Certificats X509
et
Infrastruture de Gestion de Clés

Serge Aumont - CRU
Claude Gross – CNRS/UREC
Philippe Leca – CNRS/UREC

-

Novembre 2001

Sommaire

1. Rappels sur les services de bases de la cryptologie

1.1 Définitions

Les termes de chiffrer, déchiffrer et décrypter sont utilisés dans cette article avec des sens précis. On entend par **chiffrer**, l'action de transformer *à l'aide d'une convention secrète*, appelée clé, des informations claires en informations inintelligibles par des tiers n'ayant pas la connaissance de la clé, **déchiffrer** consiste à retrouver les informations claires, à partir des informations chiffrées *en utilisant la convention secrète* de chiffrement et enfin **décrypter** consiste à retrouver l'information intelligible, à partir de l'information chiffrée *sans utiliser la convention secrète* de chiffrement.
Les termes de crypter ou encrypter n'ont pas de sens clairement défini.

La cryptologie permet de rendre un certain nombre de services de base en sécurité :

- l'**authentification** est l'assurance de l'identité d'un objet, généralement une personne, mais cela peut aussi s'appliquer à un serveur, une application, … Dans la vie courante, la présentation de la carte nationale d'identité et la signature manuelle assurent un service d'authentification ;
- l'**intégrité** d'un objet (document, fichier, message …) est la garantie que cet objet n'a pas été modifié par une autre personne que son auteur. Sur une feuille de papier toute modification est visible d'un simple coup d'œil. Sur un document électronique (courrier électronique, fichier Word, …) non sécurisé, cette détection est impossible ;
- la **confidentialité** est l'assurance qu'un document ne sera pas lu par un tiers qui n'en a pas le droit lors de la transmission de ce document ou lorsqu'il est archivé. Les documents papiers qui doivent rester secrets sont généralement stockés dans des coffres et sont transportés sous plis cachetés. Pour les documents électroniques, on utilisera le chiffrement

Un quatrième service de sécurité est appelée « **non répudiation ».** Comme ce terme l'indique, le but est que l'émetteur d'un message ne puisse pas nier l'avoir envoyé et le récepteur l'avoir reçu. Les transactions commerciales ont absolument besoin de cette fonction. Le reçu que l'on signe au livreur, la lettre recommandée sont des mécanismes de non répudiation.
Les certificats permettent d'assurer ce service. Dans la communauté enseignement-recherche, cette fonction n'est pas primordiale si ce n'est pour certains actes administratifs (votes, notations, transferts de crédits, …).

1.2 Notions de chiffrement

Pour assurer la **confidentialité** d'un document électronique, on chiffre le texte du document. Cette opération consiste à appliquer une fonction mathématique (en fait c'est un ensemble de fonctions)
avec des caractéristiques très particulières sur le texte. Cette fonction utilise une variable, la **clé de chiffrement**, qui est une suite de bits quelconque. Une fois le texte chiffré, il est illisible. Pour obtenir la version lisible, il faut le déchiffrer, c'est à dire appliquer une autre fonction mathématique, compatible avec la première, avec une autre variable, la **clé de déchiffrement**.
Ces 2 fonctions mathématiques sont appelées **algorithmes de chiffrement**. La valeur de la clé de déchiffrement dépend évidemment de la valeur de la clé de chiffrement et seul le possesseur de la clé de déchiffrement peut déchiffrer le texte. Lorsque l'on désire transmettre un document confidentiel à un correspondant à travers le réseau, on chiffre le document sur son poste de travail avec une clé de chiffrement et on envoie la version chiffrée. Le destinataire déchiffre le document sur son poste de travail avec la clé de déchiffrement, qu'il est le seul à connaître. Si une troisième personne intercepte le texte durant le transfert, il ne pourra pas le déchiffrer car il ne connaîtra pas la valeur de la clé de déchiffrement.
Il faut noter que les algorithmes de chiffrement, c'est à dire les formules mathématiques, sont publics et ont fait l'objet de standardisation. C'est le secret de certaines clés qui permet à ces algorithmes d'assurer le service de confidentialité.

Voici les étapes de la transmission d'un texte confidentiel de Alice à Bob

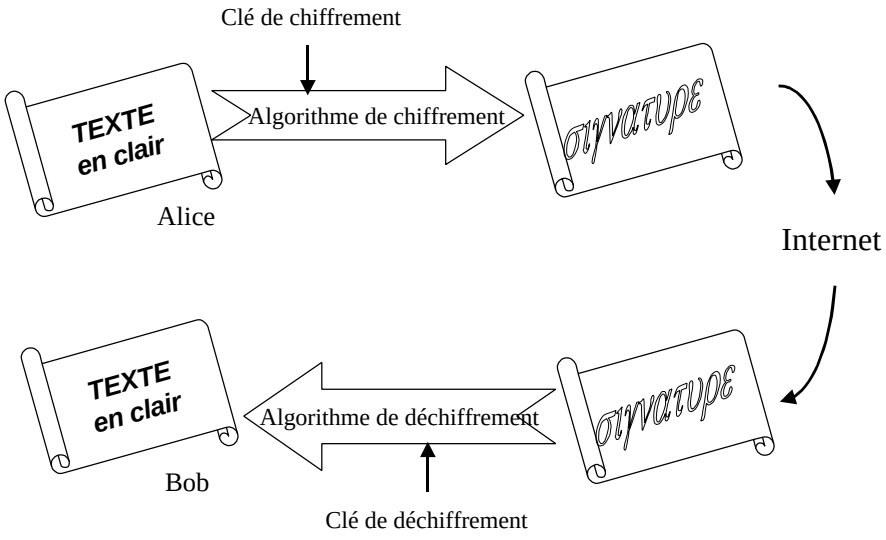

Figure 1 : chiffrement

Concrètement, il y a deux types de chiffrement possible. Le chiffrement à clés secrètes et le chiffrement à clés publiques et secrètes.

1.2.1 Chiffrement à clés symétriques (ou chiffrement à clés secrètes)

Avec cette méthode, **la clé de chiffrement est la même que la clé de déchiffrement**. De ce fait, pour que le texte chiffré ne soit lisible que par le destinataire, la valeur de cette clé (unique) doit être un secret partagé uniquement entre l'émetteur et le destinataire. Ceci explique le qualificatif de « clé secrète ».
Il existe plusieurs algorithmes qui fonctionnent sur ce principe :

- **DES** (Digital Encryption Standard) est l'algorithme à clé symétrique historiquement le plus connu. Il utilise des clés (de chiffrement et de déchiffrement) de 56 bits et chiffre par blocs de 64 bits. La version améliorée est le **Triple DES** qui consiste à chiffrer successivement trois fois de suite avec deux ou trois clés différentes. Cet algorithme tend à devenir l'un des moins performant.
- **IDEA** (**(International Data Encryption Algorithm)** est un algorithme assez récent (1992) qui effectue un chiffrement par blocs de 64 bits avec des clés de 128 bits. Son fonctionnement est proche de DES
- **AES** (Advanced Encryption Standard) : en vue de remplacer l'utilisation de DES, le NIST (National Institute of Standards and Technologie) a lancé un concours pour obtenir un algorithme à clé symétrique robuste aux différentes attaques, avec une utilisation libre de droits et une facilité d'implémentation. En octobre 2000 c'est l'algorithme belge Rijndael qui a été désigné vainqueur du concours. Il devrait devenir un des algorithmes les plus utilisés dans les applications. Les longueurs de clés prévues sont de 128, 192 et 256 bits.
- On peut aussi citer **RC2, RC4, RC5** (Rivest's Code #4, #5 » 1987) ou **BLOWFISH** (1993)

Les algorithmes utilisent des fonctions mathématiques « simples ». L'avantage est que **les opérations de chiffrement et de déchiffrement sont rapides** à exécuter sur des ordinateurs classiques.

Les algorithmes symétriques ne permettent pas de résoudre deux types de problèmes :

1. Par quel canal faire s'échanger les clés symétriques ? Ce point est critique pour des applications de l'internet.
2. Explosion du nombre de clés à gérer avec le nombre de correspondants.

Ce dernier point est mis en évidence en calculant le nombre de clés secrète pour qu'au sein d'un groupe de n personnes toutes les communications chiffrées soient possibles. Ce petit calcul montre qu'il faut disposer de n(n-1) / 2 clés ! A l'échelle de l'internet, ce problème n'a pas de solution.

1.2.2 Chiffrement à clés asymétriques (ou chiffrement à clés publiques et secrètes)

Le principe des algorithmes de chiffrement à clés asymétriques a été introduit en 1976 par Diffie et Hellman. Ils ont été conçus pour utiliser des clés qui possèdent 2 propriétés essentielles :

1. Les clés sont crées **par couple** souvent appelé bi-clé. Ce bi-clé est tel que tout texte chiffré par l'une quelconque des deux clés n'est déchiffrable que par l'autre clé. C'est cette caractéristique qui a donné leur nom aux algorithmes de chiffrement asymétriqueL
2. La connaissance d'une des deux clés ne permet pas de déduire l'autre.

En pratique, chaque protagoniste dispose d'un bi-clé (au moins un). On décide arbitrairement, pour chaque bi-clé, que l'une des clés est publique et l'autre privée. L'usage de cette dernière clé est strictement réservé au titulaire du bi-clé et elle est protégée avec soin. Elle peut par exemple, être implantée sur une carte à puce. A l'inverse, la clé publique elle, pourra être publiée dans un annuaire LDAP.

Quand Alice envoie un message chiffré avec un algorithme asymétrique à Bob, son outil chiffre le texte du message avec la clé publique de Bob (l'accès à celle-ci se fait soit par un cache local des clés publiques des correspondants d'Alice, soit par un accès à un annuaire). Ce texte ne peut alors être déchiffré qu'avec la clé privée de Bob. Ainsi le message peut transiter via un réseau réputé non sûr sans risque d'être déchiffré par un tiers. A la réception par Bob, le texte est déchiffré avec la clé privée de celui-ci.

Voici par exemple, l'émission par Alice d'un texte chiffré adressé à Bob avec un algorithme de chiffrement asymétrique.

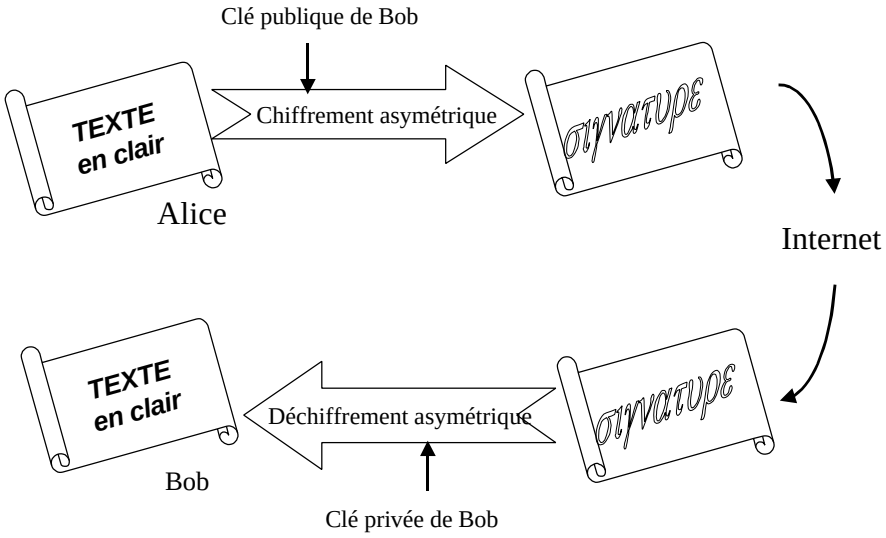

Figure 2 : chiffrement avec algorithme asymétrique

Réciproquement, quand Bob envoie un texte chiffré à Alice il le chiffre avec la clé publique d'Alice. Celle-ci le déchiffre avec sa clé privée.

RSA (rfc2437), du nom des 3 inventeurs Rivest, Shamir, Adleman, est l'algorithme de chiffrement à clés asymétriques le plus répandu. Il a été mis au point en 1976 et est devenu public (fin des droits de la Sté RSA) depuis Septembre 2000.
La robustesse du RSA est basée sur la non divulgation de la clé privée, sur la difficulté qu'il existe à factoriser des grands nombres et sur l'absence de méthodes mathématiques pour déduire la clé privée de la clé publique.
DSA est un autre algorithme à clés asymétriques.

Ce découplage entre clé publique et clé privée est très utile pour une utilisation « planétaire » du chiffrement. Alors que les algorithmes symétriques obligent à échanger un secret, la clé secrète, avec chaque interlocuteur, là il suffit d'avoir un annuaire qui permette de trouver la clé publique de chaque internaute et ce système peut fonctionner entre tous les internautes. Quand un utilisateur voudra envoyer un message chiffré à un correspondant, il consultera l'annuaire qui lui indiquera la clé publique de son correspondant. Avec cette clé, il chiffrera le message. Celui-ci ne pourra être déchiffré qu'avec la clé privée du correspondant, donc que par le correspondant.

Mais il reste un problème**.** Avec les algorithmes asymétriques, **le temps pour les opérations de chiffrement et de déchiffrement est long.** Ainsi sur un ordinateur courant, RSA (algorithme asymétrique) est de 100 à 1000 fois plus lent que le Triple DES (algorithme symétrique).

1.2.3 Utilisation des algorithmes asymétriques pour l'échange de clés d'algorithmes symétriques

Pour contourner les mauvaises performances des traitements des algorithmes à clés asymétriques, une solution consiste à utiliser les deux types de chiffrement (à clés symétriques et à clés asymétriques) pour effectuer la transmission d'un document chiffré.
Un des interlocuteurs (Alice) tire aléatoirement une clé à usage unique. Cette clé sera utilisée pour chiffrer le document avec un algorithme à symétrique. Cette clé est elle même chiffrée avec un algorithme asymétrique en utilisant la clé publique du destinataire : Bob.

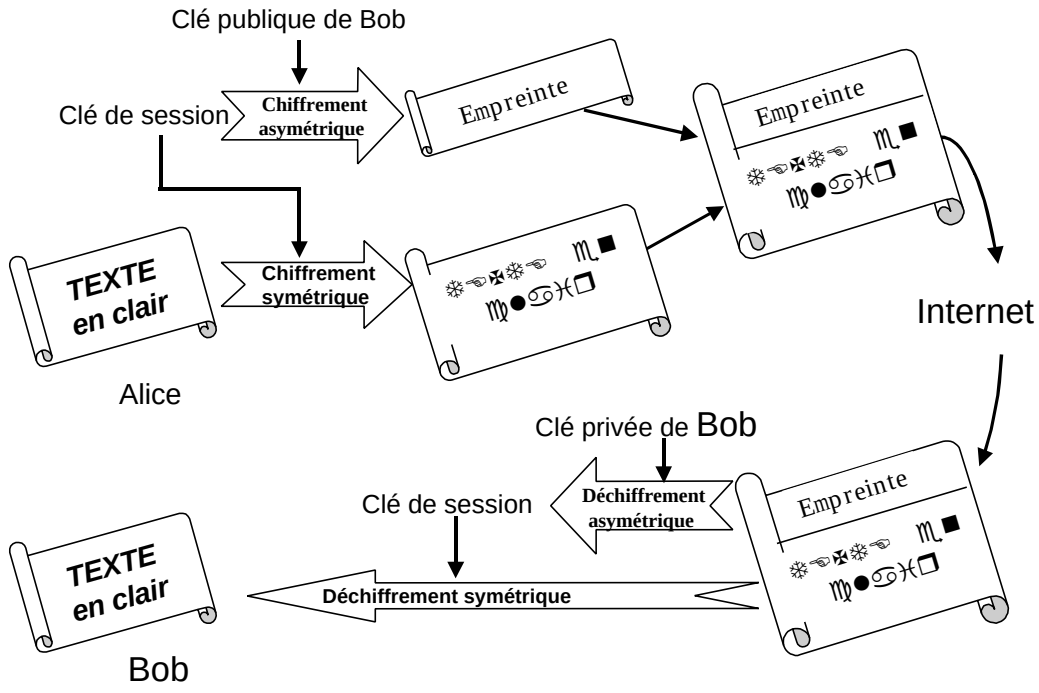

Figure 3 : chiffrement avec clé de session

Ainsi le nombre de bits chiffrés avec un algorithme asymétrique (la longueur de la clé symétrique) est très petite par rapport à la longueur du message à chiffrer, le facteur de coût de l'algorithme asymétrique est contrôlé. En limitant l'usage du chiffrement asymétrique à l'échange de clés à usage unique, on résout les deux difficultés majeures des algorithmes de chiffrement symétrique : l'absence d'un canal sûr pour l'échange des clés secrètes et la complexité en n^2 du nombre des clés.

1.2.4 Longueur de clés

La sécurité d'un système de cryptologie repose sur plusieurs facteurs. L'un d'entre eux est la difficulté de décrypter (Déchiffrer sans posséder la clé de déchiffrement) l'information. Plus la clé utilisée sera longue plus le décryptage sera difficile, la limite étant la puissance actuelle des calculateurs, avec un algorithme de chiffrement solide (bon mathématiquement) et une implémentation correcte (sans bogue).
L'augmentation de la puissance de calcul des ordinateurs impose l'augmentation de la taille des clés; la législation doit elle aussi s'adapter (avant 1999, l'utilisation du chiffrement en France était libre pour des clés allant jusqu'à 40 bits. Après 1999, la taille des clés a été autorisée jusqu'à 128 bits).

La longueur des clés des algorithmes à clés secrètes (clés couramment entre 40 et 256 bits) et celle des algorithmes à clés publiques (clés couramment entre 512 et 2048 bits), n'est pas comparable.
Pour les algorithmes à clés secrètes, la référence est l'attaque par force brute (moyenne 2^{n-1} essais pour retrouver la clé) alors que pour les algorithmes à clés publiques, la robustesse est basée sur la difficulté mathématique à résoudre le problème sur lequel est basé l'algorithme (l'attaque par force brute n'a guère de sens). Dans ce dernier cas, une avancée théorique en mathématique pourrait rendre le système à clés publiques très vulnérable (un groupe de chercheur a récemment réussi à factoriser un nombre de 512 bits).

On considère actuellement que pour une communication chiffrée il faut au moins utiliser une longueur de clé de 128 bits pour le chiffrement à clés symétriques et une longueur de clé de 1024 bits pour le chiffrement à clés asymétriques.

1.3 Signature électronique

Le chiffrement permet de rendre les services de confidentialité. La signature électronique va permettre de garantir l'authentification de l'origine d'un document ou d'un message électronique et son intégrité. Ceci implique un certain nombre de propriétés :

- une signature ne peut être falsifiée,
- une signature donnée n'est pas réutilisable par un autre document
- la modification d'un document signé altère la signature de ce document
- une signature ne peut être reniée

Pour générer une signature électronique il faut dans un premier temps utiliser une **fonction de hachage**. C'est une fonction mathématique qui, à partir d'un texte, génère un nombre caractéristique de ce texte. Ce nombre est appelé **empreinte** (fingerprint) parce que toute modification du texte entraîne la modification de son empreinte. L'empreinte n'est pas une compression puisqu'on en peut retrouver le fichier d'origine à partir de son empreinte beaucoup plus petite. S'il existe des « faux positif », c'est à dire des fichiers différents ayant la même empreinte, il n'existe pas de méthode pour fabriquer un fichier ayant la même empreinte qu'un autre fichier.

MD5 (MD pour Message Digest, rfc1321) est une fonction de hachage très répandue, elle crée une empreinte de 128 bits. **SHA** (Secure Hash Algorithm), autre fonction, crée des empreintes de 160 bits. Ce sont les deux algorithmes les plus répandus actuellement.

Le principe utilisé pour la signature d'un document est de chiffrer une empreinte de celui-ci avec la clé privée de son auteur. Tout un chacun peut déchiffrer cette l'empreinte avec la clef publique de l'auteur et recalculer l'empreinte sur le document lui même. La bonne correspondance entre ces deux empreintes constitue la preuve que :

1. le document n'a pas été modifié (puisque l'empreinte d'origine est conforme à l'empreinte du document reçu
2. l'empreinte d'origine a bien été apposée par le titulaire de la clef privée associée à la clef publique utilisée pour vérifier la signature.

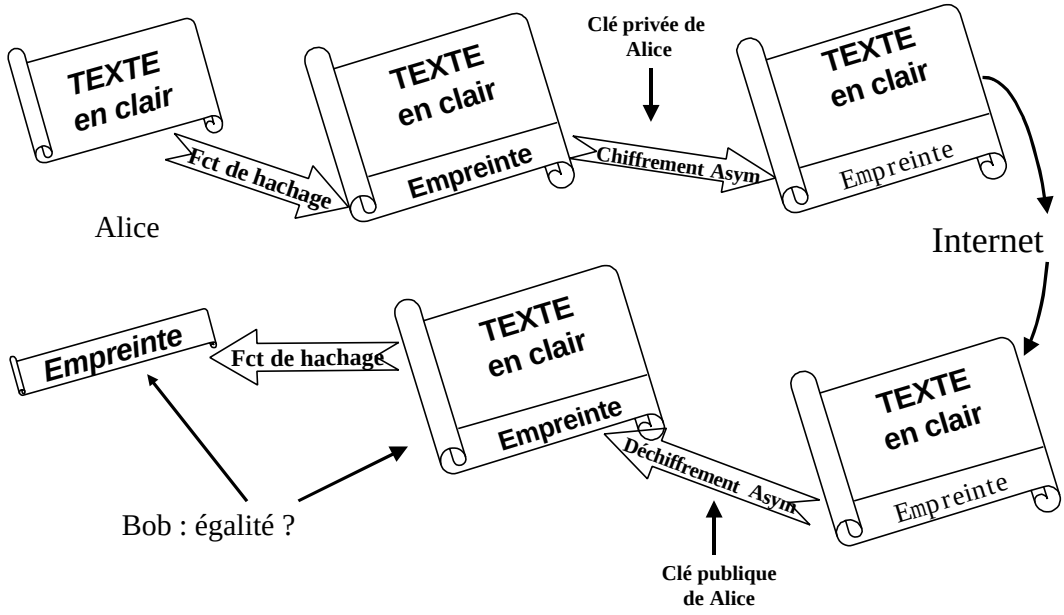

Figure 4 : signature

1.3.1 Problématique de la gestions des clés

Dans un système utilisant un chiffrement à clés symétriques, il y aura beaucoup de clés à gérer suivant le nombre d'entités concernées (en fonction du carré du nombre de liaisons à établir : n(n-1)/2). Si on veut ajouter une nouvelle liaison (un nouvel utilisateur, un nouveau service …), il faudra générer n clés et les distribuer. Cette gestion des clés est inadaptée sur une échelle importante : les problèmes d'administration deviennent rapidement insurmontables

Dans un système utilisant un chiffrement à clés asymétriques, un utilisateur a un couple de clés. La clé privée reste toujours avec l'utilisateur et la clé publique est publiée dans un annuaire publique. Un nouvel utilisateur aura uniquement besoin de son couple de clés et de publier sa clé publique dans l'annuaire pour pouvoir communiquer avec l'ensemble des autres entités.
Ce type de système introduit un nouveau problème : l'utilisation d'un couple de clés entraîne la nécessité de publication, en toute confiance, de la clé publique. Cette publication doit offrir l'assurance que :

- la clé est bien celle appartenant à la personne avec qui les échanges sont envisagés ;
- le possesseur de cette clé est « digne de confiance » ;
- la clé est toujours valide.

Cette notion de confiance est résolue avec les **certificats** et les **autorités de certification**.

2. Certificats et Autorité de Certification

2.1 Qu'est-ce qu'un certificat?

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

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

On ne parlera dans cette présentation que des certificats qui s'appuient sur un protocole normalisé X509 (ITU-T X.509 international standard V3 - 1996) *(RFC2459).* Il existe des applications ou des systèmes de cryptologie qui ne s'appuie pas sur les certificats X509 (pgp par exemple). Toutes les applications abordées dans les chapitre suivants utilise ce type de certificat.

Le certificat est signé (au sens signature électronique) : on effectue une empreinte (ou un condensé) du certificat à l'aide d'algorithme (MD5 par exemple), et on chiffre l'empreinte obtenue.
Le chiffrement s'effectue avec la clé privé de l'autorité de certification qui possède elle même son propre certificat.

2.1.1 Contenu d'un certificat

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

Data	Version
	Serial Number
	Signature Algorithm
	Issuer
	Validity
	Subject
	Subject Public Key Info
	X509v3 extensions Extension1: [critical] Valeur extension1 Extension2: [critical] Valeur extension2 … Extension*n*: [critical] Valeur extension*n*
Signature	Signature Algorithm
	Signature

Version	Indique à quelle version de X.509 correspond ce certificat.
Serial number	Numéro de série du certificat (propre à chaque autorité de certification).
Signature Algorithm	Type de signature utilisée.
Issuer	Distinguished Name (Subject) de l'autorité de certification qui a émis ce certificat.
Validity	Période de validité.
Subject	Distinguished Name du propriétaire de ce certificat.
Subject public key info	Infos sur la clef publique de ce certificat.
X509v3 Extensions	Extensions génériques optionnelles, introduites avec la version 3 de X.509.
Signature	Signature numérique de l'AC sur l'ensemble des champs précédents.

Le certificat associe à la clé publique des informations spécifiques à l'entité (physique ou morale) à laquelle elle se rapporte (informations s'ajoutant aux informations de bases que sont : numéro de

version, numéro de série, algorithme de signature, période de validité… contenues dans un certificat X509).

Les extensions introduites dans la norme X509v3 permettent de spécifier un certain nombre d'informations en fonction de l'usage prévu d'un certificat. Les extensions ci-dessous ne sont pas exhaustives :

- *X509v3 Basic Constraint* : indique s'il s'agit du certificat d'une Autorité de Certification ou non, c'est à dire permettant d'émettre des certificats
- *X509v3 Key Usage* : Donne une ou plusieurs fonctions de sécurité auxquelles la clé publique est destinée. Ce champ permet de spécifier plusieurs services de sécurité

« Key Usage »	Usage de la clé	Combinaisons valides				
		Fonction de signature	Fonction de certification (signature de certificats / CRL).	Fonction de chiffrement de clé	Fonction de chiffrement de données	Fonction de négociation de clé
digitalSignature	Signature numérique	X				
nonRepudiation	Non répudiation	X				
keyEncipherment	Chiffrement de clé			X		
dataEncipherment	Chiffrement de données				X	
keyAgreement	Négociation de clés					X
keyCertSign	Clé de signature de certificat		X			
cRLSign	Signature de CRL		X			
encipherOnly	Chiffrement seul					
decipherOnly	Déchiffrement seul					

- *X509v3 subjectAltName*: Ce champ contient un ou plusieurs noms alternatifs pour le porteur de certificat, exprimé sous diverses formes possibles.
- *X509v3 issuerAltName*: Ce champ contient un ou plusieurs noms alternatifs pour l'AC qui a généré ce certificat, exprimé sous diverses formes possibles.
- *X509v3 CRL Distribution Points* : adresse de la CRL permettant de connaitre le status de ce certificat

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

Avant que la norme X509v3 n'ai été créée, Netscape a introduit plusieurs extensions permettant de limiter l'usage d'un certificat :

- *netscape-cert-type*: spécifie le type de certificat :
  - *SSL client* – certificat pouvant être utilisé pour ouvrir une connexion SSL en tant que client
  - *SSL server* - certificat pouvant être utilisé pour un serveur SSL
  - *S/MIME* – certificat pouvant être utilisé pour la messagerie S/MIME
  - *Object Signing* - certificat pouvant être utilisé pour signé du code (applets java, …)
  - *SSL CA* - certificat d'AC pouvant être utilisé pour émettre des certificats pour SSL
  - *S/MIME CA* - certificat d'AC pouvant être utilisé pour émettre des certificats S/MIME
  - *Object Signing CA* - certificat d'AC pouvant être utilisé pour émettre de certificats pour la signature de code

- *netscape-base-url*:
  URL de base à ajouter à toutes les URLs relatives présentes dans le certificat
- *netscape-revocation-url*:
  URL d'une ressource permettant de savoir si un certificat est valide ou non
- *netscape-ca-revocation-url*:
  URL d'une ressource permettant de connaître le statut de tout certificat issu par l'AC ayant généré ce certificat. Cette extension n'est possible que dans le certificat d'une AC.
- *netscape-cert-renewal-url*:
  URL d'un formulaire permettant de demander le renouvellement de ce
- *netscape-ca-policy-url*:
  URL d'un document décrivant les règles et procédure (policies) qui ont été utilisées pour la création de ce certificat.
- *netscape-comment*:
  Commentaire qui sera affiché lors de la visualisation du certificat.

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

Exemple de certificat :

```
Certificate:
     Data:
         Version: 3 (0x2)
         Serial Number: 4 (0x4)
         Signature Algorithm: md5WithRSAEncryption
         Issuer: C=FR, O=CNRS, CN=CNRS-Standard
         Validity
             Not Before: May  3 09:00:43 2001 GMT
             Not After : May  3 09:00:43 2002 GMT
         Subject: Email=Philippe.Leca@urec.cnrs.fr, CN=Philippe Leca, OU=UPS836, O=CNRS, C=FR
         Subject Public Key Info:
             Public Key Algorithm: rsaEncryption
             RSA Public Key: (1024 bit)
                 Modulus (1024 bit):
                     00:f8:c4:f7:d9:0a:51:ba:b5:45:8d:f5:2c:f2:c1:
                     21:41:19:cb:e4:24:5f:c9:54:6f:d3:bd:11:e6:ff:
                     b6:e9:d6:78:87:90:89:ad:93:60:46:28:80:4f:85:
                     1a:94:58:6a:18:57:47:d6:51:b6:79:2e:8f:ff:5e:
                     6e:a8:54:bb:f7:b2:9f:d3:ea:66:51:89:2c:e7:cd:
                     e9:c0:91:ff:d4:8d:b2:30:23:36:09:9b:09:b9:f9:
                     99:d0:3e:fe:38:ac:ed:66:62:33:93:1f:d5:72:f1:
                     ad:e5:c1:93:4f:f8:8f:fa:bd:7d:ff:c4:d1:43:c9:
                     45:a0:96:74:14:73:ee:36:73
                 Exponent: 65537 (0x10001)
         X509v3 extensions:
             X509v3 Basic Constraints:
                 CA:FALSE
             Netscape Cert Type:
                 SSL Client, S/MIME
             X509v3 Key Usage:
                 Digital Signature, Non Repudiation, Key Encipherment
             Netscape Comment:
                 Certificat CNRS-Standard.
             X509v3 Subject Key Identifier:
                 CE:5D:A2:36:19:86:F5:E1:D7:9F:EE:41:26:1C:D5:93:3F:12:C8:80
             X509v3 Authority Key Identifier:
                 keyid:67:59:A5:E5:07:74:49:03:EF:05:CF:CC:2E:A4:18:D5:10:C8:9E:3C
                 DirName:/C=FR/O=CNRS/CN=CNRS
                 serial:02

             X509v3 CRL Distribution Points:
                 URI:http://igc.services.cnrs.fr/cgi-bin/loadcrl?CA=CNRS-Standard&format=DER

             Netscape Revocation Url:
                 http://igc.services.cnrs.fr/cgi-bin/loadcrl?CA=CNRS-Standard&format=DER
             Netscape Renewal Url:
                 https://igc.services.cnrs.fr/cgi-bin/renew?CA=CNRS-Standard
     Signature Algorithm: md5WithRSAEncryption
         af:c4:87:ad:75:bc:b4:79:f9:c7:67:cf:eb:4a:9c:bf:64:e3:
         50:7f:97:0b:50:77:1e:1c:8e:d8:f2:c8:a0:c8:30:24:84:6c:
         2c:d9:0e:94:c3:d0:41:41:e9:6d:c7:9e:17:1c:7e:7d:f3:17:
         89:2e:64:04:47:e2:e6:52:f1:87:93:87:3a:fe:14:cd:a5:56:
         7e:a4:26:86:d2:4d:92:72:eb:99:b2:c1:cf:67:f4:a2:f0:47:
         26:36:b2:3d:09:5a:4c:46:47:30:42:74:80:18:27:06:99:24:
         d5:74:aa:e7:e3:36:cd:c9:97:54:67:14:a5:4a:c3:b9:c9:23:
         c7:b8:bb:30:9b:7f:44:37:12:85:3b:0d:88:d8:07:9f:c0:e4:
         b3:4b:a5:59:b1:a5:23:4a:7a:09:26:65:ac:b9:5e:7f:20:6c:
         ae:96:b1:0b:72:43:fe:3b:1f:76:59:b9:37:d0:48:6b:f6:30:
         79:85:90:53:5c:24:db:4c:70:a2:67:b6:96:6f:93:09:03:52:
         aa:84:57:43:88:c8:bb:1f:24:a2:15:65:b2:6e:96:ad:a8:94:
         34:ad:7a:1e:94:2b:4d:91:fb:e6:b9:b0:5f:50:ac:33:9b:98:
         8a:6b:2c:e7:e2:1d:67:dd:6d:dc:15:f3:04:62:aa:ba:63:4d:
         4a:03:5e:ea
```

2.1.2 Vérification d'un certificat

La vérification s'effectue avec la clé publique de l'autorité de certification. Toute personne voulant vérifier un certificat délivré par une autorité de certification doit connaître la clé publique de cette autorité de certification.
Sur la figure ci-dessous, on teste la validité du certificat d'Alice délivré par l'Autorité de certification CNRS

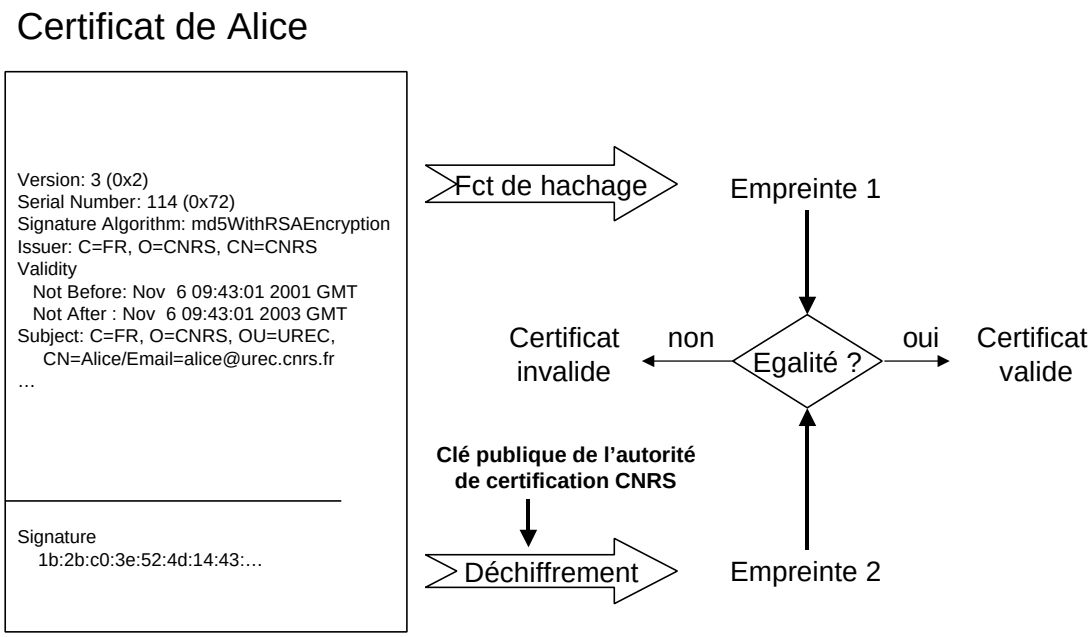

Figure 5 : Vérification de certificat

2.2 Qu'est-ce qu'une Autorité de Certification?

Une Autorité de Certification est une organisation qui délivre des certificats électroniques à une population.

Une AC possède elle-même un certificat (autosigné ou délivré par une autre AC) et utilise sa clé privée pour créer les certificats qu'elle délivre.

N'importe qui peut se déclarer Autorité de Certification. Une AC peut être organisationnelle (exemple : CNRS), spécifique à un corps de métiers (exemple : notaires) ou encore institutionnelle.

Selon le crédit accordé à l'AC, les certificats délivrés auront un champ d'applications plus ou moins importants :

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

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

Exemple de certificat de CA

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=FR, O=CNRS, CN=CNRS
        Validity
            Not Before: Apr 27 05:46:49 2001 GMT
            Not After : Apr 25 05:46:49 2011 GMT
        Subject: C=FR, O=CNRS, CN=CNRS
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:dc:e1:1e:21:3d:06:8b:ea:bd:5e:b4:88:db:0f:
                    93:97:b4:6d:07:3d:86:62:00:2d:ca:ff:b5:4a:8e:
                    e7:56:a4:8f:61:2c:f1:a0:2a:ab:f6:2a:dd:7c:2c:
                    bf:ef:75:55:0b:ac:09:4e:e7:4e:61:c0:e7:0c:f0:
                    90:15:45:12:02:c2:8c:eb:c3:12:64:e2:63:10:18:
                    2e:cb:07:31:d9:81:e5:dc:29:82:9b:31:56:e2:81:
                    1e:8a:6f:a7:e8:a9:58:11:44:56:83:5d:b3:4e:78:
                    70:2d:df:b6:fd:72:81:45:d5:f1:ee:4d:ce:ef:be:
                    d5:3d:0c:90:20:45:9a:09:80:af:0f:4c:da:20:0e:
                    80:bf:3a:b3:eb:27:80:c0:b9:0f:c0:a1:4e:40:dc:
                    3a:fd:6a:2a:bf:40:d5:2c:71:80:f9:f8:ba:6b:e4:
                    ea:2a:00:ab:2f:be:9a:f0:a7:76:6d:98:29:9c:0f:
                    2f:f0:42:f2:18:97:5b:c9:f6:cc:19:5f:ba:c2:be:
                    12:d2:5c:b0:90:94:c0:b7:cb:06:04:ef:8f:30:ed:
                    32:2d:7a:4a:f7:93:bb:a0:09:a4:b4:ee:33:cb:d0:
                    83:9b:b5:b5:b3:90:de:8e:90:1e:59:9c:20:d5:4b:
                    1e:ed:d7:4c:4f:86:fa:1c:3a:2a:a1:e9:ac:05:a0:
                    9d:bf
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Subject Key Identifier:
                67:59:A5:E5:07:74:49:03:EF:05:CF:CC:2E:A4:18:D5:10:C8:9E:3C
            X509v3 Authority Key Identifier:
                keyid:56:EB:68:B9:D2:5C:7E:98:B5:A5:53:C3:91:6F:63:58:C4:F9:6B:B7
                DirName:/C=FR/O=CNRS/CN=CNRS
                serial:00

            X509v3 Key Usage:
                Certificate Sign, CRL Sign
    Signature Algorithm: md5WithRSAEncryption
        06:03:47:83:72:45:90:c2:4e:e1:21:d7:ab:17:a9:01:55:06:
        ca:40:6d:55:a2:1d:5e:eb:e2:14:23:59:e4:09:e2:90:f6:3c:
        8d:36:06:0f:4b:a7:26:23:65:c2:ea:06:9a:72:bb:b8:8c:cb:
        8a:5f:ef:79:36:25:7e:00:d7:f3:06:94:fb:83:44:29:26:37:
        c7:ee:e9:87:ce:6c:86:80:1b:71:3d:d2:62:af:f6:cd:62:6c:
        53:0f:e6:7a:93:00:8c:7b:2e:33:e0:41:1d:aa:be:65:98:76:
        f1:95:07:74:b3:e6:3f:53:75:d5:4b:06:36:4b:29:c4:f6:dc:
        8e:13:80:40:10:73:82:ad:15:7b:04:71:50:b5:37:33:f2:c8:
        64:bb:a1:10:7e:36:c6:ad:af:6f:70:52:a6:d1:ae:cc:cc:ba:
        b0:e8:59:12:8f:62:0d:ad:03:dd:4b:2a:e8:89:39:88:51:2f:
        ed:61:e8:b7:30:87:db:27:55:6d:66:87:a3:51:09:80:61:71:
        51:05:be:13:1d:d9:41:30:fc:75:5f:0a:96:9b:18:ff:be:90:
        81:b4:13:c0:72:11:08:fd:6a:9a:6a:07:bd:f4:83:2c:b4:60:
        36:64:07:fa:3d:6a:a7:b0:90:04:76:83:dd:33:cb:34:e2:17:
        98:04:0b:a1
```

2.3 Certification croisée et certification hiérarchique

Une AC peut cautionner une autre AC en créant un certificat en signant sa clé publique. On peut ainsi mettre en place des relations de confiance hiérarchiques ou croisées entre ACs.

Dans le cas d'une relation hiérarchique, une AC dite racine délivre un certificat à une ou plusieurs autres ACs, qui elles mêmes peuvent délivrer un certificat à d'autres ACs et ainsi de suite.

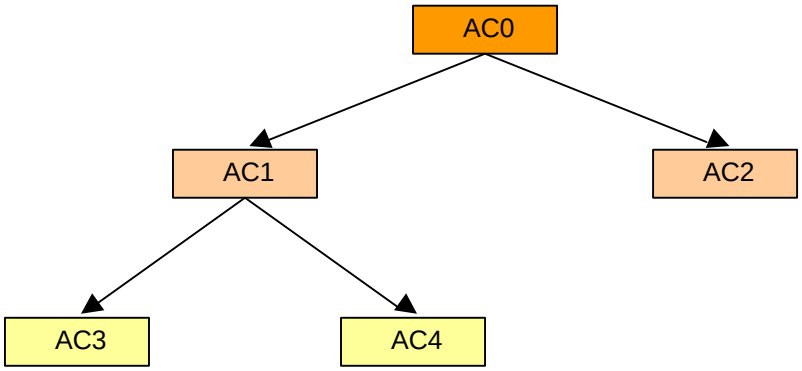

Figure 6 : Hiérarchie d'ACs

Au sommet de ces arborescences, on trouve les ACs racines dont le certificat est signé avec leur propre clé privée (certificat autosigné).
Dans le cas de la figure ci-dessous, pour valider un certificat issu de l'AC AC4, il faut les certificats des ACs AC0, AC1 et AC4. On appelle cela une **chaîne de certification**.
Une chaîne de certification est l'ensemble des Certificats nécessaires pour valider la généalogie d'un certificat d'un porteur de certificat.
Dans une architecture horizontale simple, la chaîne se compose du Certificat de l'Autorité de Certification qui a émis le certificat et de celui du Porteur de Certificat.
Dans une architecture arborescente comme ci-dessus, c'est l'ensemble des certificats formant le chemin entre le certificat à vérifier et celui de l'AC racine.

En général dans ce type de schéma, seuls les ACs "feuilles" délivrent des certificats à des personnes ou à des services.
La confiance accordée à une AC est héritée par toutes ses ACs "filles". Ainsi si l'on décide de faire confiance à une AC racine, cette confiance sera également accordée à l'ensemble des ACs appartenant à la même hiérarchie.

Les relations de confiance croisées sont utiles lorsqu'un organisme O1 et O2 ayant chacun leur propre AC, AC1 et AC2, n'appartenant pas à une même arborescence, veulent se faire confiance. Dans ce cas, AC1 et AC2 peuvent chacun créer un certificat en signant la clé publique de l'autre. Ainsi un utilisateur de O1 pourra vérifier le certificat d'une personne de O2.

3. Infrastructure de Gestion de clés (IGC)

On parle aussi d'"Infrastructure à clés publiques" (**ICP**) ou de "Private Key Infrastructure" (**PKI**).

Un certificat électronique peut être comparé à une carte d'identité.
Dans le cas d'une carte d'identité, la personne fera sa demande à la mairie de sa commune, qui procédera à la vérification des informations fournies. La demande est alors transmise à la préfecture qui procédera à l'émission de la carte d'identité.
Dans le cas des certificats électroniques, l'AC peut être comparée à la préfecture (ou l'Etat) qui assume la responsabilité des pièces d'identité qu'elle délivre.
L'ensemble des procédures (demande, contrôle, émission,…) qui existe pour les cartes d'identité existe de la même façon pour les certificats électroniques. Elles sont mises en œuvre à travers une infrastructure qui est l'IGC.

L'IGC est constituée de l'ensemble des matériels, logiciels, personnes, règles et procédures nécessaire à une Autorité de Certification pour créer, gérer et distribuer des certificats X509.

Une IGC est donc une structure à la fois technique et administrative permettant une mise en place, lors de l'échange de clef, de ***relations de confiance entre des entités morales et/ou physiques et/ou logiques***.

Les fonctions principales d'une IGC sont :

- Émettre et révoquer des certificats
- Publier les certificats dans un annuaire
- Éventuellement, fournir un service de séquestre et de recouvrement des clés privées

Elle est constituée par :

- Une autorité de certification (**AC**)
- Une autorité d'enregistrement (**AE**)
- Un opérateur de certification (**OC**)
- Un annuaire de publication de certificats
- Un service de validation
- Éventuellement, un service de séquestre de clés

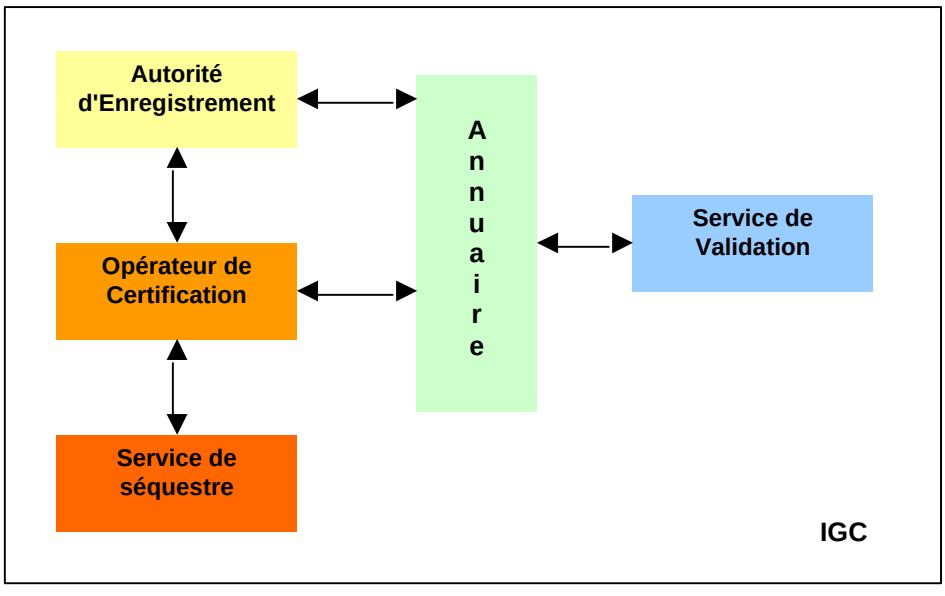

Figure 7 : Structure d'une IGC

3.1 Autorité d'Enregistrement

L'AE a pour tâche principale de recevoir et de traiter les demandes de création, de renouvellement et de révocation de certificats.

Pour cela, elle doit

- assurer le contrôle des données identifiant le demandeur de certificat,
- valider les demandes de révocation,
- assurer lors de la délivrance d'un nouveau certificat (sur date de péremption atteinte) un recouvrement des certificats afin d'assurer la continuité pour la fonctionnalité signature et/ou chiffrement.

Elle travaille en étroite collaboration avec l'opérateur de certification ; elle possède un bi-clef certifié pour s'authentifier et pour accomplir les tâches qui lui incombent.

L'AE comprend en général 2 composants :

- une interface permettant aux utilisateurs de faire leurs demandes,
- une interface de gestion réservés aux opérateurs de l'AE permettant de valider les demandes.

3.2 Opérateur de certification

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

3.3 Annuaire de publication

Les certificats émis par une IGC doivent être rendus publiques afin que les différents partenaires qui les utilisent puissent s'échanger leur clé publique. Pour cela, les certificats sont publiés dans un annuaire d'accès libre.
Cet annuaire peux également contenir le certificat de l'AC et les CRLs.
Des annuaires LDAP sont généralement utilisés pour cette fonction.

Scénario de demande de certificat :

La figure ci-dessous donne un exemple de scénario possible pour une demande de certificat. Dans ce scénario, le bi-clé est généré par l'utilisateur.

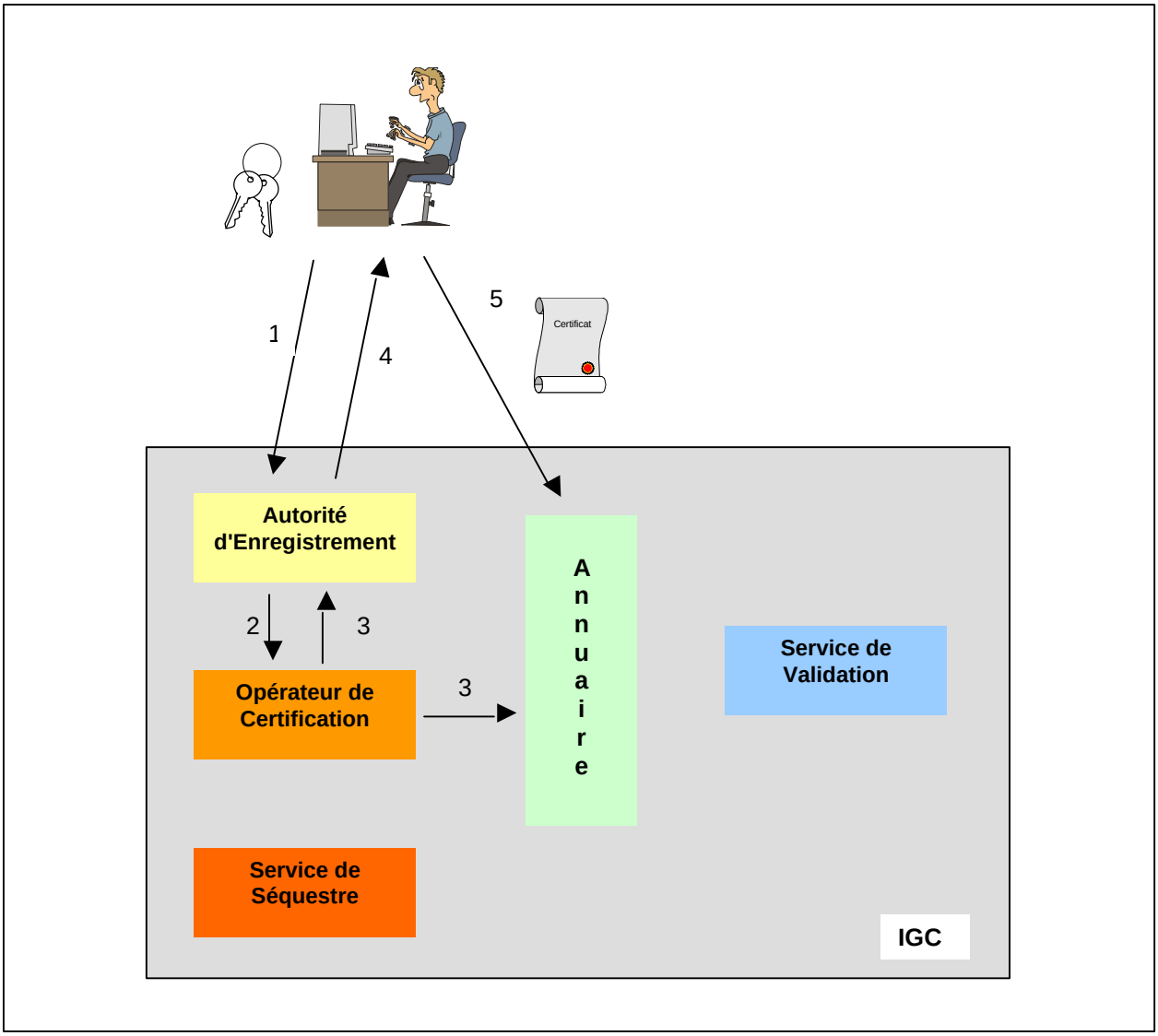

Figure 8 : scénario de demande de certificat

1. l'utilisateur fait sa demande à l'AE
2. l'AE vérifie les données d'identification et vérifie la possession de la clef privée. Si tout est OK, l'AE valide la requête qui est transférée à l'OC
3. l'OC vérifie la validité de la requête et génère le certificat. Le certificat est publié dans l'annuaire et transmis à l'AE
4. l'AE avertit l'utilisateur que son certificat est disponible
5. l'utilisateur récupère le certificat dans l'annuaire

3.4 Service de validation

Un certificat émis par une IGC peut devenir invalide pour différentes raisons :

- perte ou vol de la clé privée associée
- fin de mandat
- date de péremption
- …

Le cas de la date de péremption est traitée par le fait que les certificats contiennent les dates de début et de fin de validité.
Dans les autres cas, l'IGC doit procéder à la *révocation* des certificats concernés.

L'utilisateur (service ou personne) d'un certificat doit donc avoir un moyen de vérifier que ce certificat est valide à un instant donné.
Pour cela, une IGC doit fournir un service de validation permettant à tout instant de vérifier la validité des certificats qu'elles délivrent.

La méthode la plus employée jusqu'à maintenant est la publication d'une liste appelée *Certificate Revocation List* (**CRL**) qui est la liste des numéros de série des certificats révoqués.
Une CRL a un format standardisé et est signée à l'aide de la clé privée de l'AC émettrice. Elle peut être subdivisée en plusieurs CRLs, ceci pour des raisons de performances. Elle peut être publiée via le même annuaire de publication que les certificats.

Exemple de CRL

```
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: /C=FR/O=CNRS/CN=CNRS-Standard
        Last Update: Oct 29 23:14:44 2001 GMT
        Next Update: Nov 28 23:14:44 2001 GMT
Revoked Certificates:
    Serial Number: 01
        Revocation Date: May 16 11:11:21 2001 GMT
    Serial Number: 02
        Revocation Date: May 11 10:25:22 2001 GMT
    Serial Number: 03
…
        Revocation Date: May  2 14:26:52 2001 GMT
    Serial Number: 06
        Revocation Date: May  4 16:08:23 2001 GMT
    Serial Number: 2B
        Revocation Date: Aug  8 07:11:23 2001 GMT
    Signature Algorithm: md5WithRSAEncryption
        7a:73:1a:5c:c6:30:58:79:56:f9:c9:5a:cb:10:f7:77:88:17:
        df:a8:33:3f:89:47:60:4c:98:e2:63:41:ec:76:81:61:c3:7e:
        d2:0c:70:f2:fe:3e:a9:5c:96:af:69:bd:ae:7a:0a:e6:1f:9c:
        30:9d:3d:da:53:2b:c9:64:e0:d3:63:2d:60:79:4e:4e:9b:74:
        c0:95:b1:17:6a:cb:1c:81:5c:b6:94:6c:7e:7d:ca:d8:2f:3d:
        33:d0:d5:74:3e:64:90:f9:08:72:32:93:6a:a0:6e:78:73:ee:
        13:be:e4:6a:d6:df:77:e9:73:ec:8c:39:32:57:86:fa:bc:b2:
        bc:d1:f8:67:45:ba:b0:aa:a9:51:11:ea:0c:83:d0:c2:ff:3f:
        e7:93:3c:08:7f:e3:4a:1d:1b:b7:ba:06:5f:f0:4b:6f:d9:3d:
        9e:6c:ec:5f:01:a8:58:4b:9d:48:74:00:fa:ca:97:4d:12:a7:
        f1:9f:87:a9:2c:2f:ca:52:17:a4:2e:79:50:aa:54:a7:b5:db:
        06:49:13:d3:a7:8d:2b:3e:10:36:ea:85:17:08:d5:aa:b8:66:
        58:b5:12:7b:35:c8:d5:f8:a6:68:7e:42:fc:1f:89:15:2d:42:
        c2:8a:46:7f:74:81:bd:89:9b:d7:52:86:38:09:0a:c2:47:85:
        4d:9b:c5:8b
-----BEGIN X509 CRL-----
MIICrDCCAZQwDQYJKoZIhvcNAQEEBQAwNDELMAkGA1UEBhMCRlIxDTALBgNVBAoT
BENOUlMxFjAUBgNVBAMTDUNOUlMtU3RhbmRhcmQXDTAxMTAyOTIzMTQ0NFoXDTAx
…
UKpUp7XbBkkT06eNKz4QNuqFFwjVqrhmWLUSezXI1fimaH5C/B+JFS1CwopGf3SB
vYmb11KGOAkKwkeFTZvFiw==
-----END X509 CRL-----
```

La technique des CRL s'avère peu optimisée que ce soit pour des raisons de performances (volumes des certificats à gérer) ou d'efficacité. En effet les CRLs ne permettent pas de diffuser rapidement l'information de révocation d'un certificat, ce qui peut-être très préjudiciable dans les cas d'applications nécessitant un haut niveau de sécurité.

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

3.5 Service de Séquestre

Si un utilisateur perd la clé privée associé à son certificat, il perd également la possibilité de déchiffrer toutes les données chiffrées avec la clef publique associée. Pour éviter cette situation, le Service de Séquestre a pour fonction principale d'archiver les couples de clés privées/publiques correspondants aux certificats délivrées par l'IGC. Dans le cas de la perte d'une clé privée, ce service permet ainsi de la récupérer.
Archiver les clés privées implique que le couple de clés soit générer au niveau de l'IGC et non par les utilisateurs.
Ce service ne peut être mis en place que pour les certificats de chiffrement, pour lequel il est vraiment nécessaire. Par contre, archiver les clés privées de certificats de signature est non seulement beaucoup moins utile (il suffit de délivrer un nouveau certificat), mais en plus il ne faut pas le faire si on veut pouvoir donner un sens légal à cette signature.

3.6 Scénario de contrôle d'accès à une ressource par certificat

La figure ci-dessous illustre un exemple de scénario possible pour une demande d'accès à une ressource :

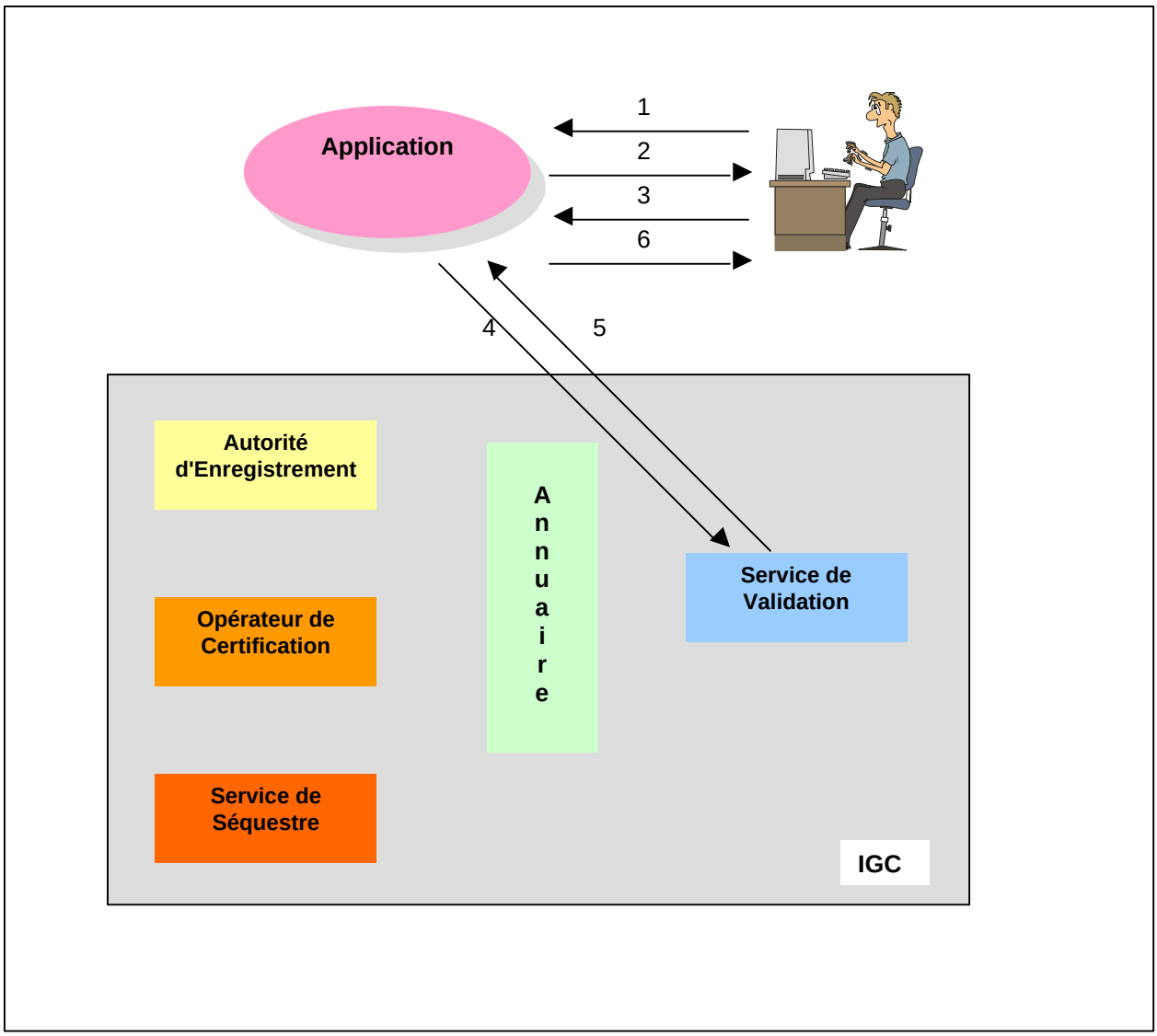

Figure 9 : Demande d'accès à une ressource

1. demande d'accès à l'application
2. demande par l'application du certificat de l'utilisateur
3. présentation du certificat par l'utilisateur
4. si le certificat est valide, l'application vérifie auprès du service de validation de l'IGC s'il n'est pas révoqué
5. réponse du service de validation
6. ouverture ou non de l'accès suivant le résultat de la vérification

Le seul fait d'avoir un certificat valide délivré par une AC peut, dans certains cas, suffire pour définir un droit d'accès. Mais un certificat est avant tout une méthode d'authentification et non une méthode de gestion de droits. En général, une IGC doit être couplée avec un autre système pour la gestion des droits.

3.7 Politiques de Certification

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

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

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

La Déclaration des Pratiques de Certification (DPC) a pour but de décrire les détails des processus techniques mise en oeuvre au sein des différentes composantes de l'IGC (CA, RA,...), conformément à la PC.

En résumé, la PC spécifie des objectifs de sécurité qu'il est nécessaire d'atteindre pour la sécurisation de l'application utilisatrice des services de l'IGC, alors que la DPC spécifie les moyens mises en oeuvre pour atteindre ces objectifs.

Les politiques de certifications peuvent permettre d'établir des normes communes d'interopérabilité et des critères d'assurance communs entre plusieurs organismes.

3.8 Protection des clés privées

La mise en place d'une IGC dans le but de déployer des applications utilisant les certificats X509 n'a de sens que si un niveau de confiance suffisant peut être établi entre les différents protagoniste.
Vis à vis de l'IGC elle-même, la confiance est apportée par sa politique de certification et les moyens mis en œuvre pour la faire respecter.
Le point faible concernent les possesseurs de certificats, car ce sont eux qui ont la responsabilité de la protection de leur certificat et de la clé privée associée.

En effet si on veut pouvoir faire confiance à un certificat, il est nécessaire que la clé privée associée soit correctement protégée. Ceci est vrai aussi bien pour les certificats de personnes que pour ceux de serveurs.
Concernant les certificats de personnes, il est absolument indispensable de mettre en place une formation dans le but de faire comprendre aux utilisateurs ce qu'est un certificat, sa fonction et pourquoi il faut protéger la clé privée associée.

Le choix du support physique des certificats et de la clé privée associée dépendra du niveau de sécurité que l'on veut atteindre :

- **un fichier sur disque dur**
  Cette solution a l'avantage d'être facile à mettre en œuvre. Mais elle implique que le niveau sécurité du certificat et de la clé privée sera directement dépendant du niveau de sécurité de l'ordinateur (poste de travail ou serveur) sur lequel ils seront installés. De plus le certificat et la clé privée peuvent facilement être installés sur différents postes de travail.
- **une carte à puce**
  Cette solution, plus chère, matérialise le certificat et la clé privée sous la forme d'une carte. Elle facilite l'"éducation" des utilisateurs qui associe mieux le certificat à une pièce d'identité personnelle.

Il est clair que dans le cas d'applications sensibles, seule une solution à base de cartes à puce pourra apporter un niveau de sécurité raisonnable.

3.9 Formats de messages

Pour permettre le transfert des requêtes et des certificats entres les composant d'une IGC ainsi qu'entre l'IGC et ses utilisateurs, plusieurs formats de messages ont été standardisés :

- *PKCS7* : format de données signées. Permet par exemple le transfert d'un ou de plusieurs certificats.
- *PKCS10* : format de requête de certification d'une clé publique
- *PKCS12* : format de stockage de certificats et de clés privées. C'est ce format qui est utilisée par exemple pour la sauvegarde d'un certificat et de sa clé privée.
- Etc.

Les certificats X509 ainsi que ces différents types de messages utilisent un encodage appelé *DER* (Distinguished Encoding Rules) qui est un sous-ensemble de l'encodage *BER* (Basic Encoding Rules). Le format DER des certificats numériques se prête mal aux applications du type courrier électronique car il contient des octets qui pourraient être "malmenés" par certains logiciels de transfert de messages. Le format *PEM* (Privacy Enhanced Mail) y remédie en utilisant l'encodage base 64.

3.10 Les standards

Le groupe PKIX a été mis en place en 1995 par l'IETF. Ce groupe travaille sur tous ce qui touche à l'utilisation de certificats X509 et de leurs applications dans l'Internet, en particulier :

- Une instanciation des certificats X.509v3 et des listes de révocation (Certificate Revocation List, CRL) X.509v2 pour une infrastructure adaptée à l'Internet. En effet, un certificat X.509v3 est une structure de données complexe, composée de nombreux champs, et comportant de nombreuses extensions optionnelles et de nombreuses options. Cette particularité permet l'utilisation des certificats X.509 pour beaucoup de situations et d'environnements mais rend difficile la création de produits interopérables. C'est pourquoi le groupe PKIX a défini, dans le RFC 2459, une instanciation particulière de la norme X.509 pour l'Internet en décrivant le contenu des certificats et la liste des extensions obligatoires et optionnelles.
- Des protocoles d'exploitation, qui permettent de distribuer les certificats et les listes de révocation aux systèmes qui les utilisent. Divers systèmes de distribution sont développés, en particulier des procédures basées sur LDAP, HTTP, FTP, et X.500.

- Des protocoles de gestion, qui permettent aux différentes entités composant la PKI (autorités de certification, possesseurs de certificats...) de dialoguer et d'échanger les informations relatives à la gestion de l'ensemble : demande de création ou de révocation de certificat, certification réciproque entre autorités de certification...
- Des règles d'usage et des considérations pratiques : exigences en matière d'identification des sujets, sécurité physique, règles pour la révocation des certificats...

4. SSL : Secure Socket Layer

Une des applications du chiffrement à base de certificats X509 consiste à sécuriser la couche de transport sur TCP. Contrairement à IPSec mis en œuvre au cœur du réseau et qui concerne les couches basses, cette approche de bout en bout n'impose pas un déploiement sur l'ensemble de l'infrastructure existante. Par ailleurs, elle est indépendante des applications des couches supérieures et peut profiter à toutes les applications classiquement utilisées au dessus de TCP.
Le principale intérêt de SSL consiste à « tunneler » des services de chiffrement sur TCP/IP sans bouleversement ni de l'infrastructure réseau, ni des applications ainsi sécurisées.

4.1 Les objectifs

La problématique visée par SSL est double :

1. le chiffrement des échanges sensibles, essentiellement celui des mots de passe.
2. le contrôle d'accès à certaines applications qu'il faut restreindre à une catégorie de la population. Ce contrôle pouvant aller jusqu'à une identification individuelle.

Dans de nombreux cas, le mot de passe, qui circule en clair, est une ressource critique. Ce problème de sécurité est d'autant plus redoutable que de nombreux établissements unifient les mots de passe de toutes les applications grâce à un référentiel unique des usagers sous forme d'un annuaires LDAP. Cette démarche ambitieuse ne doit-elle pas obligatoirement s'accompagner du chiffrement systématique du mot de passe, au moins quand celui-ci sort de l'Intranet ? Nous verrons que SSL offre de bonnes solutions dans ce domaine.

SSL permet, selon les conditions dans lesquel il est utilisé, d'assurer la confidentialité de la communication et de prévenir l'usurpation d'identité :

1. **Chiffrement** : seul le serveur dispose d'un certificat, celui-ci a été émis par une autorité de certification inconnue du client. La communication entre le serveur et le client peut être chiffrée. Il n'y a cependant pas d'authentification possible des parties en présence.
2. **Chiffrement et authentification du serveur** : seul le serveur dispose d'un certificat, et celui-ci a été émis par une autorité reconnue comme autorité de confiance par le client ; le « distinguished name » du certificat du serveur contient un champ « CommonName » contenant le nom du serveur contacté. La communication est alors chiffrée et le client authentifie le serveur avec lequel il communique.
3. **Chiffrement et authentification mutuelle** : le serveur et le client disposent l'un et l'autre d'un certificat X509 ; ces certificats ont été émis par des autorités reconnues par le client et le serveur. La communication est chiffrée et une authentification mutuelle du serveur et du client est effective dans la couche SSL. Malheureusement, les applications n'héritent pas forcément de cette authentification. Dans ce cas l'application doit souvent pratiquer elle même une authentification supplémentaire (en général par mot de passe).

4.2 Principes généraux de SSL

Le standard SSL a été proposé par Netscape, la version 3 de ce protocole date de 1996, elle est disponible dans de très nombreuses applications et est très utilisée. SSL V2 est en voie d'extinction ; TLS (Transport Layer Security) version 1 est la version estampillée IETF (RFC2246) de SSLv3, TLSv1 est largement compatible avec SSLv3.

La figure ci-dessous montre la place de SSL dans l'empilement des couches de protocole.

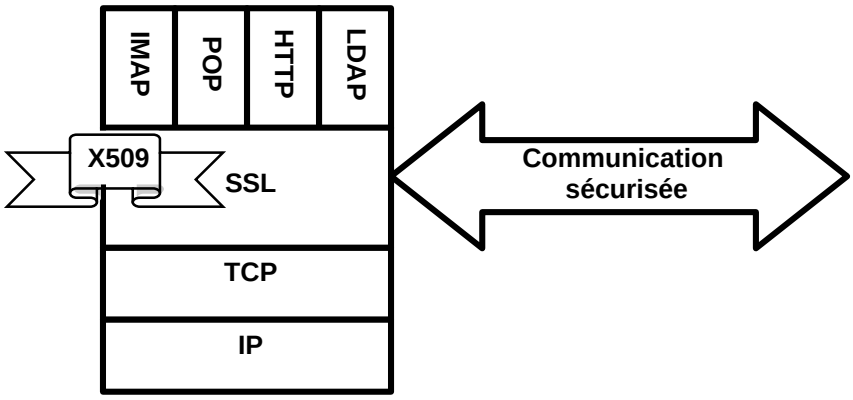

Figure 10 : couche SSL

Un certain nombre d'applications a été adapté pour exploiter la couche SSL ; des ports standards ont été attribués à ces applications :

PROTOCOLE	PORT
LDAPS	636
POP3S	995
IMAPS	993
NNTPS	563
HTTPS	443

SSL est divisé en deux sous-couches, la sous-couche basse « **ssl record protocol** », appuyée sur une couche transport, principalement TCP, encapsule les éléments de protocoles supérieurs. La sous-couche « **SSL handshake protocol** » permet avant tout autre échange entre le serveur et le client de s'authentifier et de négocier un algorithme de chiffrement. En effet, SSL ne spécifie pas directement un algorithme de chiffrement mais par la négociation de celui-ci il permet l'interopérabilité entre deux entités pourvu que celles-ci disposent d'un algorithme de chiffrement (cipher) en commun.

Les apports de SSL :

- La communication est confidentielle. A l'issue de la phase de négociation, les parties en présence disposent d'une clef secrète à usage unique utilisée pour le chiffrement symétrique (DES RC4, …) de la suite des échanges.
- Les parties peuvent s'authentifier en utilisant leurs clefs publiques et des algorithmes de chiffrement asymétriques (RSA, DSS, …)
- L'intégrité des échanges est garantie par l'emploi d'une empreinte numérique (SHA, MD5, …).

La négociation initiale (SSL handshake protocol) permet de choisir les paramètres de chiffrement de la session. Au démarrage de cette session, une version du protocole SSL, une sélection d'algorithmes de chiffrements est retenue. Des techniques de chiffrement à clefs publiques sont

utilisées pour générer un secret partagé afin de protéger la confidentialité de la suite des échanges par du chiffrement symétrique. SSL handshake protocole assure la coordination entre les parties qui partagent un état de la session comprenant en particulier :

- Un identificateur de session (une séquence d'octets arbitraire choisie par le serveur et permettant d'identifier de part et d'autre une session SSL parmi plusieurs).
- Les certificats X509 V3 des partenaires si utiles
- La méthode de compression des données (compression avant chiffrement)
- L'algorithme de chiffrement choisi pour cette session
- Les clefs de chiffrement symétriques

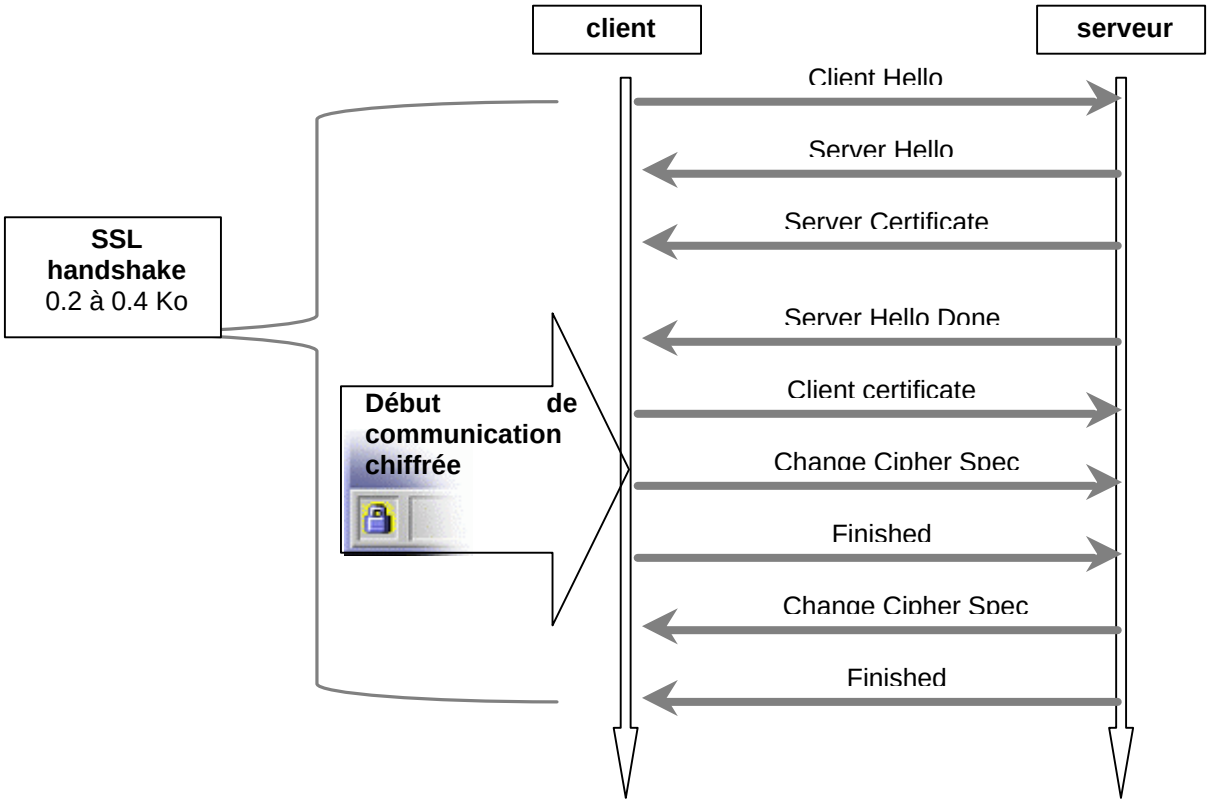

Figure 11 : négociation SSL

Les éléments de protocole « Client Hello » et « Server Hello » permettent d'arrêter la version de SSL, un identificateur de session, une sélection de *cipher* et un algorithme de compression. Durant cette phase le serveur envoie son certificat et la chaîne de certification de son certificat puis un «Certificat-Request» si l'authentification du client est requise. En réponse le client envoie son certificat et sa chaîne de certification ainsi que le «Certificat Verify» : une chaîne chiffrée avec la clef privée du client permettant au serveur de vérifier que le client est bien titulaire du certificat remis durant la session.

Pour reprendre une session initialisée, ou créer une nouvelle session identique, le client envoie un « Client Helllo » en utilisant l'identificateur de la session concernée. Si le serveur trouve cet identificateur dans son cache des sessions, il retourne un « Server Hello » avec le même identificateur de session et restaure en interne les propriétées de cette session (en particulier la clef de chiffrement symétrique commune aux deux parties). Il peut aussi forcer une nouvelle négociation SSL handshake en utilisant un nouvel identificateur.

Le choix de l'algorithme de chiffrement ne doit pas être négligé. La plupart des produits SSL client ou serveur permettent de configurer les algorithmes de compression acceptables lors de la négociation. On veillera au minimum à interdire « NullCipher » qui supprime purement et simplement le chiffrement. Rappelons qu'il est possible en France d'utiliser du chiffrement avec clefs de 128 bits.

4.2.1 TLS : Transport Layer Socket

TLS "Transport Layer Socket" est une évolution du protocole de Netscape SSL V3. TLS est extrêmement voisin de SSL . Une annexe du RFC 2246 décrit comment une implémentation de TLS V1 peut inter-opérer avec SSL V3. La plupart des implémentations de SSL (cleints et serveurs) supporte les deux protocoles.

4.2.2 Openssl : l'implémentation de référence de SSL

Openssl http://www.openssl.org (historiquement SSLeay) est une suite logicielle de chiffrement essentiellement destinée aux développeurs. De très nombreux produits utilisent Openssl, aussi, ce produit fait maintenant partie de la plupart des distributions Linux et plus généralement de la plupart des paquets Unix. Openssl se présente sous la forme d'un ensemble impressionnant de librairie, on peut aussi utiliser la commande en ligne `openssl` qui permet en particulier de manipuler des certificats x509.

4.2.2.1 Quelques opérations de base sur des certificats avec OpenSSL

Afficher le DN du titulaire d'un certificat

Le certificat est dans le fichier cert.pem

```
# /usr/bin/openssl x509 –in cacert.pem -subject -inform PEM -noout
subject= /Email=ca-admin@cru.fr/CN=ca-rssi/O=cru/C=fr
```

Voir le contenu d'un certificat

Si le certificat 'cert.pem' est au format PEM (dans le cas d'un format DER renseigner le paramètre inform avec la chaîne 'DER')

```
# /usr/bin/openssl x509 –inform PEM –in cert.pem –text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3 (0x3)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: Email=ca-admin@cru.fr, CN=ca-cru, O=CRU, C=FR
        Validity
            Not Before: Apr 26 12:54:43 2001 GMT
            Not After : Mar 22 12:54:43 2023 GMT
        Subject: Email=ca-admin@cru.fr, CN=ca-rssi, O=cru, C=fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:f9:54:bd:4f:c4:4b:4c:cb:9c:5e:55:ab:26:76:
                    ........
                    08:7d:3d:2f:18:b6:18:43:b7:42:ee:00:7b:86:62:
                    df:19
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Subject Key Identifier:
                4E:18:88:D0:0D:1E:77:08:B1:0B:F2:F9:9F:0A:52:06:15:DB:77:65
            X509v3 Authority Key Identifier:
                keyid:CE:E8:69:48:74:9C:42:87:18:2F:16:3E:71:94:95:57:FC:F6:E1:D3
                DirName:/Email=ca-admin@cru.fr/CN=ca-cru/O=CRU/C=FR
                serial:00

            X509v3 Key Usage:
                Certificate Sign, CRL Sign
            X509v3 CRL Distribution Points:
                URI:http://pki.cru.fr/cautil/ca-cert.cgi?ca=ca-cru&crl=true

            Netscape Revocation Url:
                http://pki.cru.fr/cautil/ca-cert.cgi?ca=ca-cru&crl=true
    Signature Algorithm: md5WithRSAEncryption
        12:33:e3:33:a1:f1:f9:81:21:ea:cb:99:d0:25:4c:b6:64:ab:
        ......................
        23:c1:00:0b:0f:3f:4f:69:b0:b9:67:6d:67:9c:9a:09:2a:50:
        e6:a2:ff:c5
```

Vérifier un certificat

Vérifier le certificat cert.pem vis à vis des autorités de certification dont les certificats sont compilés dans le fichier ca-bundle.pem.

```
# /usr/bin/openssl verify -CAfile cabundle.pem -purpose any cert.pem
cert.pem: OK
```

Supprimer la *pass phrase* d'une clef privée

Attention ! Êtes vous certain de devoir faire cela ?

```
# /usr/local/openssl rsa -in key.pem -out key.en-clair.pem
read RSA key
Enter PEM pass phrase:
writing RSA key
```

Protéger une clef avec une *pass phrase*

Le chiffrement utilisé dans cet exemple est « triple des ».

```
# /usr/local/openssl rsa -in key.en-clair.pem -des3 -out key.pem
read RSA key
writing RSA key
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

Un client interactif SSL

Openssl permet de lancer un client SSL pour voir une trace d'une session. On peut l'utiliser pour vérifier les éléments de négociation du SSL *handshake protocol* ou pour dialoguer en interactif avec l'application distante (comme nous avons l'habitude de le faire avec « `telnet host port` »)

```
# /usr/bin/openssl s_client -CAfile ca-bundle.crt -connect pki.cru.fr:443
CONNECTED(00000003)
depth=2 /Email=ca-admin@cru.fr/CN=ca-cru/O=CRU/C=FR
verify return:1
depth=1 /Email=ca-admin@cru.fr/CN=ca-servers/O=cru/C=fr
verify return:1
depth=0 /Email=webmaster@cru.fr/CN=pki.cru.fr/O=cru/C=fr
verify return:1
---
Certificate chain
 0 s:/Email=webmaster@cru.fr/CN=pki.cru.fr/O=cru/C=fr
   i:/Email=ca-admin@cru.fr/CN=ca-servers/O=cru/C=fr
 1 s:/Email=ca-admin@cru.fr/CN=ca-servers/O=cru/C=fr
   i:/Email=ca-admin@cru.fr/CN=ca-cru/O=CRU/C=FR
 2 s:/Email=ca-admin@cru.fr/CN=ca-cru/O=CRU/C=FR
   i:/Email=ca-admin@cru.fr/CN=ca-cru/O=CRU/C=FR
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEcTCCA1mgAwIBAgIBAjANBgkqhkiG9w0BAQQFADBQMR4wHAYJKoZIhvcNAQkB
……
xBN/m6F/96yMyNwj86kBLaJ0ucDzA1dz0CCH4V0+hH9iIxzjcg==
-----END CERTIFICATE-----
subject=/Email=webmaster@cru.fr/CN=pki.cru.fr/O=cru/C=fr
issuer=/Email=ca-admin@cru.fr/CN=ca-servers/O=cru/C=fr
---
No client certificate CA names sent
---
SSL handshake has read 3819 bytes and written 314 bytes
---
New, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol  : TLSv1
    Cipher    : EDH-RSA-DES-CBC3-SHA
    Session-ID: D0FF7E11E997158E890B0628B42BBD9EB5580AB165C00A63FE50D7023A7F787C
    Session-ID-ctx:
    Master-Key:
A12AA22E2A9D45909AFF51C24C9103D4479F5A61A1BE7D6429DCEB067ACAE02CB304EEC3AEF3EB6CDF72B85CE74D6F28
    Key-Arg   : None
    Start Time: 1003496958
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

Openssl permet aussi de créer des certificats et il est utilisé par l'UREC et le CRU pour leurs applications de PKI. A noter aussi la possibilité de chiffrer, déchiffrer, signer, vérifier des messages S/MIME.

4.3 STUNNEL : multiplatform SSL tunneling proxy

Comme l'indique son nom *STUNNEL* n'est pas directement une application mais un produit permettant de créer un tunnel SSL. *STUNNEL (http://www.stunnel.org)* est un produit libre (licence GPL) et multi-plateforme. Il est principalement utilisé en frontal (*wrapper*) de serveurs. L'idée est d'utiliser des démons comme POP3, LDAP ou IMAP installés sans fonctionnalité SSL mais à travers *stunnel. Stunnel* est disponible pour Windows et Unix.

stunnel dispose de nombreuses options permettant de l'adapter à de nombreux usages.

4.3.1 Mode client ou serveur

Par défaut *stunnel* est en mode serveur ssl, l'option « `-c` » permet de l'utiliser en mode client ssl. Dans ce cas, c'est stunnel qui prend l'initiative d'ouvrir la session SSL. Stunnel peut aussi être utilisé en mode proxy transparent grâce à l'option « `-T` ». Dans ce dernier cas, le serveur stunnel usurpe l'adresse IP du client pour établir la connexion avec le serveur applicatif SSL.

4.3.2 Utilisation avec inetd

Inetd peut invoquer stunnel (version Unix uniquement). Ce mode permet d'ouvrir n'importe quel service qu'*inetd* peut activer. A cet effet éditez inetd.conf pour y ajouter une définition du service demandé à travers *stunnel* .Exemple :

```
pop3s stream tcp  nowait root /usr/sbin/stunnel … -l /usr/sbin/ipop3d
```

Il convient de noter que ce mode d'utilisation impacte fortement les sessions SSL. En effet, chaque accès impose le « fork » de *stunnel (*puis de l'application visée : serveur pop, imap , etc). Le cache de session SSL est impossible puisque Stunnel n'est pas résident en mémoire, la négociation (*handshake protocol*) est refaite pour chaque accès. Cet usage est donc peu recommandé.

Mode démon

L'option `–d` permet de mettre *stunnel* en mode démon*,* il est alors possible d'utiliser le mécanisme de cache des sessions SSL piloté par le paramètre `–t timeout` (durée de vie des sessions inactives avant de forcer une renégociation de la session ssl).

4.3.3 Compatibilité avec tcp/wrapper

tcp/wrapper est un mécanisme de contrôle des accès configuré dans les fichiers `hosts.allow` et `hosts.deny` ; *stunnel* est compatible avec `tcp-wrapper`, il utilise la librairie `lwrap`. Cela permet plus facilement de respecter une politique de sécurité qui utiliserait tcp-wrapper et de prévenir des failles éventuelles dans le contrôle d'accès.
La syntaxe de `host.allow` est inchangée : `<service> :<host>` indique que l'on accepte les accès au service `<service>` depuis le host `<host>`. Cependant, la définition d'un service peut varier parce qu'il est possible que *stunnel* soit utilisé pour plusieurs applications, en particulier, si stunnel est utilisé pour rediriger un flux vers une autre machine. Se référer à http://www.stunnel.org/faq/run.html#ToC4

4.3.4 Utilisation des certificats

Selon les cas de figure (client ou serveur, avec ou sans contrôle des certificats clients), stunnel peut manipuler quatre catégories de certificats :

1. les certificats des autorités de confiances (trusted CA)
2. le certificat et la clef privée du serveur ssl si stunnel est en mode serveur
3. le certificat et la clef privée du client si stunnel est utilisé en mode client ssl et si la session ssl n'est pas limitée au chiffrement.
4. l'ensemble des certificats clients acceptés si ssl est utilisé en mode serveur avec contrôle d'accès des clients.

4.3.4.1 Certificat serveur

En mode serveur, il est impératif d'installer un certificat serveur accessible dans l'environnement de stunnel ; faute de ce certificat le « ssl handshake protocol » ne peut être établi. Le certificat ainsi que la clef privée associée doivent être concaténés dans un même fichier au format PEM (Privacy Enhanced Mail) et séparé par une ligne vide. Le path de ce fichier est passé en argument à *stunnel* avec l'option «-p». La même méthode est utilisée si *stunnel* est utilisé en mode client avec authentification du client.
Il est important de noter que *stunnel* ne peut utiliser la clef privée associée à son certificat si celle-ci est protégée par un mot de passe. Stunnel impose donc de stocker en clair la clef privée du serveur (ou du client) ! En pratique, cela limite l'utilisation de *stunnel* à des serveurs sur lesquels les seuls *login* autorisés sont ceux des administrateurs de la machine. Il ne semble pas raisonnable d'utiliser une solution stunnel avec certificat du client sur un poste de travail en imposant aux utilisateurs de conserver leur clef privée en clair.

Stunnel est livré avec un certificat de test et la clef privée associée. Cette clef est donc une clef privée publique disponible en GPL pour tout utilisateur d'Internet ! La première chose à faire est donc de se munir d'un certificat correct pour le serveur ; on évitera un certificat auto-signé sauf, éventuellement, si le service à protéger est destiné à une poignée d'administrateurs. En effet il serait déraisonnable d'encourager les utilisateurs à accepter des certificats auto signés !

4.3.4.2 Certificats des autorités de confiance

L'option `-A` permet de désigner un fichier contenant les certificats des autorités de confiance. Tout certificat ne sera considéré comme valide que si sa chaine de certification croise la liste des certificats contenus dans ce fichier. Le format de fichier est semblable à celui utilisé par openssl (openssl est le moteur ssl de stunnel). Ce fichier est la concaténation des certificats au format PEM séparés par des lignes vides.

4.3.4.3 Contrôles d'accès basés sur les certificats clients

Stunnel permet de choisir entre plusieurs configurations pour contrôler l'accès au tunnel SSL. Le contrôle d'accès opéré à partir des propriétés du certificat du client est possible par deux méthodes (nous nous limitons au cas probable dans lequel *stunnel* est utilisé en mode serveur).

- Stunnel permet de vérifier la validité du certificat client. Dans cette configuration, *stunnel* vérifie si ce certificat a été émis par une autorité de confiance. En choisissant une (ou des) autorité(s) de certification dédiée(s) à une classe d'utilisateurs on limite alors l'accès à cette classe d'utilisateurs. Utiliser les options `-v level2` pour spécifier ce mode de contrôle des certificat user et –l'option `-A` pour la liste des autorité de confiance.
- On peut aussi contrôler individuellement la listes des certificats clients autorisés. A cet effet, l'option –a permet de spécifier un répertoire contenant tout les certificats acceptables (utilisation en conjonction avec l'option `-v level3`). Cette méthode permet un contrôle fin mais impose de gérer cette liste de certificats. Ainsi on doit récupérer les certificats de toutes les personnes autorisées (éventuellement plusieurs certificats pour la même personne). Il convient par exemple de mettre à jour ce répertoire quand un utilisateur renouvelle un certificat expiré. Enfin, il est impossible de baser l'expression des certificats acceptables par

une expression portant sur le *distinguished name* . Exemple : accepter tout les certificats dont l'autorité émétrice est une des CA du CRU et dont le champs Organisation est « Université de Rennes1 ». De ce fait, cet usage est à réserver à de très petites communautés d'utilisateurs.

4.3.5 Exemples d'utilisation

Dans l'exemple décrit, on cherche à sécuriser une communication IMAP entre un serveur et un client imap qui ne seraient pas capables ni l'un ni l'autre d'établir/recevoir des sessions imap sur SSL. C'est par exemple le cas de du webmail *IMHO* qui comporte une fonction cliente IMAP et du serveur IMAP de l'université de Washington. Ce dernier supporte nativement les sessions IMAPS mais ne peut malheureusement pas faire un contrôle sur les certificats clients acceptables ce que stunnel autorise.

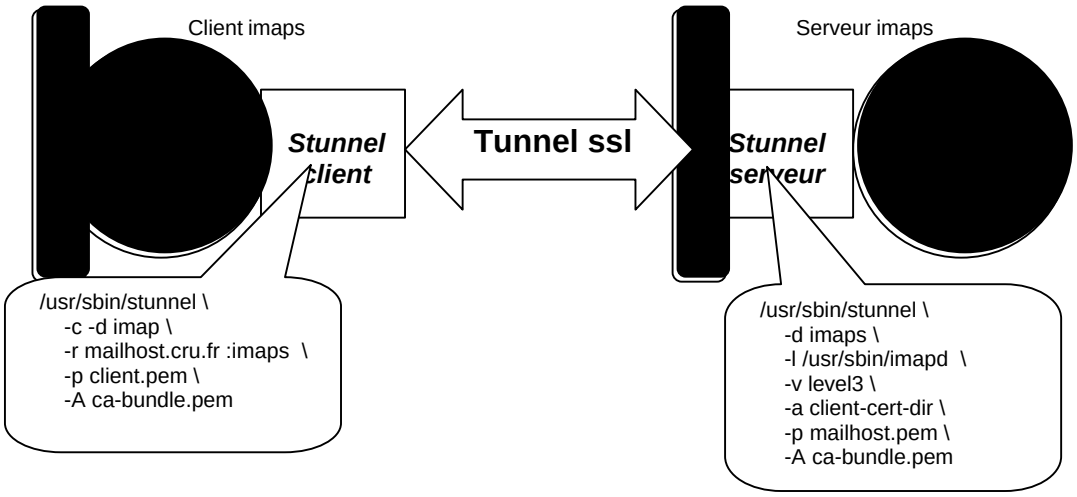

Figure 11 : Tunnel SSL avec STUNNEL

Dans cette configuration, l'ensemble client imap + Stunnel client constitue un client IMAPS, réciproquement, le serveur Stunnel couplé au serveur IMAP constitue un serveur IMAPS.

Client stunnel	Serveur stunnel
`-c` pour être utilisé en mode client **`-d imap`** fonctionne en mode démon en écoute sur le port imap **`-r mailhost.cru.fr:imaps`** redirige le traffic sur le port imaps du host `malihost.cru.fr` **`-p client.pem`** utilise la clef privée et le certificat contenu dans le fichier `client.pem` **`-A ca-bundle.pem`** `ca-bundle.pem` est le path d'un fichier contenant une compilation des certificats des autorités de confiance.	**`-d imaps`** fonctionne en mode démon en écoute sur le port `imaps` **`-l /usr/sbin/imapd`** lance le programme imapd pour traiter les données reçues **`-v level3 –a client-cert-dir`** n'accepte de session entrante que si celle-ci est établie avec un certificat valide présent dans le répertoire `client-cert-dir` **`-p mailhost.pem`** utilise la clef privée et le certificat contenu dans le fichier `mailhost.pem` **-A ca-bundle.pem** idem configuration du client

4.4 IMAPS, POPS

L'utilisation de SSL avec IMAP et POP est un bon exemple du besoin de chiffrer les échanges utilisés pour l'authentification. En effet, la messagerie est la première application que les utilisateurs souhaitent pouvoir utiliser durant leurs déplacements. Les clients de messageries les plus usuels intègrent imaps et pops, aussi, une politique de sécurité supprimant les accès entrant dans le réseau local en protocole POP et IMAP au profit de POPS et IMAPS n'est pas trop contraignante. Signalons que *Netscape messenger*, *Outlook*, et *Eudora* à partir des versions 5.X supportent IMAPS et POPS (*Opera* supporte POP sur SSL mais ne propose pas de service IMAP).

IMAPS et POPS permettent non seulement de chiffrer les échanges entre clients et serveurs, mais aussi d'installer un contrôle d'accès sur le certificat client dès la couche SSL. Cependant, IMAPS n'est que l'empilement de IMAP sur SSL. Aussi, l'authentification IMAP par user + password est le plus souvent conservée même si le certificat X509 du client contient toutes les informations utiles pour authentifier l'utilisateur.

L'utilisation de Stunnel comme décrite dans l'exemple précédent n'est pas indispensable car les serveurs usuels supportent les accès SSL (Washington University, Exchange, …). Dans notre exemple, l'emploi de Stunnel est indispensable parce que le client ne présente pas son certificat dans la phase « handshake protocol » que le serveur en fait la demande, ce qui n'est pas le cas du serveur IMAP WU.

4.5 SMTP/TLS

L'activité pour sécuriser les services de messagerie est intense, dans le domaine de la confidentialité et de la signature, elle concerne S/MIME et PGP, en ce qui concerne l'accès aux boites aux lettres, IMAPS, APOP ou POPS . Le protocole SMTP reste un protocole sans authentification ni chiffrement, il est donc sensible à certaines attaques tel que

- l'écoute du traffic pour capture des contenus de messages
- analyse du traffic pour reconstruire le log des échanges
- détournement de la fonction de relais d'un moteur STMP.

Ce défaut est une des raisons pour laquelle nous recevons tous tant de «spam». Sans précaution particulière, un serveur SMTP peut être utilisé par quiconque pour relayer un très grand nombre de messages vers des destinations sans rapport avec le domaine géré par ce serveur SMTP. Depuis plusieurs années, les CERTs ou le NIC français recommandent aux postmasters de bloquer la fonction de relais, c.a .d. de bloquer les messages provenant de l'extérieur du domaine courant et à destination de l'extérieur de celui-ci. Cette précaution élémentaire a cependant l'inconvénient d'empêcher les utilisateurs nomades d'utiliser normalement leur messagerie.

On peut contourner cette restriction de service par exemple en imposant l'emploi d'un webmail. Ces passerelles permettent un accès http adapté aux usages nomades etles messages émis le sont par la passerelle webmail. Celle-ci étant toujours dans le domaine local du moteur SMTP n'est pas concernée par les mesures anti-relais. Une autre astuce « POP before » consiste à accepter de relayer les messages provenant d'une machine extérieur au domaine mais ayant établie préalablement une session POP avec le serveur. En effet, le protocole POP un est protocole avec authentification.
Une solution basée sur l'emploi de TLS permet de chiffrer les échanges (y compris l'échange des adresses e-mails des différents correspondants) et éventuellement de contrôler les serveurs autorisés à établir des sessions SMTP entrantes ainsi qu'autoriser sélectivement le relais. Pour ces deux cas, la session TLS comporte une demande de certificat du client. Le serveur vérifie alors que le titulaire du certificat est autorisé à faire un tel usage du service de messagerie. Cette solution très sure s'avère conviviale pour les utilisateurs de PC portable sur lequel sont installés un certificat personnel.

SMTP/TLS est décrit dans le rfc 2487. le principe de SMTP/TLS consiste en l'ajout d'une directive **« STARTTLS »** envoyée par un serveur supportant SMTP sur TLS pendant la session Extended SMTP. Si le client accepte cet élément de service en répondant avec la même directive et en entamant la négociation TLS. Ces échanges continuent sur le port SMTP standard, il n'est donc pas indispensable d'ouvrir les filtres des gardes barrières pour un nouveau port.

Postfix puis Sendmail permettent de mettre en œuvre des sessions SMTP/TLS. On trouve de nombreux documents concernant la mise en œuvre de sendmail/tls ou postfix/tls. Ci dessous deux bonnes références :

- Postfix : http://www.hsc.fr/ressources/breves/postfix-tls.html
- Sendmail : http://www.sendmail.org/~ca/email/starttls.html

4.6 HTTPS

HTTPS est l'application la plus connue de SSL. L'intérêt n'est pas seulement celui du « e-commerce », beaucoup d'applications du web comportent une phase d'authentification par mot de passe. Celle-ci est parfois intégrée à l'application ou réalisée par le protocole http d'après la configuration du serveur (fichiers .htaccess pour Apache). HTTPS permet de chiffrer :

- les pages envoyées au client,
- les URLs des documents
- les données des formulaires postées depuis le client
- les cookies
- le user/password du protocole http.

Le chiffrement des échanges de données sensibles comme un mot de passe n'est pas le seul intérêt de HTTPS, dans le cas ou les utilisateurs dispose d'un certificat X509, celui-ci peut être utilisé pour contrôler les accès à certaines parties du serveur. On peut même authentifier l'utilisateur sans login. Le développement d'applications sécurisées et libérées des problèmes de gestion de mot de passe est alors possible.

4.7 Apache

Nous nous limiterons au cas d'utilisation de https sur serveur Apache. Apache est configuré par défaut avec le module *mod_ssl* permettant de faire du HTTPS dans de nombreuses distribution de Linux. Cette solution est opérationnelle sur toute plateforme Unix et sur Windows NT. Elle est basée sur l'incontournable librairie OpenSSL.

Le module pour Apache mod_ssl est hérité d'apache-ssl date de 1995. mod_ssl contient 12 000 lignes de code (pour comparaison, Apache : 80 000 lignes, OpenSSL : 180 000 lignes), on estime qu'il est utilisé par 12% des serveurs Apache.

4.7.1 Installation

Mod_ssl peut être utilisé en mode DSO (Dynamic Shared Object), mais si vous devez le compiler avec Apache la méthode reste très simple. On suppose ci-dessous que les sources des dernières versions d'apache de mod-ssl et openssl sont dans /usr/local/src (les numéros de versions ne sont qu'indicatifs).

```
cd /usr/local/src/openssl-0.9.6;                          Compiler
./configure; make                                         openssl

cd ../mod-ssl-2.8.2;                                      Compiler
./configure –with ../apache_1.3.19 \                      mod_ssl
   –with-ssl ../open-ssl-0.9.6 –prefix /usr/local/apache

cd ../apache_1.3.19; make ; make certificate type=dummy   Compiler Apache
 make install                                             Générer un
                                                          certificat
                                                          auto-signé,
                                                          installer

/usr/local/apache/bin/httpd –DSSL                         Démarrer avec
                                                          SSL
Netscape https://localhost                                Tester
```

Bien entendu, le démarrage du serveur sans configuration particulière après la compilation ou l'installation d'une distribution binaire présente peu d'intérêt.

4.7.2 Configuration

La première étape consiste à obtenir et installer un certificat adapté pour le serveur (les machines du CNRS peuvent bénéficier d'un certificat UREC, celles des universités et grandes écoles d'un certificat CRU).

Le reste de la configuration est décrit dans le fichier « `httpd.conf` », les variables du module mod_ssl ont toutes un nom préfixé par la chaîne SSL. Elles sont très bien décrites sur la page http://www.modssl.org/docs/2.8/ssl_reference.html . Il est inutile de passer en revue ces variables dans ce document. Voici cependant, quelques points auxquels vous devez faire attention.

4.7.2.1 Protéger la clef privée du serveur

On peut chiffrer la clef privée du serveur (voir **Protéger une clef avec une *pass phrase***). Toutefois cette protection n'est pas sans inconvénient : lors de chaque démarrage du serveur, l'opérateur est contraint de saisir la « *pass phrase* ». Par ailleurs si le serveur est compromis, la protection par chiffrement de la pass phrase n'est peut être pas suffisante : l'attaquant qui se serait introduit sur le système peut par exemple modifier Apache pour s'emparer de la clef privée après que le serveur ait déchiffré celle-ci.
La variable **`SSLPassPhraseDialog`** de `mod_ssl` permet de choisir comment se déroule le dialogue pour cette saisie et permet de remplacer le mécanisme interne d'Apache par un programme de votre choix, pourvu que celui-ci écrive dans `stdout` la *pass phrase* de protection de la clef. Ce programme peut par exemple vérifier l'intégrité de fichiers critiques (calcul et vérification de l'empreinte MD5 du binaire et de la configuration d'apache, etc) avant la saisie de la *pass phrase*.
Dans de nombreux cas on préfèrera ne pas chiffrer la clef privée, il importe alors de protéger au maximum le serveur (serveur dédié, accessible uniquement via son interface http, pas de login autre que celui de l'administrateur, protection par un garde barrière etc).

4.7.2.2 Gestion du cache de session

Afin de ne pas forcer une renégociation de la couche SSL pour chaque accès et en particulier pour chaque accès parallèle, mod_ssl gère un cache des sessions SSL partagé par les différents processus Apache. Plusieurs variables permettent de régler le fonctionnement de ce cache :

`SSLSessionCache`	indique la méthode utilisée pour le partage des caches entre processus. Par défaut cette fonctionnalité n'est pas activée ce qui peut pénaliser fortement les performances et la charge du serveur en particulier lorsqu'un client web charge en parallèle plusieurs images référencées dans une page HTML. Exemple : `SSLSessionCache dbm:logs/ssl_scache`
`SSLSessionCacheTimeout`	durée de conservation des infos du cache (en secondes). Si un client tente de reprendre une session inactive depuis cette durée, le serveur force la renégociation SSL (La valeur par défaut : 300 secondes est un peu faible quand on demande aux utilisateurs de remplir des formulaires complexes) .
`SSLMutex`	méthode de prévention des conflits d'écriture dans le cache. Exemple : `SSLMutex file:/var/log/ssl_mutex`

4.7.2.3 Gestion des certificats

Plusieurs variables permettent de spécifier la localisation des différents certificats. Ces variables peuvent être redéfinies pour chaque serveur virtuel.

`SSLCertificateFile`	le certificat du serveur
`SSLCertificateKeyFile`	la clef privée du serveur
`SSLCACertificateFile`	les certificats des autorités reconnues pour la vérification des certificats clients. Compilation dans un fichier au format PEM. Attention, la valeur par défaut contient les certificats de toutes les autorités de certifications pré-initialisées dans le navigateur Netscape et seulement celle-ci.
`SSLCACertificatePath`	Idem mais les certificats sont regroupés dans un répertoire à raison d'un certificat par fichier. Les fichiers doivent être suffixés par l 'extension `.crt` et installés avec le Makefile distribué avec Apache. Le but est de permettre d'accéder directement à un certificat à partir du DN de l'autorité de certification grâce à un jeu de liens (ce fichier Makefile est installé dans le répertoire par défaut de `SSLCACertificatePath).`
`SSLCACertificateChainFile`	Lors de l'établissement de la session SSL, le serveur envoie outre le certificat du serveur la chaîne de certification c'est à dire la liste des certificats de chaque autorité de certification depuis l'autorité racine jusqu'à l'autorité ayant émis le certificat serveur. Votre autorité de certification vous a probablement remis ce fichier avec votre certificat.
`SSLCARevocationPath`	Le répertoire contenant les listes de révocation. Il est important d'utiliser cette possibilité si votre serveur utilise les certificats clients pour authentifier des utilisateurs.

4.7.2.4 Les ports en écoute

Le numéro de port privilégié pour le service `https` est le 443. Si l'on désire ne pas indiquer de numéro de port dans les URL, la directive d'Apache «`listen`» (ou la définition du virtual host concerné) doivent mentionner ce numéro de port.

4.7.2.5 Les serveurs virtuels et la gestion de certificats

Rappelons le mode de fonctionnement des serveurs virtuels avec le protocole http 1.1 : une directive host permet au client d'adresser le serveur virtuel demandé. Le serveur reconnaissant cette directive applique alors le contexte du serveur virtuel demandé par le client. Ce mécanisme permet d'installer un grand nombre de serveur web sur une machine ne disposant que d'une seule adresse IP mais autant d'alias (enregistrement CNAME du DNS) que de serveurs virtuel.

Nous avons expliqué que la couche SSL inclut l'échange et la vérification des certificats. Le client reçoit une alarme en particulier si le nom du host contacté et celui figurant dans le DN du certificat du serveur diffèrent (échec de l'authentification du serveur). Un serveur http doit donc disposer d'autant de certificats que de serveurs virtuels HTTPS. En outre, la phase d'authentification du serveur se faisant dans la couche SSL, elle est préalable à l'établissement du dialogue http. En conséquence, il est impossible pour le serveur de choisir le certificat qu'il doit présenter au client n'ayant pas reçu la directive http host de désignation du serveur contacté.
Cela impose d'utiliser une adresse IP par serveur virtuel. Dans ce cas le serveur peut sélectionner le certificat du serveur virtuel contacté grâce à l'adresse IP appelée. Cette restriction n'est pas une spécificité d'Apache mais elle est liée à l'architecture même de HTTPS.

4.7.2.6 Chiffrement

Pour « mettre en route » le chiffrement pour un serveur virtuel, vous devez positionner la variable « `SSLEngine on` » et choisir les algorithmes de chiffrement négociables par le serveur en renseignant la variable `SSLCipherSuite` . A noter qu'il est possible d'autoriser l'absence de chiffrement (cipher NULL). La valeur par defaut de ce paramêtre n'autorise pas la négociation du cypher NULL. Si l'on souhaite modifier cette valeur, par exemple pour forcer un chiffrement avec une clef d'une longueur d'au moins 128 bits, on se reportera à la documentation de mod_ssl (http://www.modssl.org/docs/2.8/ssl_reference.html#ToC9).

4.7.2.7 Contrôle d'accès

L'authentification des utilisateurs par leurs certificats est une des fonctionnalités les plus intéressantes de mod_ssl. Bien entendu, elle suppose la diffusion de certificat. La directive «`SSLVerifyClient require`» permet de forcer la vérification du certificat client. Dans ce cas, seuls les possesseurs d'un certificat valide ont accès à la partie du serveur sur laquelle s'applique cette directive.

- Certificat client émis par une autorité de confiance du serveur (directive `SSLCACertificate`)
- Chaîne de certification du certificat client d'une longueur inférieur ou égale à la valeur spécifiée par la directive `SSLVerifyDepth`
- Certificat client non révoqué (directive `SSLCARevocationPath`)

La directive `SSLRequire` permet de filtrer, parmi les certificats valides, lceux qui sont acceptés, en appliquant une expression sur un ensemble de variables notamment celles héritées de la session SSL. Exemple :

```
SSLRequire (%{SSL_CLIENT_S_DN_O} eq "CRU" ) \
        and %{REMOTE_ADDR} =~ m/^195\.220\.94\.[0-9]+$/ )
```

Pour cette utilisation, il est fondamentale de s'assurer que les autorités de certifications partagent des règles de nommage communes, dans l'exemple, que le champs O=CRU est réservé à des entités du « Comité Réseau des Universités ». La syntaxe d'écriture des expressions de la directive SSLRequire est assez riche (opérateurs booléens, parenthèsage, appartenance ensembliste, compaison, expressions régulières, etc) , voir http://www.modssl.org/docs/2.8/ssl_reference.html#ToC23 .

S'il n'est pas possible de sélectionner à l'aide d'une expression régulière l'ensemble des personnes autorisées, on peut se replier sur une énumération des Distinguish Name. Le principe retenu consiste à énumérer les DN autorisés dans un fichier d'authentification de type *user-password*. Cette méthode et la syntaxe choisie pour mod_ssl sont assez étranges et déparent quelque peu la grande rigueur de conception de mod_ssl. Un fichier d'authentification basic est utilisé avec pour chaque entrée de ce fichier un mot de passe fictif. L'option `SSLOptions +FakeBasicAuth` permet de spécifier à Apache l'emploi de ce fichier .

Exemple :

```
SSLVerifyClient      require
SSLOptions           +FakeBasicAuth
SSLRequireSSL
AuthName             "authentification par certificat"
AuthType             Basic
AuthUserFile         /usr/local/apache/conf/htpasswd
require              valid-user
```

Contenu du fichier `htpasswd` :

```
/C=FR/O=CRU/CN=foo:xxj31ZMTZzkVA
/C=FR/O=CNRS/OU=UREC/CN=toto/EMAIL=toto@urec.cnrs.fr:xxj31ZMTZzkVA
/C=FR/O=CNRS/OU=DR15/CN=toto:xxj31ZMTZzkVA
```

Ici, la valeur du mot de passe n'est pas un exemple mais bel et bien la valeur que vous devez mettre dans ce champs. Cette valeur étrangement universel est le resultat du chiffrement avec DES de la chaîne «password» avec le salt «xx» (essayer perl -e 'print crypt("password","xx"), "\n";').

4.7.2.8 Programmation CGI

Il est possible d'exploiter dans des CGI des données héritées de la session SSL. On peut alors dépasser le simple contrôle d'accès et authentifier la personne à partir de son certificat pour lui proposer un environnement personnalisé. A cet effet positionner l'option `StdEnvVars` qui permet d'exporter vers le cgi un nombre important de variables. On veillera à limiter la portée de cette option aux cgi ou *server side include* qui en ont besoin car cette option est coûteuse. Cette possibilité permet la programmation très rapide d'applications très conviviales puisque toute demande de mot de passe devient superflue.

`SSLOption +StdEnvVars`

On peut visualiser les variable disponibles et leur valeur en appelant le petit cgi «`printenv.cgi`», (attention toutefois à ne pas laisser ce cgi dans une partie publique de votre serveur : il révèle des données qui pourraient aider des personnes hostiles).

```
print "Content-type: text/html\n\n";
while (($key, $val) = each %ENV) {
        print "$key = $val<BR>\n";
}
```

Exemple en perl de programmation utilisant cette méthode (dans ce cas, on identifie la personne avec le champs email du DN du certificat du client, si ce certificat a été validé par mod_ssl) :

```
## use https client certificat information if define.
    if (($ENV{'SSL_CLIENT_S_DN_Email'}) && ($ENV{'SSL_CLIENT_VERIFY'} eq 'SUCCESS')) {
       $user->{'email'} = $ENV{'SSL_CLIENT_S_DN_Email'};
}else{
…
}
```

4.7.2.9 Un exemple simple de configuration

```
SSLSessionCache         dbm:logs/ssl_scache          # cache SSL en dbm
SSLSessionCacheTimeout  600                          # renégociation après 10m
SSLMutex  file:logs/ssl_mutex                        # exclusion mutuelle par lock
SSLLog  logs/ssl_engine_log                          # log spécifique ssl
SSLLogLevel info

SSLEngine on
SSLCertificateFile conf/ssl.crt/server.crt           # le certificat serveur
SSLCertificateKeyFile conf/ssl.key/server.key        # la clef
SSLCACertificateFile conf/ssl.crt/ca-bundle.crt      # la compilation des CA de confiance
SSLCertificateChainFile conf/ssl.crt/ca.crt          # la chaîne de certification du serveur
SSLCARevocationPath conf/ssl.crl                     # les CRL


SSLVerifyClient require                              # accès avec certificat valide uniquement
SSLVerifyDepth  3                                    # longueur max de chaîne de certification
                                                     # du client

<Files ~ "\.(cgi|shtml)$">                           # On remonte dans les cgi et SSI les
     SSLOptions +StdEnvVars                          # variables extraites de la session SSL
</Files>

<Location />                                         # on impose des clients en
                                                     # /C=FR/O=CNRS|CRU

        SSLRequire  %{SSL_CLIENT_S_DN_C} eq  "FR" and %{SSL_CLIENT_S_DN_O} in {"CRU", "CNRS"}
</Location>
```

5. S/MIME

L'application des techniques à base de certificats X509 à la messagerie porte le nom de S/MIME. La signature et le chiffrement des messages sont bien entendu les services visés par S/MIME. La version initiale, S/MIME v2 a été développée par un consortium privé d'éditeurs. S/MIME v2 est techniquement lié à l'algorithme d'échange de clefs RSA. Celui-ci est breveté par RSA Data Security, Inc, en conséquence, les RFCs qui décrivent S/MIME n'ont pas été adoptés comme des standards de l'IETF, ils ont tous un «*status informational*».

- S/MIME Version 2 Message Specification (RFC 2311)
- S/MIME Version 2 Certificate Handling (RFC 2312)
- PKCS #1: RSA Encryption Version 1.5 (RFC 2313)
- PKCS #10: Certification Request Syntax Version 1.5 (RFC 2314)
- PKCS #7: Cryptographic Message Syntax Version 1.5 (RFC 2315)
- Description of the RC2 Encryption Algorithm (RFC 2268)

S/MIME v3 a fait l'objet de RFCs adoptés par l'IETF comme des standards (*Internet Official Protocol Standards)* en 1999 :

- Cryptographic Message Syntax (RFC 2630)
- S/MIME Version 3 Message Specification (RFC 2633)
- S/MIME Version 3 Certificate Handling (RFC 2632)
- Diffie-Hellman Key Agreement Method (RFC 2631)

Comme le nom de ces standards l'indique, S/MIME utilise le standard MIME des messages pour structurer l'envoi des informations de chiffrement et de signature de messages. A la différence de SSL, les éléments de chiffrement ne sont pas négociés par les parties en communication. En effet, le routage des messages de proche en proche par des serveurs SMTP ne permet pas une communication directe.

5.1 Intégrité et signature

La signature est constituée d'une empreinte (par exemple SHA1) chiffrée avec la clef privée de l'émetteur. Tout destinataire peut vérifier l'intégrité en recalculant cette empreinte et en la comparant à l'empreinte déchiffrée avec la clef publique de l'émetteur. La provenance du message est garantie puisque seul le titulaire de la clef privée associée au certificat a pu chiffrer l'empreinte. La signature est valide si la chaîne de certification est correcte et si elle appartient aux autorités de confiance. Le type MIME d'un message signé est `multipart/signed`, la signature est contenue dans la structure `pkcs7-signature`. Il n'est pas utile que votre certificat soit déposé dans un annuaire accessible de vos correspondants pour que vous puissiez leur envoyer des messages signés. En effet, puisque ces informations sont publiques, un message signé peut contenir le certificat de l'émetteur ainsi que la chaîne de certification associée. En outre, l'agent de messagerie de votre correspondant peut stocker le certificat de l'émetteur du message reçu. Dès lors, il peut envoyer des messages chiffrés vers ce correspondant. Cette mécanique simple permet l'utilisation effective de la signature et du chiffrement même sans le déploiement d'annuaires par simple échange préalable de messages signés.

Exemple de message signé :

```
From: Dupont <dupont@cru.fr>
Message-ID: <3BDEB0ED.DF991014@cru.fr>
Date: Tue, 30 Oct 2001 14:53:49 +0100
MIME-Version: 1.0
To: durand@cru.fr
Subject: Message signe
```

```
Content-Type: multipart/signed; protocol="application/x-pkcs7-signature"; micalg=sha1;
  boundary="------------separateur"

This is a cryptographically signed message in MIME format.

--------------separateur
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: 8bit

Coucou
--------------separateur
Content-Type: application/x-; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIIKsQYJKoZIhvcNAQcCoIIKojCCCp4CAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3DQEHAaCC
. . .
VR0fBEEwPzA9oDugOYY3aHR0cDovL3BraS5jcnUuZnIvY2F1dGlsL2NhLWNlcnQuY2dpP2Nh
PWNhLWNydSZ
-------------separateur--
```

En théorie, la signature d'un message permet de vérifier l'intégrité de celui-ci. En fait, S/MIME (et S/MIME V2) sécurise une entité MIME, c'est à dire, une partie de message ou un message avec toutes ces parties et sous-parties de corps ainsi que les entêtes MIME (essentiellement `Content-Type:` et `Content-Transfert-Encoding:`). Une entité MIME n'inclut pas les entêtes RFC-822 du message. Il en résulte que la signature d'un message n'est pas altérée si des entêtes comme `Date:`, `Subject:` sont modifiées ! Cette sérieuse limitation concerne aussi l'entête `from:`, toutefois, la signature du message contient le «distinguished name» du signataire et les applications vérifient que l'attribut email de ce DN correspond bien au champ `From:` du message. Dans le cas contraire une anomalie est détectée sur la signature.

Il n'est donc pas possible de signer des messages si le champ `From:` de ceux-ci contient un alias de l'adresse d'émetteur. Celle-ci doit donc toujours être la même et être qualifiée, même si le message est local à la machine ou au domaine. Souvenez-vous que rien ne permet de deviner que les adresses nom@host.domaine.fr, nom@domaine.fr et prénom.nom@domaine.fr sont équivalentes.

5.2 Le chiffrement

Pour chiffrer un message, votre agent de messagerie doit disposer du certificat de votre correspondant. En effet, la confidentialité d'un message est obtenue parce que celui-ci est chiffré avec la clef publique contenue dans le certificat du destinataire. Lui seul peut déchiffrer le message à l'aide de la clef privée associée à son certificat.
Le chiffrement est réalisé avec un algorithme de chiffrement symétrique à l'aide d'une clef à usage unique tirée aléatoirement. Cette clef est elle-même chiffrée avec un algorithme asymétrique en utilisant le certificat du destinataire.

Si un message chiffré est adressé à plusieurs correspondants, la clef de chiffrement symétrique est alors chiffrée pour chacun des correspondants et le groupage SMTP (envoi d'un seul exemplaire d'un message ayant plusieurs destinataires) n'est plus possible.

```
Message-ID: <3BE8F0BE.470B0CDD@cru.fr>
Date: Wed, 07 Nov 2001 09:28:46 +0100
From: Aumont <dupont@cru.fr>
MIME-Version: 1.0
To: durand@cru.fr
Subject: un test
Content-Type: application/x-pkcs7-mime; name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"
Content-Description: S/MIME Encrypted Message
```

```
MIAGCSqGSIb3DQEHA6CAMIACAQAxggHZMIHpAgEAMFIwTTEeMBwGCSqGSIb3DQEJARYPY2Et
…
nshgKJB
```

Attention : le chiffrement d'un message rend caduque la plupart des traitements par antivirus ; seuls des traitements effectués lors de la sauvegarde «en clair» d'un attachement ou ceux qui seraient intégrés dans le visualiseur associé à un attachement pourraient éventuellement rester actifs. Par définition, tous les traitements faits par les serveurs de messagerie sont inapplicables aux messages chiffrés.

5.3 Messages signés et chiffrés

Il existe deux méthodes pour fabriquer des messages signés et chiffrés. On peut signer un message chiffré ou chiffrer un message signé. Dans le premier cas, la signature n'est pas confidentielle (vérification des éléments de la signature avant la phase de déchiffrement), dans le second, seul le destinataire du message peut vérifier la signature de celui-ci. Il peut transmettre le message à un tiers sans en altérer la signature.

5.4 Les produits

Outlook, Netscape dans les versions 4.7X sont conformes à S/MIME V2 et inter-opèrent assez bien entre eux. Malheureusement, Netscape 6.0, 6.1 et 6.2 ne supporte pas S/MIME.
Les principales difficultés rencontrées par les utilisateurs concernent la gestion des certificats : longueur des clés utilisables (40 ou 128 bits), importation délicate des autorités de certification et des certificats personnels. Par ailleurs, s'il est vivement recommandé d'utiliser les listes de révocation (CRL) pour toute application à base d'une PKI, dans l'état actuel des choses, cette gestion quasi manuelle avec ces deux produits, est source de nombreuses difficultés souvent liées à l'expiration d'une des CRLs utilisées. Peut-être est-il préférable de réserver l'usage des CRLs sur des serveurs (https ou autre) dont l'administration est assurée par un informaticien ?
Le logiciel Eudora, aussi bien dans les versions 4 et 5 ne supporte pas S/MIME (les versions 5.X supportent IMAPS, POPS et SMTP/TLS). Il est cependant possible de palier cette carence grâce à l'installation d'un « pluggin» ou d'un «proxy» supportant S/MIME. Nous vous conseillons plutôt de reconsidérer le choix de l'outil de messagerie. Nous avons testé une solution de type proxy , *security box mail* dont le principal avantage est d'être complètement indépendante de la version utilisée et même du logiciel utilisé. Voir http://pki.cru.fr/smimeeudora/

Références

Documentation

IGC CNRS
http://www.urec.cnrs.fr/ igc/Doc/IGC_docs.html

IGC du CRU
http://pki.cru.fr/

PKIX Working Group
http://www.imc.org/ietf-pkix/index.html

PKI Forum
http://www.pkiforum.org/

The Open–source PKI Book
http://ospkibook.sourceforge.net/docs/OSPKI-2.4.6/OSPKI/ospki-book.htm

The PKI Page
http://www.pki-page.org/

Serveur de RSA
http://www.rsasecurity.com

The SSL Protocol
http://home.netscape.com/eng/ssl3/ssl-toc.html
http://developer.netscape.com/tech/security/ssl/howitworks.html

Serveur DCSSI
http://www.scssi.gouv.fr/

Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure
http://www.counterpane.com/pki-risks.html

Standards

RFC 2459 - PKIX Certificate and CRL Profile
http://www.ietf.org/rfc/rfc2459.txt

RFC 2510 - PKIX Certificate Management Protocols
http://www.ietf.org/rfc/rfc2510.txt

RFC 2511 - PKIX Certificate Request Message Format
http://www.ietf.org/rfc/rfc2511.txt

RFC 2527 - Certificate Policy and Certification Practices Framework
http://www.ietf.org/rfc/rfc2527.txt

RFC 2560 - PKIX Online Certificate Status Protocol (OCSP)
http://www.ietf.org/rfc/rfc2560.txt

RFCs S/MIME V2:
http://www.ietf.org/rfc/rfc2311.txt
http://www.ietf.org/rfc/rfc2312.txt
http://www.ietf.org/rfc/rfc2313.txt
http://www.ietf.org/rfc/rfc2314.txt

RFCs S/MIME V3
http://www.ietf.org/rfc/rfc2630.txt
http://www.ietf.org/rfc/rfc2631.txt
http://www.ietf.org/rfc/rfc2632.txt
http://www.ietf.org/rfc/rfc2633.txt

Produits

Openssl
http://www.openssl.org/

Modssl
http://www.modssl.org/

Stunnel
http://www.stunnel.org

Sendmail/TLS
http://www.sendmail.org/~ca/email/starttls.html

Postfix/TLS
http://www.hsc.fr/ressources/breves/postfix-tls.html