichiers avec extension « `.xuda` ». L'utilisation du protocole SSL assure l'authentification de l'autorité de certification au début de la phase s'inscription ainsi que l'intégrité des données échangées. La même procédure pour l'autorité de certification Microsoft utilise des fichiers avec extension « `asp` » pour gérer sa procédure d'*enrollment*. Ces deux méthodes semblent assez similaire car on retrouve du *Java Script* dans les fichiers « `asp` ». Actuellement, la CA Microsoft du laboratoire accepte automatiquement, les requêtes de certificat sans vérifier l'identité du requérant. Elle n'est pas complètement sécurisée avec le protocole SSL ce qui permet en théorie à un pirate d'intercepter les données. Ainsi, n'importe quelle personne mal intentionnée peut obtenir un certificat de cette CA et l'utiliser à mauvais escient.

6 Liste de Certificats Révoqués [RFC2459 – 129 pages]

6.1 Composition d'une liste de révocation

Une CA publie des CRLs *Certificate Revocation List* pour diffuser la liste des certificats révoqués et donc inutilisables dans le futur. Cette CRL doit inclure la date à laquelle la CRL suivante sera publiée dans le champ « `nextUpdate` ».

Une CRL se décompose au format ASN1 comme suit :

```
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
}
```

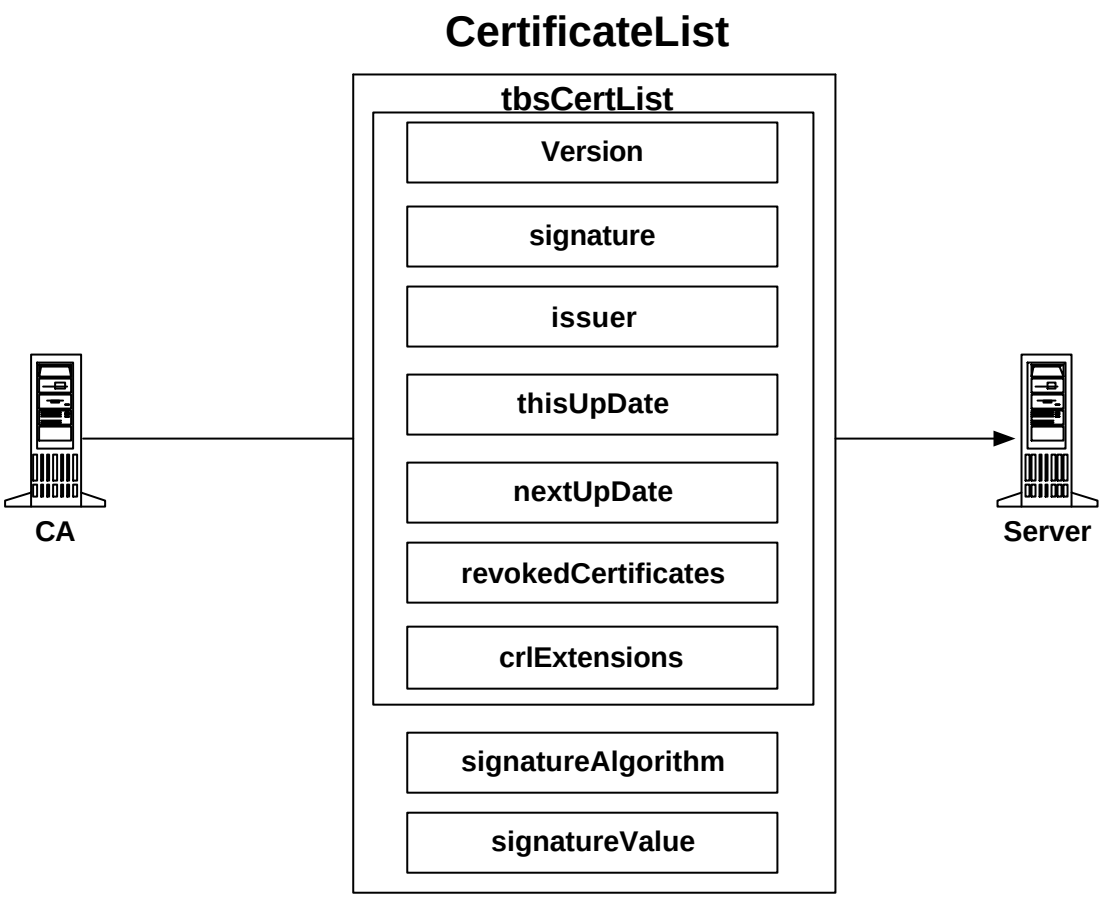
Le premier champ est le « `tbsCertList` ». Ce champ est un ordre contenant le nom de l'émetteur, la date de publication, la date de publication de la liste suivante, la liste de certificats révoqués et des extensions CRL facultatives. Plus loin, chaque entrée dans la liste de certificats révoqués est définie par un ordre de numéro de série de certificat utilisateur, la date de révocation et des extensions d'entrées CRL facultatives

Le champ « signatureAlgorithm » contient l'identificateur d'algorithme employé par la CA pour signer la CRL.

Le champ « `signatureValue` » contient une signature digitale calculée sur l'ASN.1 codée en DER de « `tbsCertList` ». La représentation ASN.1 codée en format DER du champ « `tbsCertList` » est employée comme entrée à la fonction de signature. Cette valeur de signature est une chaîne de bits BIT STRING au format ASN1 le tout inclus dans le champ « `signatureValue` » de la CRL.

La représentation graphique d'une CRL est représentée Figure 6.1 :

Figure 6.1 Représentation graphique d'une CRL



Pour retrouver ces différents champs, on peut aussi visualiser la CRL avec l'outil OpenSSL en utilisant la commande suivante :

```
> openssl crl -in cacrl.pem –text –noout
```

- `openssl crl` : commande de traitement des CRLs
- `-in` : option désignant un fichier en entrée
- `cacrl.pem` : fichier d'entrée
- `-text` : option de visualisation des informations du fichier d'entrée
- `-noout` : option signifiant qu'il n'y a pas de fichier de sortie

Le résultat qui s'affiche est alors le suivant :

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=CH/ST=gva/L=geneve/O=eig/OU=labotd/CN=rootscep/Email=testca@e
ig.unige.ch
        Last Update: Nov 21 10:26:40 2001 GMT
        Next Update: Nov 21 14:02:00 2001 GMT
        CRL extensions:
            X509v3 CRL Number:
            19
            X509v3 Authority Key Identifier:
            keyid:80:E8:68:5D:97:83:5E:11:EF:2B:1A:E1:CD:8A:CE:42:06:04:66:7A

Revoked Certificates:
    Serial Number: 43CDDA1886B4A182B288BBF0126C3B29
        Revocation Date: Nov 14 09:28:52 2001 GMT
    Serial Number: 490C71F2B89B06ACF8A859F35250050E
        Revocation Date: Nov 17 14:36:59 2001 GMT
    Serial Number: E3DD06CA038B1B487C7941F0FD856F02
        Revocation Date: Nov 14 10:08:14 2001 GMT
    Signature Algorithm: sha1WithRSAEncryption
        0e:5a:e3:05:58:f2:e9:91:c8:94:c2:91:47:68:da:47:91:fa:
        34:33:e3:0b:73:eb:69:bd:87:84:50:38:d9:7a:f2:ec:19:b3:
        2a:91:d8:2a:98:ea:0d:16:f4:b5:88:7b:9c:54:2c:e5:cd:4c:
        f6:9d:d1:c8:08:49:df:02:21:70:a5:32:ea:e6:a5:18:7a:bd:
        c5:62:d8:0a:35:4d:ae:1c:f5:9e:2d:bb:1a:c6:d3:86:47:69:
        8a:d7:a9:4d:0f:34:2b:7f:f9:1b:6a:44:04:65:af:34:d8:2c:
        d9:45:0f:47:29:2e:5d:37:ef:64:00:ef:62:ab:97:c5:70:9b:
        c9:bf
```

On peut visualiser les différents champs et notamment le numéro de série des certificats révoqués.

6.2 Conclusion

Le moyen traditionnel de gestion des certificats révoqués est de publier périodiquement une structure de données signées appelée *Certificat Revocation List (CRL)*. Les CRLs sont standardisées et largement utilisées. Une solution existe pour restreindre la quantité d'information qui s'échange entre les différentes entités. On l'appelle la Delta CRL. Il s'agit d'une liste incrémentale qui précise uniquement les changements qui sont arrivés depuis le dernier échange de CRL. Ces Delta CRLs sont donc beaucoup moins lourdes à déployer qu'une CRL complète. La charge moyenne du réseau est de ce fait réduite par rapport à l'utilisation basic des CRLs. Une autre variante pour diminuer les piques de charge du réseau suite aux requêtes est appelée *Over-Issued CRL*. Dans cette dernière, il existe différentes CRLs chacune avec une date d'expiration différente. La charge du réseau est alors distribuée dans le temps. On voit qu'il existe différentes variantes des CRLs qui sont plus ou moins avantageuses. Cependant, la précision d'une CRL reste restreint par sa période de mise à jour ce qui permet à la personne détentrice d'un certificat révoqué d'effectuer des transactions jusqu'à la mise à jour suivante.

6.3 Références

Voici la liste des documents qui m'ont aidé tout au long de ce chapitre. Ils sont classés par ordre d'importance :

RFC 2459

Article: Selecting Revocation Solutions for PKI by André Arnes, Mike Just, Svein J. Knapskog, Steve Lloyd, Henk Meijer.

- Très bon article sur les différents types de CRL et sur OCSP
- Bonne étude sur les temps de simulation de chaque protocole.

7 Online Certificate Status Protocol [RFC2560 – 23 pages]

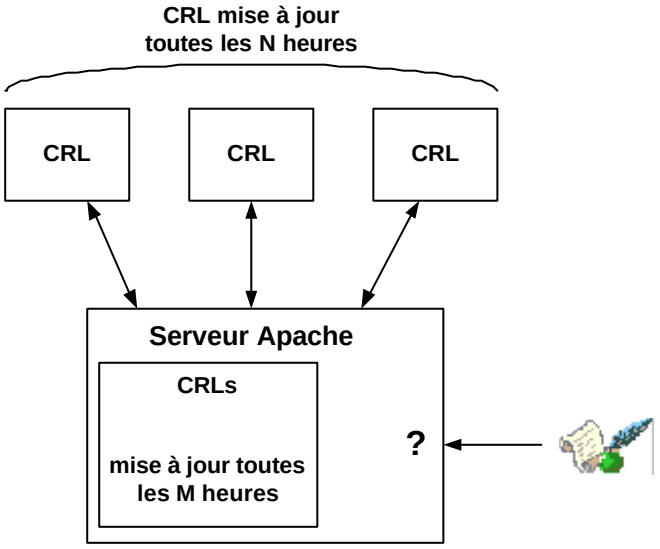
7.1 Problématique

La meilleure approche pour valider les certificats consiste à utiliser un protocole qui permet à un client OCSP d'émettre une requête sur l'état d'un certificat particulier auprès d'une autorité de confiance et si possible en temps réel. La technique classique utilise les CRLs (*Certificate Revocation List*) données par l'autorité de certification émettrice. Cette approche présente deux inconvénients :

- La CRL est **fournie périodiquement**, de telle sorte qu'il existe une intervalle de temps pendant laquelle un certificat qui vient d'être révoqué est considéré comme valide.
- Avec l'augmentation du nombre de CA et donc aussi des CRLs, le processus de gestion de celles ci devient de plus en plus lourd et de moins en moins évolutif.

C'est pourquoi l'IETF (*Internet Engineering Task Force*) a introduit le protocole OCSP (*Online Certificate Status Protocol*).

Figure 7.1 Problématique des CRLs

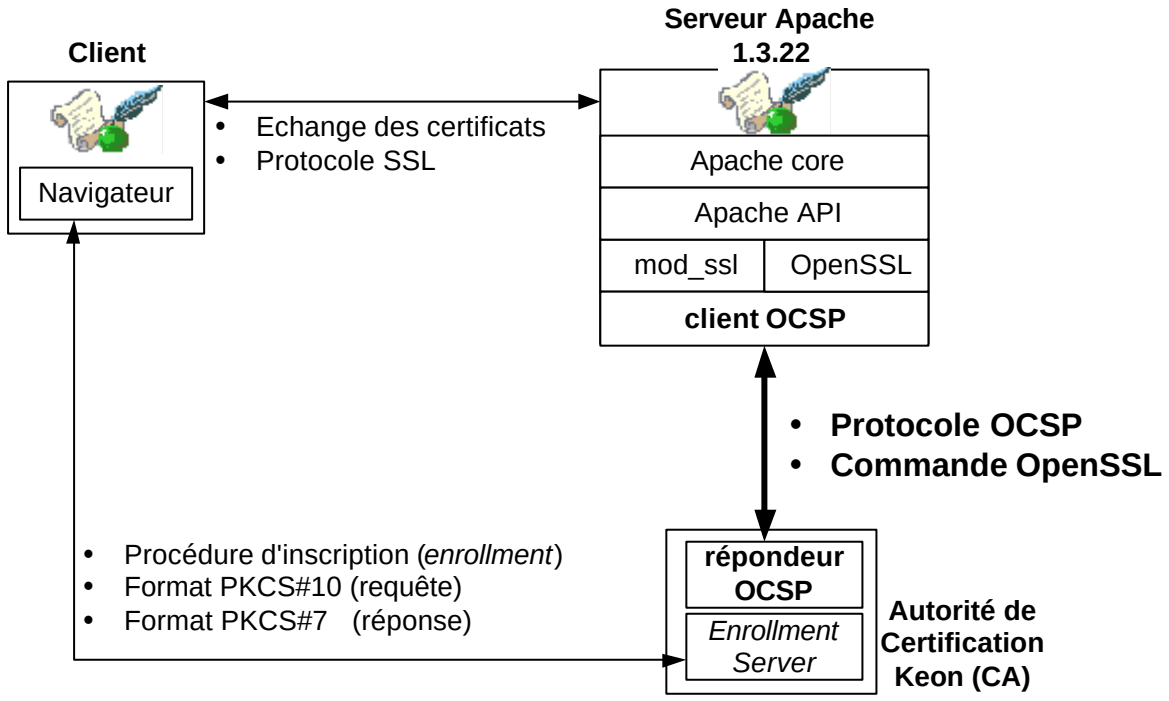


7.2 Définition d'OCSP

Online Certificate Status Protocol est un protocole dans lequel l'information sur la révocation des certificats est disponible « on line » depuis un répondeur OCSP à travers un mécanisme de requête/réponse. Il est utilisé exclusivement pour **vérifier l'état des certificats numériques**.

Dans ce protocole, le serveur demande l'état d'un ou de plusieurs certificats à la fois, au répondeur OCSP. Celui ci vérifie l'état du certificat impliqué et retourne une réponse à l'entité de fin.
La Figure 7.1 met en évidence la partie de l'architecture qui utilise ce protocole.

Figure 7.1 Architecture générale



Le relevé de l'échange OCSP à l'analyseur est disponible en ***annexe C.***

7.2.1 Composition d'une requête OCSP

Une **requête OCSP** se compose des éléments suivants :

- une version du protocole
- le nom du requérant quand la requête est signée
- certaines informations sur le certificat cible, le certificat n'est pas transmis entièrement au répondeur, seul certains champs sont nécessaires.
- les extensions facultatives qui peuvent être traitées par le répondeur OCSP

Ces différents champs sont visibles Figure 7.4

A la réception de cette demande, le répondeur OCSP vérifie si :

- le message est bien formé
- il est configuré pour fournir le service demandé
- la demande contient les informations nécessaires pour être traitée

Si l'une des trois conditions précédentes n'est pas respectée le répondeur produit un message d'erreur, dans les autres cas il renvoie une réponse définitive.

7.2.2 Les différentes erreurs possibles

Le répondeur OCSP peut être amené à renvoyer un **message d'erreur**. Ces erreurs peuvent être de différents types :

- **MalformedRequest** : la demande reçue ne se conforme pas à la syntaxe OCSP.
- **InternalError** : indique que le répondeur OCSP est dans un mauvais état interne, il faut effectuer à nouveau la requête ou changer de répondeur. Dans le cas où le répondeur serait opérationnel, mais incapable de rendre l'état du certificat une réponse « TryLater » est retournée. Cette réponse indique que le service demandé existe mais qu'il est temporairement incapable de répondre.
- **SigRequired** : lorsque le serveur exige que le client signe sa demande pour retourner une réponse.
- **Unauthorized** : lorsque le client n'est pas autorisé à demander le type de service désiré.

Ces différents champs sont visibles Figure 7.5

7.2.3 Composition d'une réponse OCSP

Les **réponses OCSP** peuvent être de différents types, mais il existe une réponse de base qui doit être supportée par tous les serveurs OCSP. Les messages de réponses définitives doivent être **signés**. La clé utilisée pour signer cette réponse doit appartenir à l'une des entités suivantes :

- la CA qui a publié le certificat en question.
- un répondeur approuvé pour lequel le demandeur à confiance en sa clé publique.
- le répondeur désigné d'une CA qui tient à jour une liste de certificat publié et qui peut effectuer des réponses OCSP.

Cette réponse OCSP est composée des éléments suivants :

- la version de la syntaxe de la réponse
- nom du répondeur
- la réponse pour chaque certificat d'une requête qui sera constitué de :
    - l'identificateur du certificat cible
    - la valeur de l'état du certificat
    - l'intervalle de validité de la réponse
    - les éventuelles extensions
- les éventuelles extensions
- l'algorithme de signature (OID)
- la signature calculée avec le condensé de la réponse

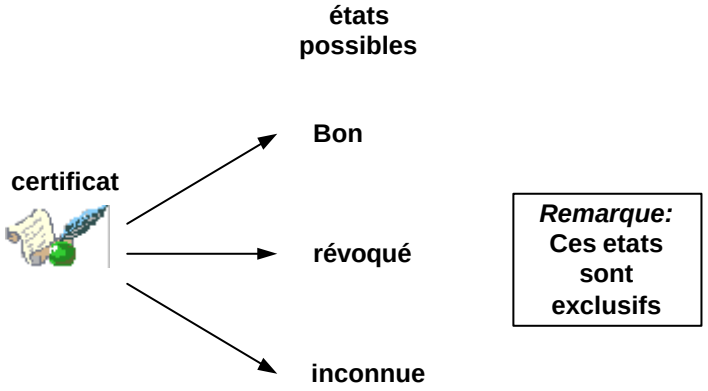
Ces différents champs sont visibles Figure 7.5 - 7.6 - 7.7

Lorsqu'un certificat est dit **«bon »** (*good*), ce résultat indique une réponse positive à la requête d'état. Le certificat n'est pas révoqué. Il ne signifie pas nécessairement que le certificat n'a jamais été révoqué et que le moment où la réponse a été produite est dans l'intervalle de validité du certificat.

L'état « **révoqué** » (*revoked*) indique que le certificat a été révoqué temporairement ou définitivement.

L'état « **inconnu** » (*unknow*) indique que le répondeur n'a pas d'information sur le certificat demandé.

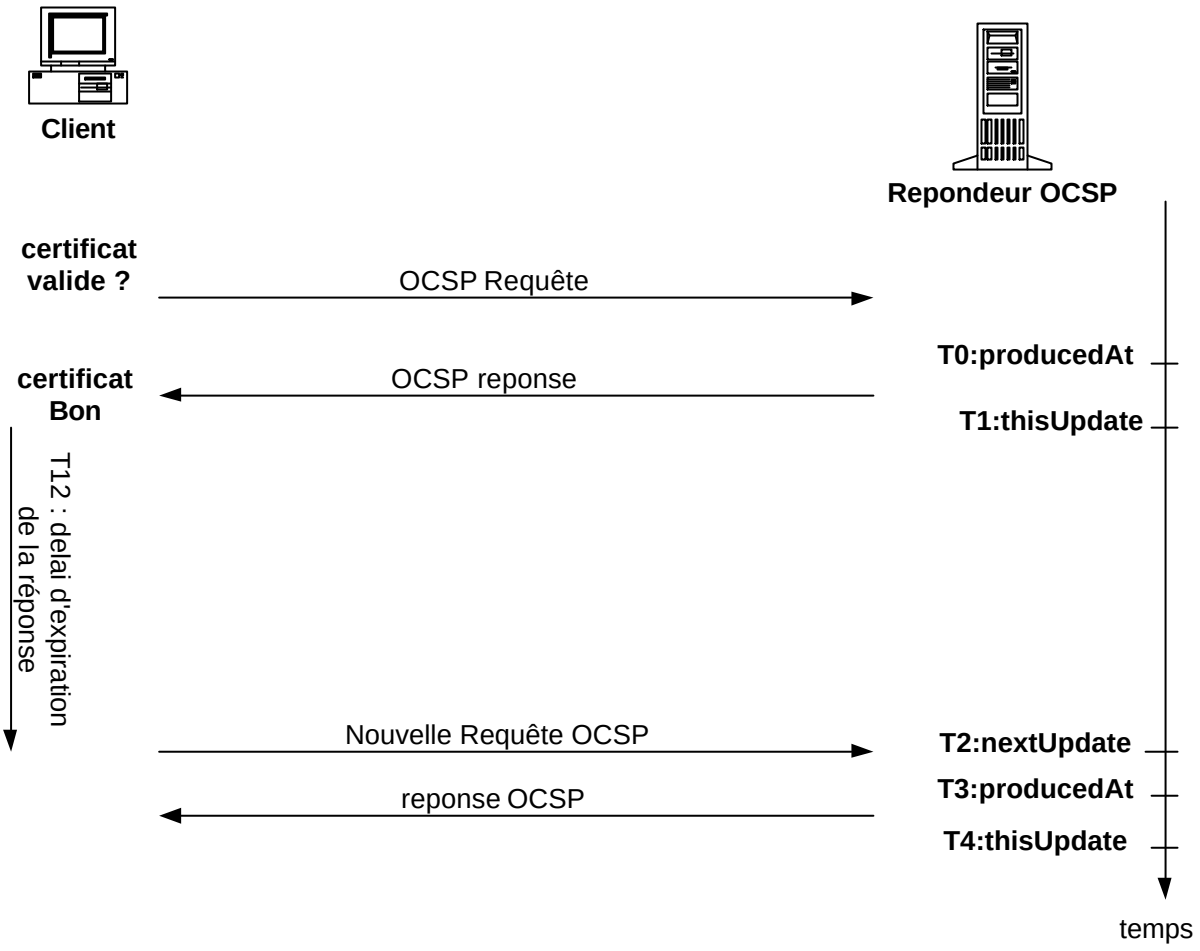
Figure 7.2 Etats possibles d'un certificat



Les réponses peuvent contenir trois champs : *« thisUpdate, nextUpdate, producedAt* » Figure 7.6 et 7.7

- *thisUpdate* : l'heure à laquelle on sait que l'état retourné est correct.
- *nextUpdate* : l'heure à laquelle la nouvelle information sur l'état du certificat sera disponible.
- *ProducedAt* : l'heure à laquelle le répondeur OCSP a signé cette réponse.

Figure 7.3 Représentation temporelle



7.2.4 Délégation d'Autorité de Signature OCSP

La clé qui signe l'information d'état d'un certificat n'a pas besoin d'être la même clé qui a signé le certificat. Sur ce point, l'émetteur du certificat délègue explicitement son autorité de signature au répondeur OCSP. L'émetteur publie un certificat contenant une valeur unique «`extendKeyUsage` » dans la signature des certificats OCSP. Ce certificat doit être publié directement au répondeur OCSP par la CA informée.

Si un répondeur OCSP sait que la clé privée d'une CA particulière a été compromise, il peut retourner l'état « révoqué » pour tous les certificats publiés par cette CA.

7.2.5 Acceptation d'une réponse signée

Avant d'accepter une réponse signée comme étant valide, le client OCSP effectue les vérifications suivantes :

- Le certificat identifié dans la réponse reçue correspond à celui qui a été identifié dans la requête.
- La signature de la réponse est valide.
- L'identité de signataire correspond au destinataire de notre requête.
- Le signataire est actuellement autorisé à signer la réponse.
- L'heure « `thisUpdate` » est suffisamment récente.
- L'heure « `nextUpDate` » vient plus tard dans le temps que le moment présent.

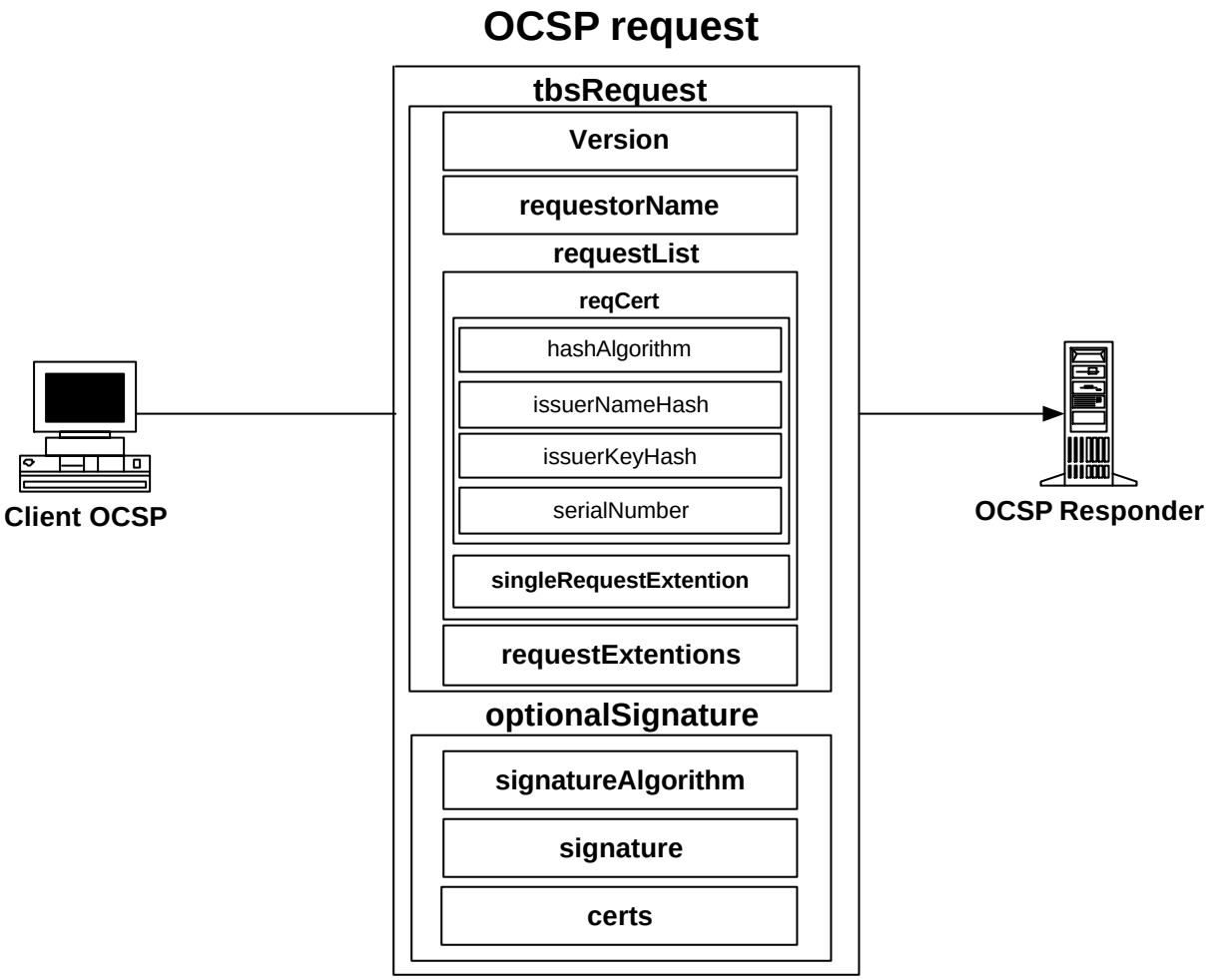
7.3 Requête OCSP au format ASN1

Une requête OCSP est décomposée en ASN1 de la manière suivante :

```
OCSPRequest     ::=     SEQUENCE {
       tbsRequest                  TBSRequest,
       optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

   TBSRequest      ::=     SEQUENCE {

       version             [0]     EXPLICIT Version DEFAULT v1,
       requestorName       [1]     EXPLICIT GeneralName OPTIONAL,
       requestList                 SEQUENCE OF Request,
       requestExtensions   [2]     EXPLICIT Extensions OPTIONAL }

   Signature       ::=     SEQUENCE {
       signatureAlgorithm      AlgorithmIdentifier,
       signature               BIT STRING,
       certs               [0] EXPLICIT SEQUENCE OF Certificate
   OPTIONAL}

   Version         ::=             INTEGER  {  v1(0) }

   Request         ::=     SEQUENCE {
       reqCert                     CertID,
       singleRequestExtensions     [0] EXPLICIT Extensions OPTIONAL }

   CertID          ::=     SEQUENCE {
       hashAlgorithm       AlgorithmIdentifier,
       issuerNameHash      OCTET STRING, -- Hash of Issuer's DN
       issuerKeyHash       OCTET STRING, -- Hash of Issuers public key
       serialNumber        CertificateSerialNumber } [extrait de RFC 2560]
```

Cette structure est plus explicite sur la Figure 7.4

Figure 7.4 Représentation d'une requête OCSP



« `IssuerNameHash` » est le condensé du nom distingué de l'émetteur. Ce condensé est calculé sur le codage DER du champ « `issuerName` » du certificat à vérifier.
« `IssuerKeyHash` » est le condensé de la clé publique de l'émetteur. Le condensé est calculé sur la valeur de la clé publique dans le certificat émetteur.
L'algorithme utilisé pour condenser ses deux valeurs est identifié dans le champ **« `hashAlgorithm` ».**
Le **« `serialnumber` »** est le numéro de série du certificat pour lequel l'état est demandé.

Note sur les condensés: L'emploie de deux condensés (nom de la CA et clé publique) pour identifier l'émetteur du certificat sert à différencier deux CA qui porteraient le même nom. En effet, deux CA n'auront jamais la même clé publique à moins qu'une des deux CA soit compromise où qu'elles aient décidé de partager leur clé privée.

Note sur la signature : Le requérant peut vouloir signer la demande OCSP, dans ce cas la signature est calculée sur la structure de « **`tbsRequest`** ». Si la demande est signée, le requérant doit spécifier son nom dans le champ « **`requestorName`** ». Pour aider le répondeur OCSP à vérifier cette signature le requérant peut inclure des certificats dans le champ « **`certs`** ».
Cette signature sert à authentifier le client OCSP et à garantir l'intégrité des données transmises.

7.4 Réponse OCSP au format ASN1

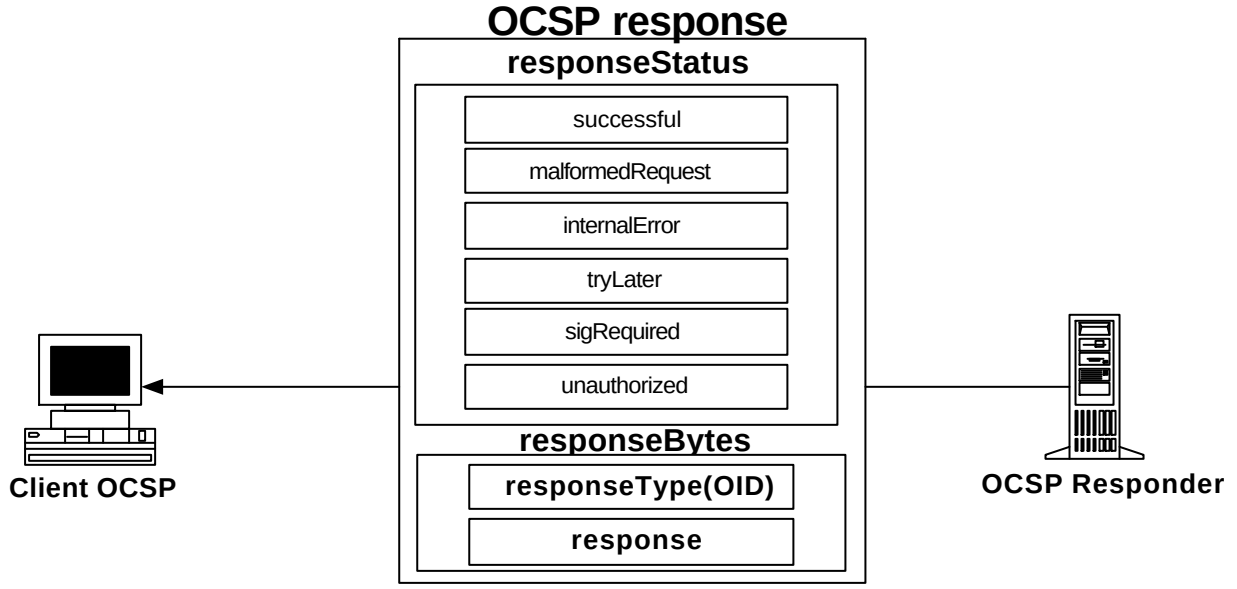
La réponse OCSP au format ASN1 est :

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL
}

      OCSPResponseStatus ::= ENUMERATED {
         successful            (0),  --Response has valid confirmations
         malformedRequest      (1),  --Illegal confirmation request
         internalError         (2),  --Internal error in issuer
         tryLater              (3),  --Try again later
                                   --(4) is not used
         sigRequired           (5),  --Must sign the request
         unauthorized          (6)   --Request unauthorized
      }

      ResponseBytes ::=       SEQUENCE {
         responseType   OBJECT IDENTIFIER,
         response       OCTET STRING }

For a basic OCSP responder, responseType will be id-pkix-ocsp-basic.

        id-pkix-ocsp           OBJECT IDENTIFIER ::= { id-ad-ocsp }
        id-pkix-ocsp-basic     OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }

OCSP responders SHALL be capable of producing responses of the id- pkix-ocsp-basic
response type. Correspondingly, OCSP clients SHALL be capable of receiving and
processing responses of the id-pkix-ocsp- basic response type.

The value for response SHALL be the DER encoding of BasicOCSPResponse.

   BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

The value for signature SHALL be computed on the hash of the DER encoding
ResponseData.

   ResponseData ::= SEQUENCE {
      version              [0] EXPLICIT Version DEFAULT v1,
      responderID              ResponderID,
      producedAt               GeneralizedTime,
      responses                SEQUENCE OF SingleResponse,
```

```
      responseExtensions   [1] EXPLICIT Extensions OPTIONAL }

   ResponderID ::= CHOICE {
      byName               [1] Name,
      byKey                [2] KeyHash }

   KeyHash ::= OCTET STRING -- SHA-1 hash of responder's public key
   (excluding the tag and length fields)

   SingleResponse ::= SEQUENCE {
      certID                       CertID,
      certStatus                   CertStatus,
      thisUpdate                   GeneralizedTime,
      nextUpdate         [0]       EXPLICIT GeneralizedTime OPTIONAL,
      singleExtensions   [1]       EXPLICIT Extensions OPTIONAL }

   CertStatus ::= CHOICE {
       good        [0]     IMPLICIT NULL,
       revoked     [1]     IMPLICIT RevokedInfo,
       unknown     [2]     IMPLICIT UnknownInfo }

   RevokedInfo ::= SEQUENCE {
       revocationTime              GeneralizedTime,
       revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }

   UnknownInfo ::= NULL -- this can be replaced with an enumeration
   (Extrait de RFC 2560)
```

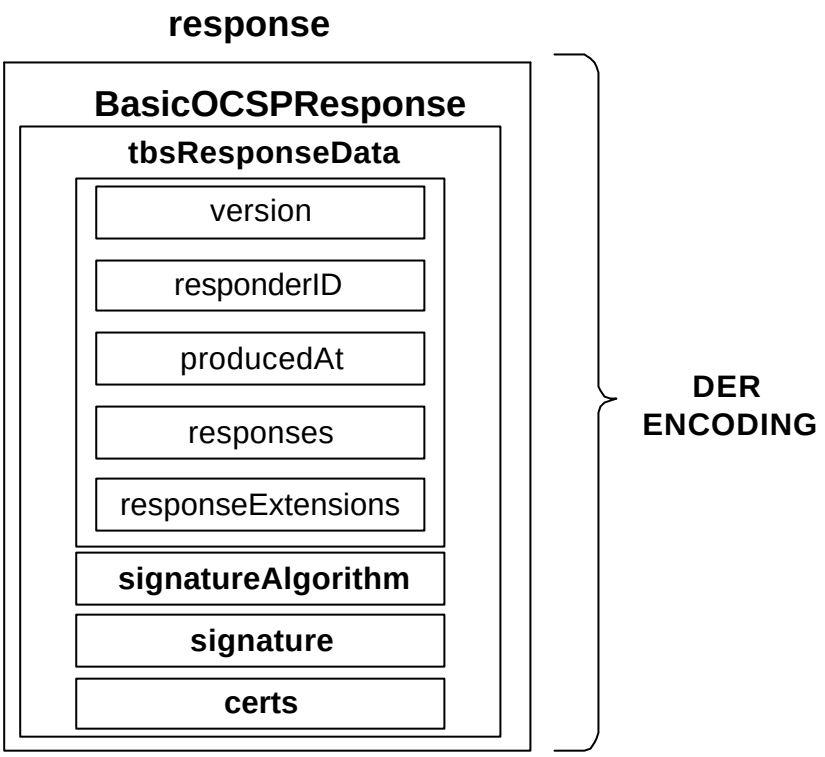
Cette structure est plus explicite sur la Figure 7.5 :

Figure 7.5 Représentation d'une réponse OCSP



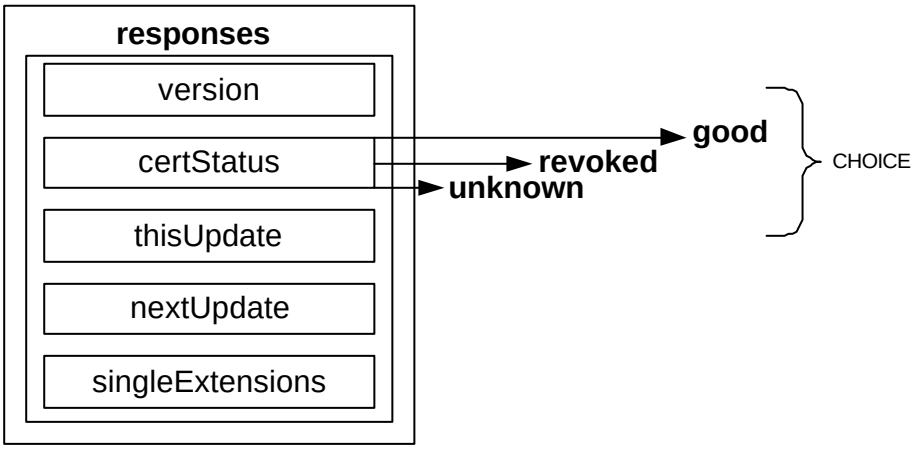
Le champ « **`response`** » peut se décomposer comme ceci :

Figure 7.6 Décomposition du champ « `response` »



A nouveau, le champ « **`responses`** » se décompose comme suit :

Figure 7.7 décomposition du champ « `responses` »



7.5 Temps de validité

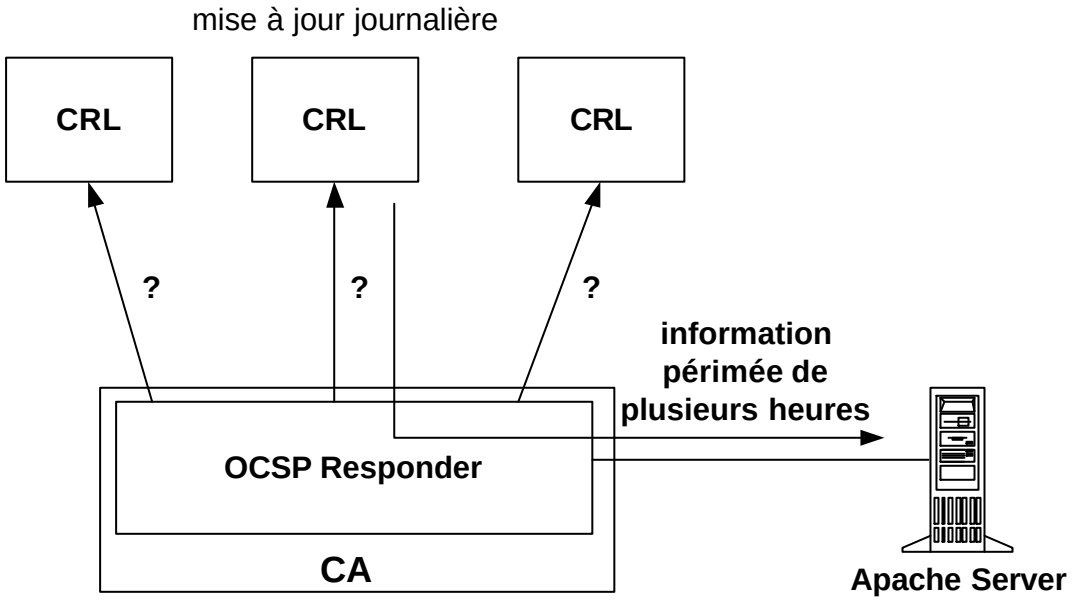
Les champs « `thisUpdate` » et «`nextUpdate` » définissent une intervalle de validité recommandée. Cette intervalle correspond à l'intervalle {`thisUpdate`, `nextUpDate`} dans les CRLs. Les règles de sécurité suivantes s'appliquent :

- Si le champ « `nextUpDate` » d'une réponse fait état d'un instant passé, résolu par rapport au temps du système local, cette réponse doit être considérée incertaine.
- Si le champ « `thisUpdate` » d'une réponse fait état d'un instant futur par rapport au temps du système local, cette réponse doit être considérée incertaine.
- Les réponses ne présentant pas de champ «`nextUpdate` » sont considérées équivalentes à une CRL sans intervalle de mise à jour.

Problématique du temps :

Le répondeur OCSP doit posséder une base de donnée où l'information sur les certificats est mise à jour en temps réel. Ce problème est illustré Figure 7.8. Supposons que le répondeur cherche l'information sur l'état d'un certificat dans une liste qui n'est mise à jour que périodiquement, typiquement toutes les 24 heures, on retrouve la même problématique qu'avec des CRLs classique mais à un niveau supérieur. Dans ce cas l'utilisation d'OCSP n'apporte rien de plus qu'une CRL.

Figure 7.8 Problématique des sources d'informations de type CRL.



Sur la CA Keon, « *Secure Directory Server* » possède une base de données sécurisée ou sont stockés tous les certificats. Pour être valable, cette base de donnée est modifiable en temps réel et sert de source d'information au répondeur OCSP qui donne alors des réponses valides aux clients OCSP.

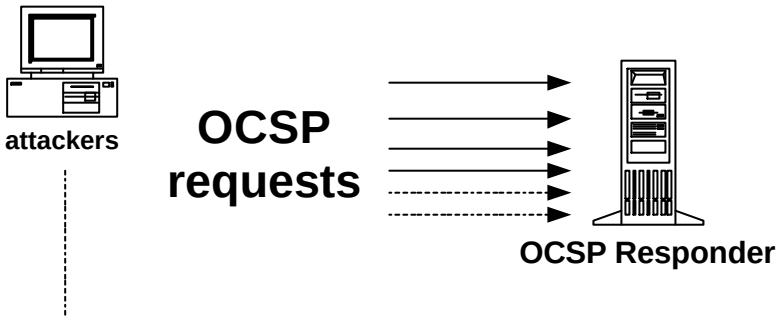
Plusieurs articles montrent que l'utilisation d'OCSP réduit considérablement le trafic en entré/sortie du *« Secure Directory Server »* où sont stockés les certificats. Il devient même insignifiant par rapport au trafic relevé lors de l'utilisation des CRLs qui est environ 100 fois plus important. Une étude réalisée sur la manipulation de plusieurs millions de certificats avec les CRLs et OCSP démontre largement l'avantage d'OCSP sur la charge du trafic autour du *« Secure Directory Server »*.

7.6 Considération de sécurité

Pour que ce service soit efficace, il faut évidemment que le système utilisant les certificats (client) puisse joindre le répondeur OCSP pour que celui ci lui fournisse l'état des certificats. Si ce n'était pas le cas il peut trouver une solution de secours en utilisant les CRLs.

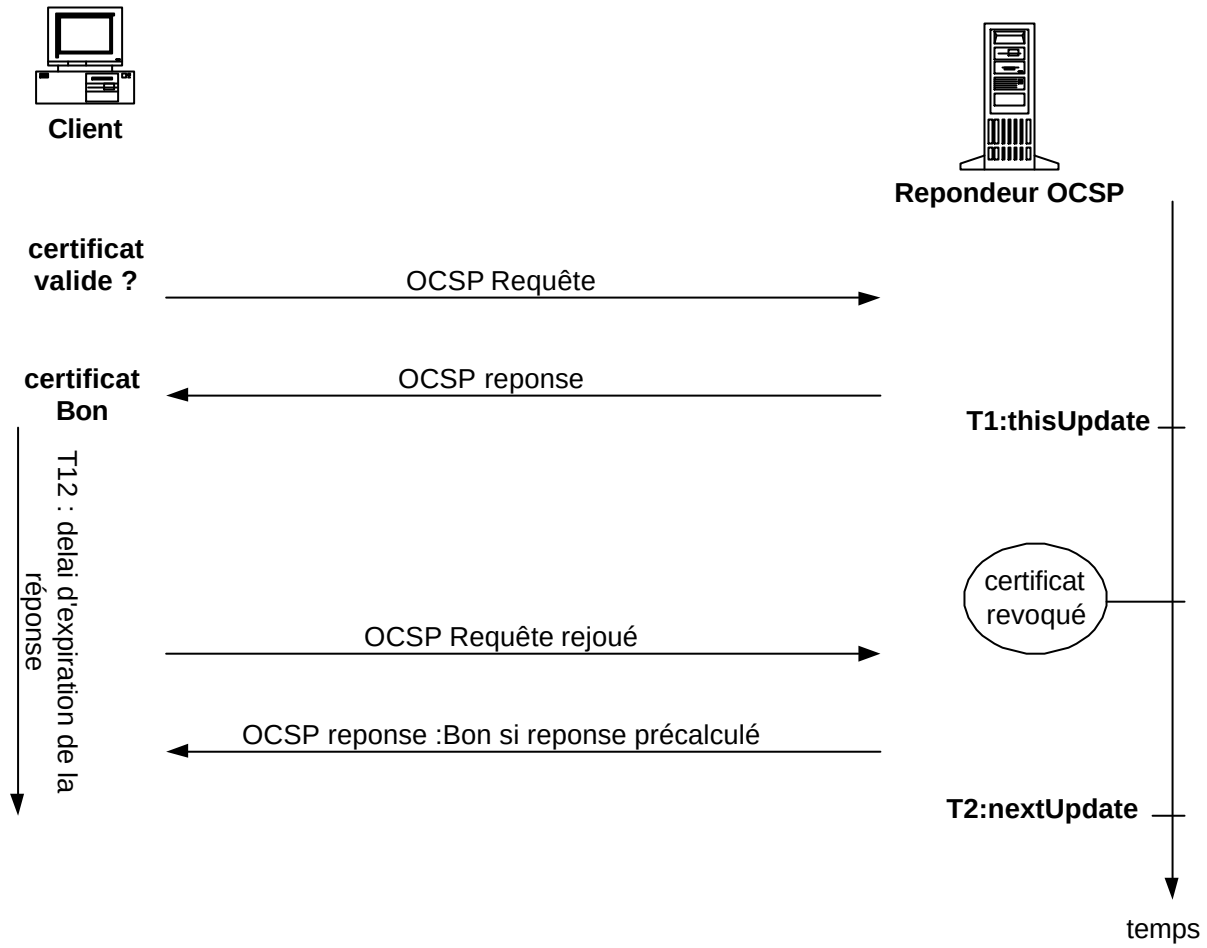
On pressent évidement une vulnérabilité à l'attaque « denial of service » sur ce système ; par exemple en inondant le répondeur OCSP de demandes (illustration Figure 7.9). La production d'une signature cryptographique affecte significativement le temps de cycle pour générer les réponses, ce qui aggrave la situation en cas d'attaque de ce type.
Les réponses erronées et non signées ouvre l'accès à un autre exemple d'attaque de type « déni de service » où les méchants envoient des fausses réponses erronées.

Figure 7.9 Attaque de type « déni de service »



L'utilisation de réponses pré-calculées permet l'attaque de type « *replay* » dans laquelle une vieille bonne réponse est rejouée avant sa date d'expiration mais après que le certificat correspondant ait été révoqué. C'est pourquoi l'utilisation de réponses pré-calculées est dangereuse et doit être soigneusement évaluée par rapport à la probabilité d'une attaque de ce type. Une illustration est faite Figure 7.10

Figure 7.10 Les réponses pré calculées



Les requêtes ne contiennent pas obligatoirement le nom du répondeur auquel elles sont adressées. Cela permet à un méchant de rejouer une demande à un nombre important de répondeurs OCSP. Pour parer cette attaque il existe un « *nounce* » qui lie cryptographiquement une demande et une réponse OCSP mais ce « *nounce* » est utilisé en extension et donc optionnel.

7.7 Conclusion

Le protocole OCSP est une bonne solution pour vérifier la validité des certificats en temps réel. De plus on diminue significativement la charge du réseau en effectuant des requêtes OCSP uniquement pour les certificats actuellement en suspend. On a plus besoin de mettre à jour une liste sur la machine serveur Apache ce qui était assez lourd point de vue échange d'informations. Une réponse OCSP doit être signée pour assurer l'intégrité de la réponse et l'authentification du répondeur OCSP. Eventuellement, les requêtes OCSP peuvent être signées pour les mêmes raisons.

7.8 Références

Voici la liste des documents qui m'ont aidé tout au long de ce chapitre. Ils sont classés par ordre d'importance :

RFC 2560 OCSP

Article : Efficient Certificate Status Handling with PKIs: an Application to Public Administration Services by Marco Prandini

- Article sur une implémentation OCSP effectuée à Modena en Italie
- Bonne étude sur les temps de simulation avec OCSP.

Article: Selecting Revocation Solutions for PKI by André Arnes, Mike Just, Svein J. Knapskog, Steve Lloyd, Henk Meijer.

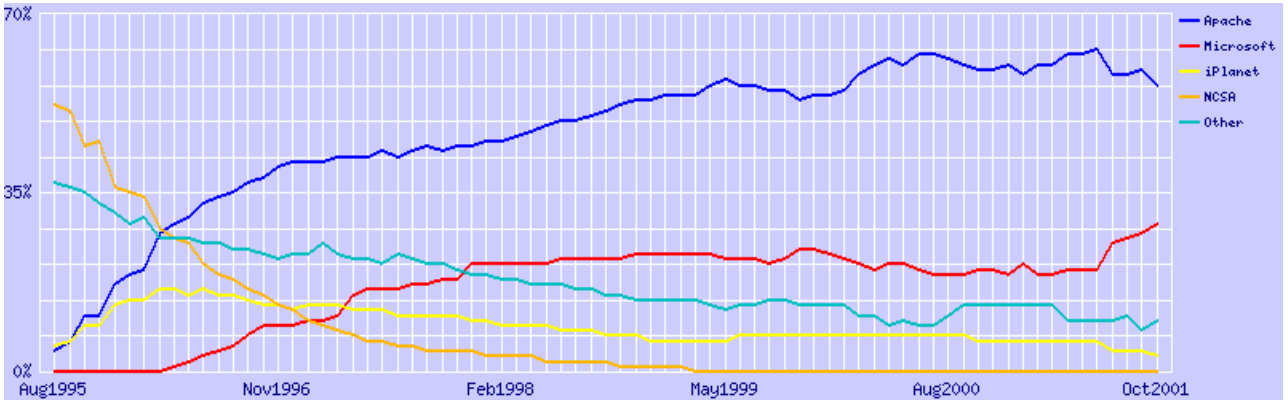
- Très bon article sur les différents types de CRL et sur OCSP
- Bonne étude sur les temps de simulation de chaque protocole.

8 Serveur Apache

8.1 Présentation

Le Projet Apache est un logiciel d'implémentation d'un serveur *Web*. Les codes sources de ce logiciel sont totalement gratuits et disponibles sur le *Web*. Le projet n'a pas été engagé par une seule individualité mais par un véritable groupe de travail constitué de volontaires du monde entier qui ont participé non seulement dans son développement mais aussi dans la constitution de sa documentation. Ces volontaires sont connus sous le nom de Apache Group.
Apache est le serveur *Web* le plus populaire sur l'Internet depuis avril 1996. Une enquête de « *Netcraft Web Server Survey* » datant d'octobre 2001 a démontré que 56 % des sites *Web* sur l'Internet utilisent Apache, celui ci reste donc plus largement employé que tous les autres types de serveurs *Web* combinés.
L'évolution des différents types de serveur sur Internet durant les cinq dernières années est représentée Figure 8.1 :

Figure 8.1 Evolution des différents types de serveur de site *Web* ces 5 dernières années



Au moment du lancement d'Apache, le logiciel dominant était le démon HTTP de NSCA qui représentait près de 51% de l'implantation des serveurs *Web*. Mais il était déjà en déclin après le départ de son créateur de NSCA. Il disparut complètement en mars 1999. Pour succéder à NSCA, trois serveurs étaient en course : Microsoft, iPlanet et Apache.

Microsoft regroupe les logiciels Microsoft Internet Information Server, Microsoft IIS, Microsoft IIS-W, Microsoft-PWS-95 & Microsoft-PWS-95 et Microsoft-PWS.
iPlanet regroupe les logiciels iPlanet-Enterprise, Netscape-Enterprise, Netscape-FastTrack, Netscape Commerce, Netscape Communications, Netsite Commerce et Netsite Communications.

On peut remarquer qu'iPlanet avait une croissance assez proche d'Apache mais décline lentement dès l'arrivée des serveurs Microsoft qui les supplantent rapidement. Apache n'est pas du tout influencé par l'entrée de la société de Bill Gates. Sa gratuité et le fait que les codes sources soient accessibles peuvent expliquer le succès d'Apache.

8.2 Possibilités de configuration

La manière la plus professionnelle d'installer Apache est bien évidemment de le faire à partir des sources, afin de garantir une compatibilité optimale avec le matériel, et surtout afin de pouvoir paramétrer à sa convenance le logiciel.
On peut en effet spécifier de nombreuses options à la compilation d'Apache. en le spécifiant auparavant, soit charger dynamiquement au démarrage du logiciel en l'ajoutant dans le fichier de configuration. Actuellement, plus de 90 modules sont disponibles Le point fort d'Apache est de fonctionner avec des modules, que l'on peut soit ajouter statiquement à la compilation (dont certains payants), mais il est également possible de construire soi-même ses modules et de les incorporer à Apache à l'aide de `apxs`, un utilitaire autonome de compilation dynamique de modules.

La configuration d'Apache s'effectue dans un fichier nommé `httpd.conf`, à l'aide de directives qui sont toutes rattachées à des modules. Il existe trois niveaux logiques de directives :

- les directives de configuration de niveau serveur (contexte global), qui permettent d'établir un paramétrage par défaut du serveur ;
- les directives de conteneur (portée limitée), qui interviennent au niveau de répertoires, fichiers, URL ou méthodes de requête HTTP ;
- les directives de configuration par répertoire, qui permettent de paramétrer chaque répertoire, dans la limitation des droits définis par les directives globales (`AllowOverride`).

L'utilisation des directives dans le fichier de configuration permet de :

- spécifier les emplacements des fichiers de logs ;
- créer des hôtes virtuels sur une même machine (utilisés par les hébergeurs notamment) ;
- effectuer des contrôles d'accès conditionnels par utilisateur (modules `mod_access` et `mod_auth`) ou par adresse IP ;
- définir des types de contenus ou de fichiers à interpréter;
- contrôler et modifier les entêtes HTTP ;
- gérer les erreurs, les alias d'URL et les redirections.

Dans le contexte actuel de développement de l'Internet et par la même occasion des pirates informatiques, la sécurité est une notion essentielle dans la mise en place d'un site *Web*. Il s'agit donc de paramétrer ou configurer Apache au mieux.
Il existe plusieurs moyens de sécuriser l'accès aux ressources d'un serveur Apache :

- authentification des utilisateurs par les modules standards (`mod_auth`) et les modules tiers (LDAP, Smb, Kerberos, Radius, Oracle, etc.)
- sécurisation des requêtes par le protocole SSL (`mod_ssl` + certificat)

Le meilleur moyen de contrôler simplement les accès aux répertoires reste l'utilisation des directives globales d'Apache et l'utilisation des fichiers `.htaccess` dans les répertoires dont les droits sont particuliers.

8.3 Installation d'Apache avec SSL

La procédure d'installation de Linux SUSE 7.0, Apache 1.3.22, modssl 2.5.8 – 1.3.22 est disponible en ***annexe D.***

Un fois le serveur Apache installé correctement , il faut le démarrer en SSL avec les commandes suivantes :

- Se placer dans le répertoire approprié :

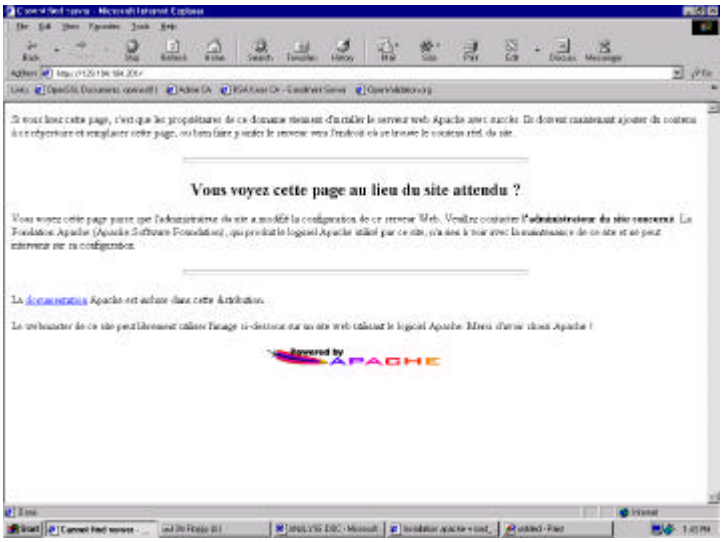
```
bash-2-04# cd /usr/local/apache/bin
```

- Lancer le Serveur en SSL :

```
bash-2-04# ./apachectl startssl
```

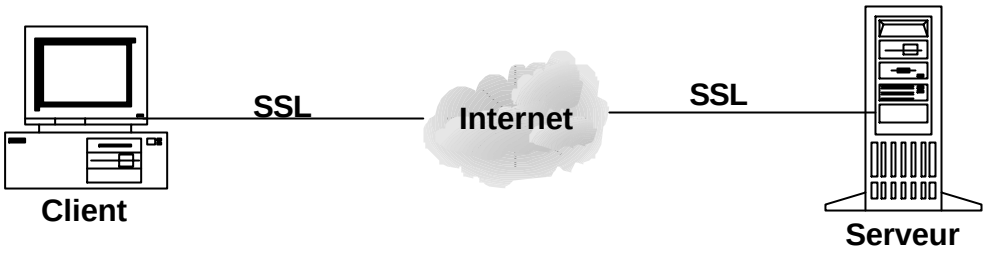
Depuis une autre machine, testons notre serveur Apache SSL : https://129.194.184.201

Figure 8.2 Ecran par défaut d'un client connecté au serveur Apache.



8.4 Module SSL

8.4.1 Mécanisme de SSL



Le client envoie au serveur sa version du protocole SSL, ses paramètres de chiffrement, des données générées aléatoirement et d'autres informations dont le serveur a besoin.

Version, paramètres….

Le serveur renvoie sa version de SSL, ses paramètres de chiffrement, des données générées aléatoirement et d'autres informations dont le client a besoin. Le serveur envoie également son propre certificat, et si le client demande une information nécessitant un certificat, il demande également un certificat client.


Version, paramètre….+

Le client utilise les informations envoyées par le serveur pour l'authentifier. Si le serveur ne peut pas être authentifié, la connexion n'a pas lieu.
Avec les données préalablement échangées, le client est en mesure d'envoyer au serveur une pré clé secrète, qu'il chiffre avec la clé publique du serveur. Si le serveur a requis une authentification du client, celui ci (le client) renvoie également au serveur un bloc de données signé ainsi que son certificat.


Pré clé secret chiffré + éventuellement 

Si le serveur a requis une authentification, il authentifie le client. Le serveur utilise alors sa clé privée de façon à pouvoir déchiffrer la pré clé secrète. Le serveur effectue alors une suite d'actions (également effectuées par le client) pour obtenir une clé secrète à partir de la pré clé secrète. Le client et le serveur utilisent la clé secrète pour générer des clés de session qui seront les clés symétriques utilisées pour le chiffrement, déchiffrement des données et l'intégrité.


Les deux entités calculent la clé symétrique

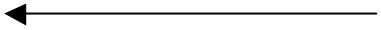

Le client envoie alors un avertissement au serveur le prévenant que les prochains messages seront chiffrés avec la clé de session. Puis il envoie un message chiffré indiquant que la phase de négociation est terminée.

Avertissement prochain échanges chiffrés

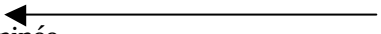
Crypté(fin de la phase de négociation)

Le serveur envoie alors un avertissement au client comme quoi les prochains messages seront chiffrés avec la clé de session. Puis il envoie un message (chiffré cette fois) indiquant que la phase de négociation est terminée.

Avertissement prochain échanges chiffrés

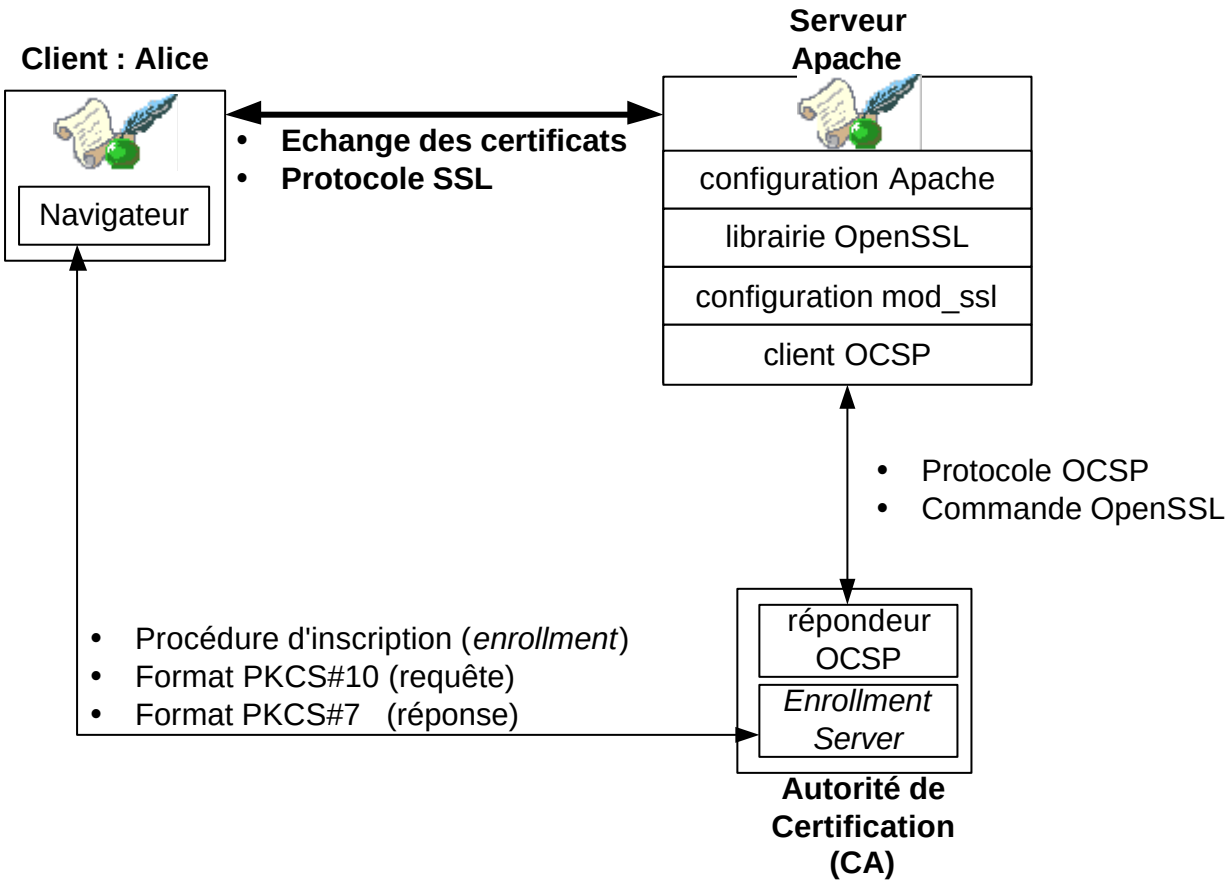


Crypté(fin de la phase de négociation)



La phase de négociation est terminée.
Situons cet échange sur le système mise en œuvre :

Figure 8.3 Architecture générale



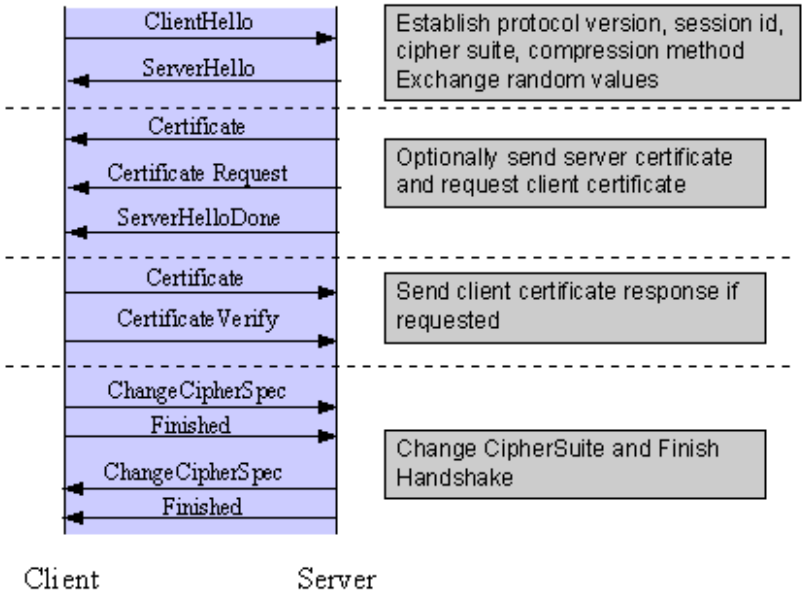
8.4.2 Apache et modssl

Le point fort d'Apache est de fonctionner avec des modules, que l'on peut soit ajouter mais aussi construire soi-même. Ces modules sont disponibles sur le site http://modules.apache.org ainsi que la fonctionnalité de chacun d'entre eux. Dans notre cas, il faut tout d'abord installer le module nommé « *mod_ssl* » qui permet d'implémenter le protocole SSL sur notre serveur Apache et ainsi lui donner la possibilité d'établir des connexions dites sécurisées. En effet *Secure Socket Layer* (SSL) est un protocole de communication réseau destiné à sécuriser les échanges entre deux applications qui communiquent par réseau. Il permet de :

- créer un canal de communication sûr entre les deux applications ;
- d'authentifier chacune des applications aux bouts du canal.

La figure suivante détaille les échanges permettant l'établissement d'une connexion sûre à l'aide de ce protocole.

Figure 8.4 Echange entre client et serveur avec le protocole SSL



Les étapes en sont les suivantes :

- une phase Hello ou le client et le serveur négocient la version du protocole SSL et les algorithmes de cryptages utilisés.
- une phase optionnelle d'authentification du serveur : celui ci transmet son certificat au client qui peut ainsi vérifier la chaîne de certification le validant.
- une phase optionnelle d'authentification du client qui est analogue à celle d'authentification du serveur.

Il existe plusieurs version du protocole SSL : SSLv2 et SSLv3. TLSv1 (*Transport Layer Socket*) est une évolution du protocole SSLv3 standardisée par l'IETF.

Ce module intègre au serveur Apache la cryptographie forte via la couche SSL à l'aide de la bibliothèque mise en œuvre dans le projet OpenSSL. Ce module « *mod_ssl* » a été créé en avril 1998 par Ralf S. Engelschall et a été à l'origine tiré du module Apache-SSL développé par Ben Laurie.

8.5 Authentification du Serveur

8.5.1 Créer un certificat pour le serveur

L'utilisation du module *mod_ssl* avec le serveur apache implique la mise en œuvre de certificat numérique pour l'authentification des deux parties. Dans notre cas, le serveur Apache possède un certificat délivré par l'autorité de certification Keon.

8.5.2 Configuration du serveur pour authentification

Une fois le certificat obtenu, il faut le séparer en trois parties :

- la requête (optionnel)
- le certificat lui même
- la clé privée

On note que si le certificat est généré avec OpenSSL, on a déjà cette répartition dans trois fichiers distincts (voir 1.5). Si par contre le certificat est généré avec une CA Keon, il faut utiliser un petit utilitaire **« *P12toKeyCert.exe* »** qui sépare un certificat au format PKCS#12 (certificat +clé privé) en deux fichiers contenants dans l'un : le certificat et dans l'autre : la clé privée.

Il faut ensuite modifier le fichier de configuration Apache « `httpd.conf` » en décommentant certaines commandes:

```
#   Server Certificate:
#   Point SSLCertificateFile at a PEM encoded certificate.  If
#   the certificate is encrypted, then you will be prompted for a
#   pass phrase.  Note that a kill -HUP will prompt again. A test
#   certificate can be generated with `make certificate' under
#   built time. Keep in mind that if you've both a RSA and a DSA
#   certificate you can configure both in parallel (to also allow
#   the use of DSA ciphers, etc.)
SSLCertificateFile /usr/local/apache/conf/ssl.crt/certificat_apache.crt
#SSLCertificateFile /usr/local/apache/conf/ssl.crt/server-dsa.crt
```

```
#   Server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/certificat_apache.key
#SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server-dsa.key
```

(Extrait de `httpd.conf`)

Enfin, il suffit de placer les fichiers correspondant au chemin spécifié :

- Le fichier contenant le certificat du serveur Apache « *certificat_apache.crt* » dans */usr/local/apache/conf/ssl.crt/*
- Le fichier contenant la clé privée correspondante «*certificat_apache.key* » dans */usr/local/apache/conf/ssl.key/*
- Eventuellement le fichier contenant la demande de certificat « *certificat_apache.csr* » dans */usr/local/apache/conf/ssl.csr/*

Relancer le serveur en SSL :

```
bash-2-04# ./apachectl stop
```

```
bash-2-04# ./apachectl startssl
```

Tester l'authentification du serveur en se connectant avec un navigateur à l'adresse suivante:

https://129.194.184.201

La fenêtre ci dessous apparaît :

L'authentification à lieu, et le client décide si il accepte ce certificat.

8.6 Authentification du Client

8.6.1 Obtenir un certificat pour le client

L'authentification du client n'est pas obligatoire et elle est beaucoup moins utilisée que celle du serveur. Néanmoins, il peut être nécessaire d'authentifier un client. Celui ci doit donc disposer d'un certificat où alors s'en faire délivrer un pour pouvoir s'authentifier auprès du serveur Apache. Pour cela, il procède comme précédemment et demande un certificat à la CA Keon.

8.6.2 Configuration du serveur pour authentification des clients

Pour effectuer une authentification des clients se connectant au serveur Apache il est nécessaire de modifier à nouveau le fichier de configuration « `httpd.conf` » comme suit :

```
#   Certificate Authority (CA):
#   Set the CA certificate verification path where to find CA
#   certificates for client authentication or alternatively one
#   huge file containing all of them (file must be PEM encoded)
#   Note: Inside SSLCACertificatePath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCACertificatePath /usr/local/apache/conf/ssl.crt
SSLCACertificateFile /usr/local/apache/conf/ssl.crt/rootscepca.crt
. . .
#   Client Authentication (Type):
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
SSLVerifyClient require
SSLVerifyDepth  4
. . .
SSLRequire (    %{SSL_CLIENT_S_DN_O} eq "eig" )
. . .
```

SSLCACertificateFile : spécifie tous les certificats « *root* » ou « intermédiaire » concaténé pour lesquels les certificats des clients sont acceptés. Un certificat client doit donc avoir le certificat de sa CA dans ce fichier pour être accepté.

SSLVerifyClient require: permet de spécifier que l'authentification du client est nécessaire pour se connecter au serveur.

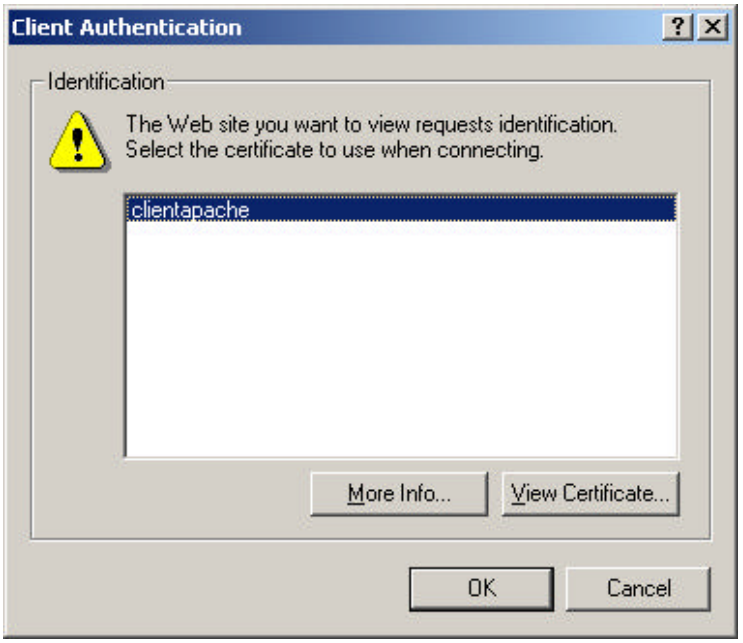
SSLVerifyDepth int : spécifie jusqu'à quelle profondeur dans l'arbre de certification le certificat peut être vérifié.

SSL Require (optionnel) : utilisé pour restreindre les certificats clients accepté. Cette commande laisse valide les certificats clients dont le champ « *Organisation* » du « *Distinguish name* » est « eig » et filtre tout les autres.

Comme on a modifié un fichier de configuration, il faut relancer le serveur Apache en SSL.

Puis, on teste l'authentification du client en se connectant avec un navigateur à l'adresse suivante:
https://129.194.184.201

On demande alors au client de choisir un certificat pour s'identifier (le navigateur n'affiche que les certificats qui passent le test de validité défini par « **`SSLCACertificateFile`** » du fichier « `httpd.conf` »)



Puis, on propose au client le certificat serveur comme vue au point 8.5.2. La page d'accueil du serveur s'affiche, l'authentification du client et du serveur est terminé.

Dans cette configuration, le serveur Apache accepte tous les certificats issus des CA dont le certificat est présent dans le fichier */usr/ocal/apache/conf/ssl.crt/rootscepca.crt*. Hélas, dans le même esprit, il accepte aussi **les certificats révoqués !!**

Il est possible d'utiliser une liste de révocation pour filtrer les certificats clients que le serveur Apache peut accepter et ainsi évité de permettre l'accès au contenu du serveur à un client possédant un certificat révoqué.

8.6.3 Utilisation des CRLs avec Apache

Pour tester le fonctionnement des « *Certicate Revocation List* » on procède en trois étapes :

- Création d'un nouveau certificat pour le client,
- Installation du certificat sur la machine client,
- Révocation du certificat avec l'autorité de certification
- On exporte la CRL de la CA Keon au serveur Apache et on modifie le fichier « `httpd.conf` »

Le fichier « *rootscepcrl.crl* » contenant la CRL a la forme suivante :

```
-----BEGIN X509 CRL-----
MIIBxjCCAS8CAQEwDQYJKoZIhvcNAQEFBQAwgYIxCzAJBgNVBAYTAkNIMQwwCgYD
VQQIEwNndmExDzANBgNVBAcTBmdlbmV2ZTEMMAoGA1UEChMDZWlnMQ8wDQYDVQQL
EwZsYWJvdGQxETAPBgNVBAMTCHJvb3RzY2VwMSIwIAYJKoZIhvcNAQkBFhN0ZXN0
Y2FAZWlnLnVuaWdlLmNoFw0wMTExMTQxMDA4NTRaFw0wMTExMTQxNDAyMDBaMEcw
IQIQQ83aGIa0oYKyiLvwEmw7KRcNMDExMTE0MDkyODUyWjAiAhEA490GygOLG0h8
eUHw/YVvAhcNMDExMTE0MTAwODE0WqAvMC0wCgYDVR0UBAMCAQowHwYDVR0jBBgw
FoAUgOhoXZeDXhHvKxrhzYrOQgYEZnowDQYJKoZIhvcNAQEFBQADgYEAiJ66CKJa
ijaOvlCNEIFfnT3o2fM0kpTUxcC8NxTlWw6d5VDUkTwCJatZSYouxZeJe6fJYDsj
c9RJ6KW7cmQke3WYm7q6LnOSrbGCIjo+4Q9CpGv42NWc0fOZX7IcGZ1bAQzn6bGR
ZF+0zEu9Ox3Yn/D5Lu8+4o8flbBz7bmPbh8=
-----END X509 CRL-----
```

Il faut placer ce fichier à l'endroit suivant sur le serveur Apache :

/usr/local/apache/conf/ssl.crl/rootscepcrl.crl

Puis, on modifie le fichier de configuration « `httpd.conf` » comme suit :

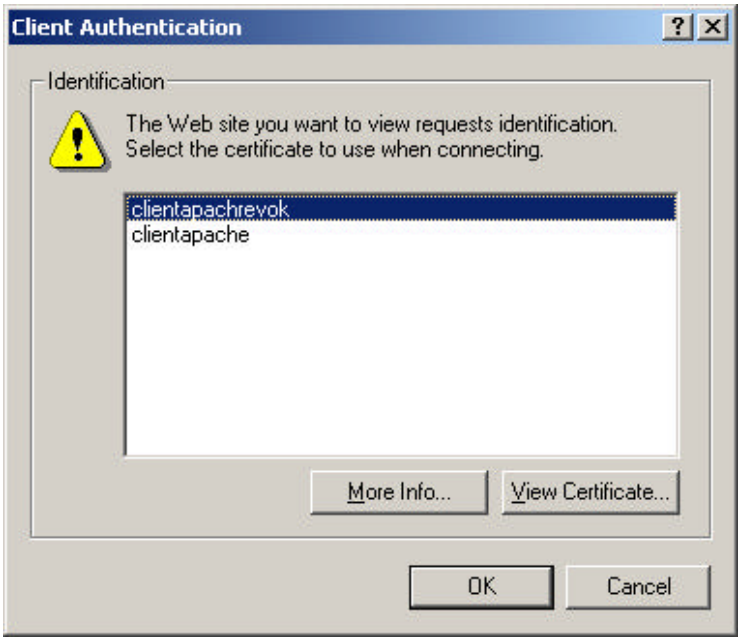
```
#   Certificate Revocation Lists (CRL):
#   Set the CA revocation path where to find CA CRLs for client
#   authentication or alternatively one huge file containing all
#   of them (file must be PEM encoded)
#   Note: Inside SSLCARevocationPath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCARevocationPath /usr/local/apache/conf/ssl.crl
SSLCARevocationFile /usr/local/apache/conf/ssl.crl/rootscepcrl.crl
```

On note que cette liste de révocation possède une intervalle de validité de quelques heures, après cette courte période, les demandes de connexion sont automatiquement refusées.

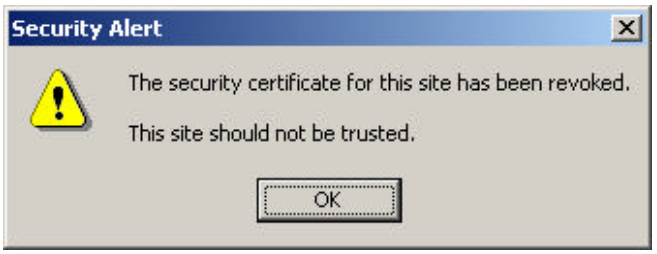
Enfin, on redémarre le serveur Apache.

Résultat :

Le client se connecte au serveur Apache et peut choisir entre ses deux certificats pour s'authentifier :



Le premier est un certificat révoqué et le second est valide. En utilisant le certificat *clientapachrevok* la page d'accueil ne s'affiche pas et un message d'erreur est retourné au client. C'est bien le certificat du client qui est invalide et non celui du serveur Apache.

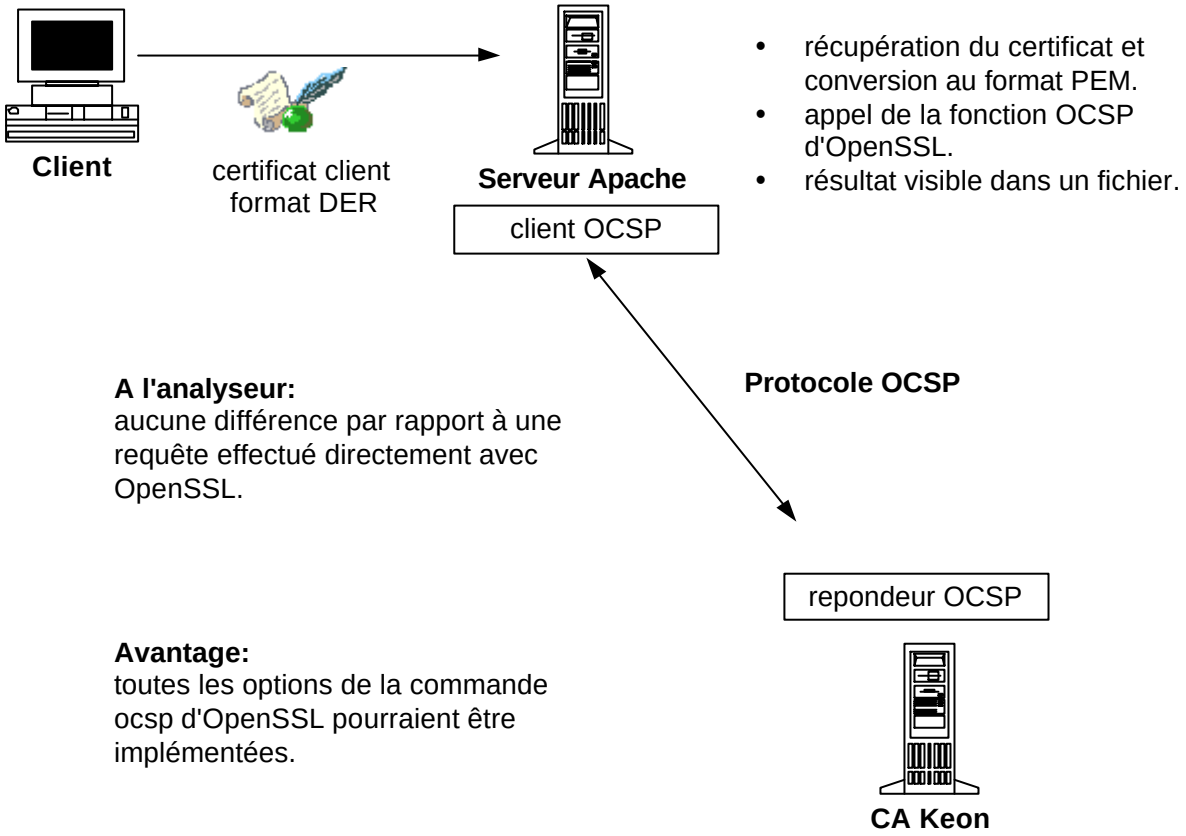


8.7 Implémantation d'OCSP dans le module SSL

8.7.1 Description de l'installation

Le schéma ci dessous représente une vue d'ensemble de la fonctionnalité à mettre en œuvre.

Figure 8.5 Description des fonctionnalités à mettre en œuvre



Comme on vient de le voir, le module SSL implémente la fonction de CRL pour l'acceptation ou du refus du certificat client. Notre but est de réaliser la même fonction avec le protocole OCSP. Au lieu de créer un nouveau module OCSP qui devra interagir avec le module SSL pour les différentes phases d'acceptation ou de refus d'un certificat, on va simplement modifier le module SSL existant en désactivant la fonction CRL et en ajoutant une fonctionnalité OCSP.
Pour cela, l'idée est de récupérer le certificat du client au format PEM et d'appeler la commande OCSP d'OpenSSL en lui donnant ce certificat client en paramètre. Puis, en analysant le fichier de sortie, où est stockée la réponse OCSP on retourne le résultat dans une variable d'environnement. Pour que le client puisse se connecter, la variable d'environnement devra être équivalente à « *good* ». Cette variable est ajoutée au fichier de configuration « `httpd.conf` » dans la partie `SSLRequire`.

8.7.2 Trace des appels de fonction

La première chose à faire est de tracer les différents appels de fonctions du module. Pour cela on modifie les directives `SSLLog` et `SSLLogLevel` du fichier de configuration. `SSLLog` permet de mettre en route la fonction de trace dans le fichier log spécifié. La directive `SSLLogLevel` renseigne sur le niveau de détail de ces traces :

- `none`
  no dedicated SSL logging is done, but messages of level ``error'' are still written to the general Apache error logfile.
- `error`
  log messages of error type only, i.e. messages which show fatal situations (processing is stopped). Those messages are also duplicated to the general Apache error logfile.
- `warn`
  log also warning messages, i.e. messages which show non-fatal problems (processing is continued).
- `info`
  log also informational messages, i.e. messages which show major processing steps.
- `trace`
  log also trace messages, i.e. messages which show minor processing steps.
- `debug`
  log also debugging messages, i.e. messages which show development and low-level I/O information.

Les résultats d'une trace en mode « `info` » est disponible en ***annexe D*** de ce mémoire. En mode « `debug` »,on retrouve le diagramme en flèche de la figure 8.4. Le mode « `debug` », c'est à dire celui qui rentre le plus dans les détails trace aussi ce qu'un analyseur pourrait voir à la sortie du serveur Apache.

Outre ces quelques directives que nous venons de voir, il en existe beaucoup d'autre que je ne vais pas commenter mais qui restent disponibles sur l'excellent site de « mod_ssl » à l'adresse suivantes :

http://www.modssl.org/docs/2.8/ssl_reference.html#ToC19

8.7.3 Structure d'un module

La structure d'un module est l'interface entre le serveur et le module. Elle contient des pointeurs sur les fonctions et les données du module. Voici la description du module exemple :

```
module example_module =
     {
         STANDARD_MODULE_STUFF,
         example_init,               /* module initializer */
         example_create_dir_config,  /* per-directory config creator */
         example_merge_dir_config,   /* dir config merger */
         example_create_server_config, /* server config creator */
         example_merge_server_config,  /* server config merger */
         example_cmds,               /* command table */
         example_handlers,           /* [7] list of handlers */
         example_translate_handler,  /* [2] filename-to-URI translation */
         example_check_user_id,      /* [5] check/validate user_id */
         example_auth_checker, /* [6] check user_id is valid *here* */
         example_access_checker,     /* [4] check access by host address */
         example_type_checker,       /* [7] MIME type checker/setter */
         example_fixer_upper,        /* [8] fixups */
         example_logger,             /* [10] logger */
     #if MODULE_MAGIC_NUMBER >= 19970103
         example_header_parser,      /* [3] header parser */
     #endif
     #if MODULE_MAGIC_NUMBER >= 19970719
         example_child_init,         /* process initializer */
     #endif
     #if MODULE_MAGIC_NUMBER >= 19970728
         example_child_exit,         /* process exit/cleanup */
     #endif
     #if MODULE_MAGIC_NUMBER >= 19970902
         example_post_read_request   /* [1] post read_request handling */
     #endif
   {;
```

MODULE_MAGIC_NUMER est défini dans le fichier `include/ap_mmn.h` et sert à tracer les modifications de l'API d'Apache.

STANDARD_MODUL_STUFF permet d'initialiser l'entête de la structure avec les valeurs par défaut. La plupart des champs seront initialisés à l'exécution.

Les gestionnaires du module qui retournent un entier peuvent générer les valeurs suivantes :

- OK : le gestionnaire a traité la requête
- DECLINED : le gestionnaire ne fait rien
- HTTP_xxxx : un des codes du protocole HTTP (voir httpd.h)

example_init : Cette fonction est appelée à l'initialisation du serveur avant qu'il accepte les requêtes. Elle est exécutée de nouveau à chaque configuration du serveur.
example_create_dir_config : Cette fonction est appelée une fois avec le paramètre **`dirspec`** égal à NULL à l'initialisation du serveur principal et pour chaque bloc **`Location`**, **`Directory`**, **`File`** ou fichier **`.htaccess`** dans lequel apparaît une directive du module.

example_merge_dir_config : Cette fonction est appelée pour fusionner deux structures liées à un répertoire dans le cas d'un héritage.

example_create_server_config : Cette fonction crée la structure liée au serveur pour le module. Elle est appelée une fois pour le serveur principal et ensuite pour chaque serveur virtuel.

example_create_dir_config : Cette fonction est appelée une fois avec le paramètre **`dirspec`** égal à NULL à l'initialisation du serveur principal et pour chaque bloc **`Location`**, **`Directory`**, **`File`** ou fichier **`.htaccess`** dans lequel apparaît une directive du module.

example_merge_dir_config : Cette fonction est appelée pour fusionner deux structures liées à un répertoire dans le cas d'un héritage.

example_create_server_config : Cette fonction crée la structure liée au serveur pour le module. Elle est appelée une fois pour le serveur principal et ensuite pour chaque serveur virtuel.

example_merge_server_config : Cette fonction est appelée pour chaque serveur virtuel avec la structure allouée pour le serveur principal. Cela donne la possibilité de gérer des héritages éventuels.

example_cmds : C'est un tableau de structures du type command_rec de description des directives.

example_handlers : Il s'agit d'un tableau de structures effectuant l'association entre le nom servant au référencement et le gestionnaire.

example_check_user_id : Cette fonction permet au module de vérifier les informations d'authentification. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

example_auth_checker : Cette fonction permet au module de vérifier si la ressource demandée requiert une autorisation. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

example_access_checker : Cette fonction permet au module de vérifier les conditions d'accès à la ressource. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

example_type_checker : Cette fonction permet au module de fixer le type du document. S'il retourne OK, aucun autre module n'est appelé.

example_fixer_upper: Cette fonction est appelée pour effectuer les dernières modifications sur les entêtes.

example_logger : Cette fonction permet au module d'effectuer les enregistrements de traces qu'il souhaite.

example_header_parser : Cette fonction permet au module d'avoir accès à l'entête de la requête au début du processus.

example_child_init : Cette fonction est appelée à l'initialisation du processus serveur avant qu'il accepte les requêtes. Cela permet d'exécuter des actions qui ne doivent être exécutées qu'une fois par processus.

example_child_exit : Cette fonction est appelée à l'arrêt du processus serveur.

example_post_read_request : Cette fonction est appelée après la lecture de la requête mais avant les autres phases. Cela permet, par exemple, de positionner des variables d'environnement

J'ai pris ce module comme exemple car il spécifie toutes les fonctions possibles. Lorsque la fonction n'a pas d'utilité dans le module on ne l'implémente pas, on la remplace par le pointeur NULL. Par analogie, la structure du module SSL est la suivante :

```
module MODULE_VAR_EXPORT ssl_module =
{
    STANDARD_MODULE_STUFF,
    ssl_init_Module,          /* module initializer                  */
    ssl_config_perdir_create, /* create per-dir    config structures */
    ssl_config_perdir_merge,  /* merge  per-dir    config structures */
    ssl_config_server_create, /* create per-server config structures */
    ssl_config_server_merge,  /* merge  per-server config structures */
    ssl_config_cmds,          /* table of config file commands       */
    ssl_config_handler,       /* [#8] MIME-typed-dispatched handlers */
    ssl_hook_Translate,       /* [#1] URI to filename translation    */
    ssl_hook_Auth,            /* [#4] validate user id from request  */
    ssl_hook_UserCheck,       /* [#5] check if the user is ok _here_ */
    ssl_hook_Access,          /* [#3] check access by host address   */
    NULL,                     /* [#6] determine MIME type            */
    ssl_hook_Fixup,           /* [#7] pre-run fixups                 */
    NULL,                     /* [#9] log a transaction              */
    NULL,                     /* [#2] header parser                  */
    ssl_init_Child,           /* child_init                          */
    NULL,                     /* child_exit                          */
    ssl_hook_ReadReq,         /* [#0] post read-request              */

    /* Extended API (forced to be enabled with mod_ssl) */

    ssl_hook_AddModule,       /* after modules was added to core     */
    ssl_hook_RemoveModule,    /* before module is removed from core  */
    ssl_hook_RewriteCommand,  /* configuration command rewriting     */
    ssl_hook_NewConnection,   /* socket connection open              */
    ssl_hook_CloseConnection  /* socket connection close             */
};
```

Il existe une multitude de variable d'environnement très utile pour retrouver différentes informations sur le module SSL. Le tableau suivant renseigne sur la signification des plus importantes :

Variable Name:	**Value Type:**	**Description:**
`HTTPS`	flag	HTTPS is being used.
`SSL_PROTOCOL`	string	The SSL protocol version (SSLv2, SSLv3, TLSv1)
`SSL_SESSION_ID`	string	The hex-encoded SSL session id
`SSL_CIPHER`	string	The cipher specification name
`SSL_CIPHER_EXPORT`	string	`true` if cipher is an export cipher
`SSL_CIPHER_USEKEYSIZE`	number	Number of cipher bits (actually used)
`SSL_CIPHER_ALGKEYSIZE`	number	Number of cipher bits (possible)
`SSL_VERSION_INTERFACE`	string	The mod_ssl program version
`SSL_VERSION_LIBRARY`	string	The OpenSSL program version
`SSL_CLIENT_M_VERSION`	string	The version of the client certificate
`SSL_CLIENT_M_SERIAL`	string	The serial of the client certificate
`SSL_CLIENT_S_DN`	string	Subject DN in client's certificate
`SSL_CLIENT_S_DN_`*x509*	string	Component of client's Subject DN
`SSL_CLIENT_I_DN`	string	Issuer DN of client's certificate
`SSL_CLIENT_I_DN_`*x509*	string	Component of client's Issuer DN
`SSL_CLIENT_V_START`	string	Validity of client's certificate (start time)
`SSL_CLIENT_V_END`	string	Validity of client's certificate (end time)

[where *x509* is a component of a X.509 DN: `C,ST,L,O,OU,CN,T,I,G,S,D,UID,Email`]

8.7.4 Explication du code ajouté

Pour utiliser la commande OCSP d'OpenSSL, j'ai rajouté dans le fichier de configuration « `httpd.conf` » les différentes options de base de la commande OCSP. (***Annexe F***)

Comme cela nécessite différentes petites modifications dans plusieurs fichiers dispersés, ils ne sont pas annexé mais seront disponibles dans le patch OCSP final. Par contre, la partie qui appelle les fonctions OCSP d'OpenSSL est disponible en***Annexe G.***

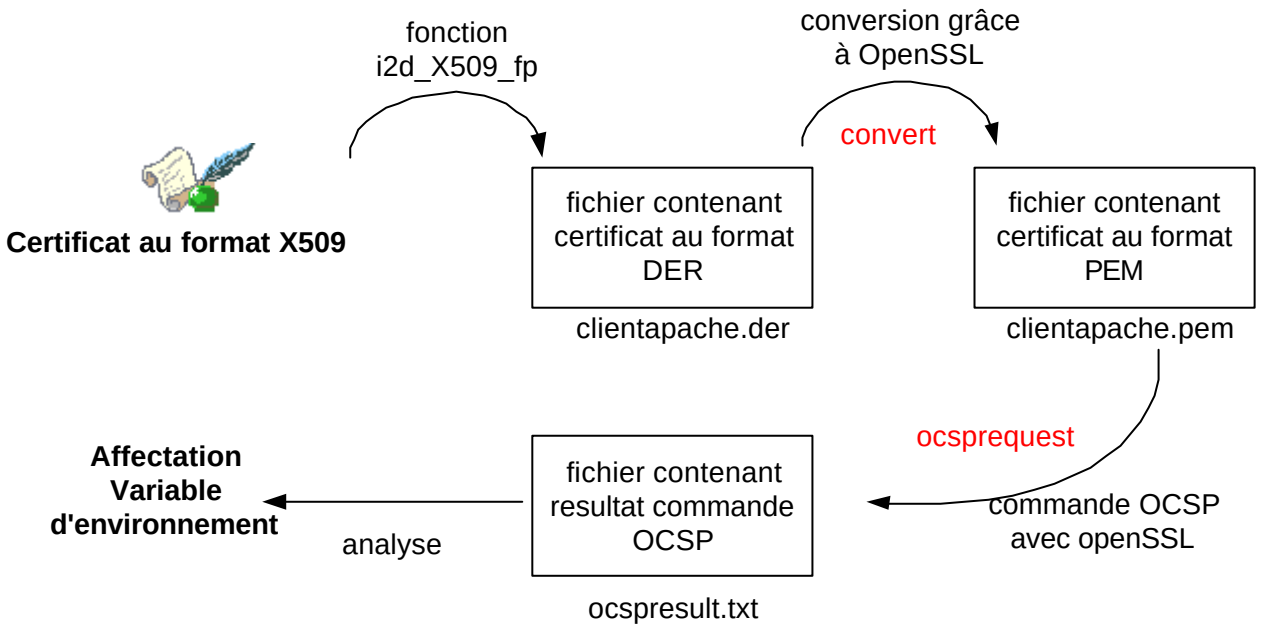
En effet j'ai rajouté deux fonctions dans `ssl_callback_SSLVerify` appelée elle même dans la fonction `ssl_hook_Access.` Elle s'exécute juste après que le client choisit son certificat pour l'authentification mais avant qu'il accepte celui du serveur.

La première fonction `ocsp_check_hook` permet de récupérer le certificat qui transite entre client et serveur et de le convertir du format DER au format PEM le tout dans un fichier « `clientapache.crt` ». La fonction `i2d_X509_fp` permet de récupérer le certificat format DER. Le petit programme «`convert`» effectue la conversion DER en PEM avec OpenSSL.

La seconde fonction `ocsp_check_hook` analyse le résultat de la commande « `ocsprequest` » et affecte une variable d'environnement **SSL_OCSP_LDAP_RESP_STATUS.**

Cette variable d'environnement est ensuite testée dans le fichier de configuration « `httpd.conf` » dans la partie « `SSLRequire` » ce qui permet de filtrer les certificats à accepter suivant l'état de cette variable d'environnement. Un certificat qui retourne un état « *good* » permet l'accès à la page *Web* alors qu'un certificat qui retourne l'état « *revoked* » affiche une erreur 403 qui est par la suite paramétrable. Ces explications sont reprises sur la figure suivante :

Figure 8.6 Explication sur les actions du code OCSP ajouté au module SSL



8.8 Conclusion

J'ai installé et configuré un serveur Apache avec le module « modssl ». Puis, j'ai implémenté la fonction OCSP qui vérifie la validité des certificats que le serveur reçoit. Pour cela, j'ai modifié le code du module SSL. Mon travail sera disponible sous forme de patch pour pouvoir l'exporter sur d'autres serveurs équipés des mêmes versions d'Apache, mod_ssl et éventuellement OpenSSL.

8.9 Références

Voici la liste des sites qui m'ont aidé tout au long de ce chapitre. Ils sont classés par ordre d'importance :

Livre [L2] : Writing Apache Modules with Perl and C
O'REILLY (1999)
Lincoln Stein & Doug MacEachern

http://www.modssl.org/docs/apachecon2001
- Excellente présentation sur la configuration du module mod_ssl.
- Utilisation de l'authentification et des CRLs.

http://www.modssl.org/docs/2.8
- Documentation du module mod_ssl
- Chapitre 3 : commande du fichier « httpd.conf »

http://www.alianwebserver.com/informatique/linux/apache.htm
- Installation d'Apache avec mod_ssl. (mod_php en plus)

http://www.linux-sottises.net/apache_install.php
- Installation d'Apache avec mod_ssl. (simple et clair, pas trop de détail)

http://www.commentcamarche.com/php/phpinst.php3
- Installation Apache et php

http://www.openssl.org/docs/.
- Téléchargement des dernières versions.
- Documentation sur les commandes OpenSSL et sur la syntaxe à utiliser.

http://guill.net/reseaux/Ssl.html
- Explication sur le protocole SSL

9 Conclusion

Nous arrivons au terme d'un travail qui m'a apporté de nouvelles connaissances dans un domaine qui, deviendra un pilier indispensable des systèmes d'information de demain: la sécurité.

Ce projet a débuté par l'installation et la prise en main de l'autorité de certification Keon. Mes connaissances de ce système étaient pratiquement nulles, elles se sont considérablement étendues pendant le déroulement de ce travail.

Après un rappel de la théorie sur les infrastructures à clé publique, nous nous sommes intéressés à l'outil OpenSSL et à ces vertus pédagogiques pour les étudiants. Il permet une multitude d'opération sur la sécurité (cryptage, certification …) et reste un formidable outil de visualisation des informations contenues dans différents fichiers (certificat PEM, DER, CRL….).

Puis, nous avons étudié différents protocoles et formats utilisés lors d'une phase d'inscription (*enrollment*) qui consiste à se faire délivrer un certificat numérique par une autorité de certification. Les plus utilisés sont sans doute les formats PKCS#10 et PKCS#7 avec SSL. D'autre protocoles plus complets et permettant un sécurisation accrue ont été étudiés comme CMP et CRMF. Nous avons analysé les différentes vulnérabilités, les sécurités de ces formats. Nous avons aussi visualisés les formats PKCS#7 et 10 à l'analyseur de protocole entre un client et l'autorité de certification.

Enfin, nous avons configuré un serveur Apache avec le module « mod_ssl ». Nous avons configuré l'authentification client et serveur, testé les CRLs, codé la fonctionnalité OCSP. Pour cela nous avons utilisé la commande OCSP d'OpenSSL.

Comme pour tout document, des retouches ont été nécessaires afin d'améliorer l'aspect didactique de celui-ci.

Ainsi nous pensons avoir rempli le cahier des charges. Les quelques points divergents entre l'énoncé du travail à fournir et sa réalisation ont été discutés et approuvés par le professeur de diplôme responsable de ce projet.