L’accès sécurisé aux données

Novembre 1999

Serge Aumont
Comité Réseaux des Universités, Rennes
Serge.Aumont@cru.fr

Roland Dirlewanger
CNRS - Délégation Aquitaine et Poitou-Charentes, Talence
Roland.Dirlewanger@dr15.cnrs.fr

Olivier Porte
CNRS – Direction des Systèmes d'Information, Meudon
Olivier.Porte@dsi.cnrs.fr

Les principes

Les applications du chiffrement

Vers une infrastructure de gestion de clefs

Introduction

Il n'est plus d'actualité aujourd'hui de parler de la nécessaire intégration des réseaux informatiques dans les systèmes d'information : c'est désormais un fait acquis.
Mais, faire ce constat ne suffit pas à résoudre les différentes problématiques liées à cette nouvelle dimension et, entre autre, la prise en compte des aspects sécurité.

Dans ce domaine, il y a, d'une manière volontairement schématique, deux composantes relativement aisées à distinguer : la sécurité physique des équipements informatiques (protection contre les incendies, mise en place de politiques de sauvegarde des données, ...) et la sécurité logique des systèmes (vérification de l'intégrité du système, précautions face à l'environnement extérieur, ...).
Jusqu'ici, cette dernière phase s'est traduite principalement par la mise en œuvre d'une génération d'outils bien connus des administrateurs systèmes et réseaux : COPS, Tcp_Wrapper, application de différentes « Checklist Sécurité» sur les systèmes ou les configurations réseau, etc. jusqu'à la mise en place de systèmes purement dédiés à la sécurité comme les « pare-feu » par exemple.

Ces outils répondaient à une double problématique : permettre de manière sûre aux utilisateurs autorisés d'accéder au système d'information et évidemment d'en interdire l'accès pour le reste du monde excepté pour un sous-ensemble « maîtrisé » de certaines informations. Sur ce dernier point concernant la visibilité de certaines informations, la tendance au niveau des moyens utilisés est actuellement clairement en faveur de la messagerie, du Web et des réseaux privés virtuels (VPN), sujets qui seront plus amplement détaillés dans la suite de ce document.

Au début de l'intégration des réseaux dans les systèmes d'information, les outils classiques de la sécurité pouvaient pour une bonne part répondre à cette problématique.
Mais désormais, le problème a changé d'échelle et devant la multiplicité et la complexité des services à mettre en œuvre, les outils standards sont devenus pour une bonne part insuffisants. Sans aucunement remettre en cause leur utilité maintes fois prouvée et toujours d'actualité, il est nécessaire aujourd'hui de développer activement la mise en œuvre de deux autres composantes importantes de la sécurité basées sur des techniques de chiffrement à travers d'une part la gestion de la confidentialité et, d'autre part, les mécanismes d'authentification.

Cette démarche est de plus encouragée actuellement par un contexte législatif sur le chiffrement qui permet le déploiement de telles technologies sans trop de contraintes liées à l'augmentation de la taille des clés.

L'authentification et la confidentialité, qu'il est possible de dissocier fonctionnellement, reposent, comme il l'a été dit, entièrement sur les concepts du chiffrement. Ces concepts sont l'objet du premier chapitre, consacré à éclairer quelques aspects sous-jacents incontournables dans ce domaine.

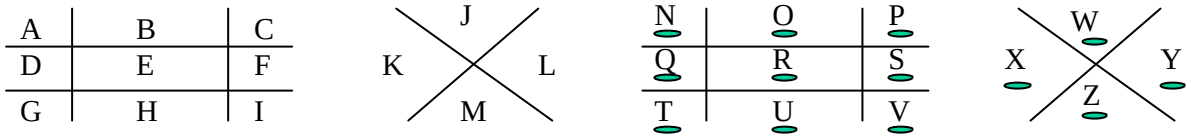
Après avoir posé la problématique du chiffrement, différents exemples illustreront son emploi dans les domaines du Web, de la messagerie et des réseaux privés virtuels (VPN).

Le lecteur intéressé pourra se reporter avantageusement à la bibliographie, non exhaustive ici tant la littérature sur le sujet est abondante et riche en articles et ouvrages.

Principes de base du Chiffrement

1 Bref historique

A l'origine, le chiffrement fut développé dans un contexte militaire afin de pouvoir envoyer des messages sans qu'un ennemi potentiel ne puisse prendre connaissance du contenu.
Déjà, Jules César lui-même envoyait des messages chiffrés à ses correspondants par l'intermédiaire de messagers en lesquels il n'avait aucune confiance.
Son système de chiffrement était alors rudimentaire et consistait à remplacer chaque lettre de l'alphabet par une autre lettre avec un simple décalage de trois caractères.
Ainsi, il remplaça chaque 'A' par un 'D', chaque 'B' par un 'E', ...
Plus tard, les systèmes ne cessèrent de se perfectionner. On peut citer pour servir d'exemple deux systèmes, qui sont celui dit des « francs-maçons » et la technique dite du «carnet de codes ». L'illustration de ces exemples est inspirée de [GARFINKEL1995].
Dans la technique dite des « francs-maçons », on considère la correspondance de code suivante, établie selon des caractéristiques graphiques :



Le mot « secret » devient alors avec cette technique :

La technique des « carnets de codes » encore appelée « chiffrement de Vernam » a longtemps été utilisée dans le domaine de l'espionnage.
Le principe est simple : consigner sur un carnet ou tout autre support une série de nombres aléatoires représentant le « décalage » par rapport à sa valeur de référence que doit subir une lettre pour obtenir son équivalent codé.

Ainsi, en considérant au départ les lettres :

A	B	C	D	et les codes qui leurs sont associées (arbitrairement)
1	2	3	4 ...	

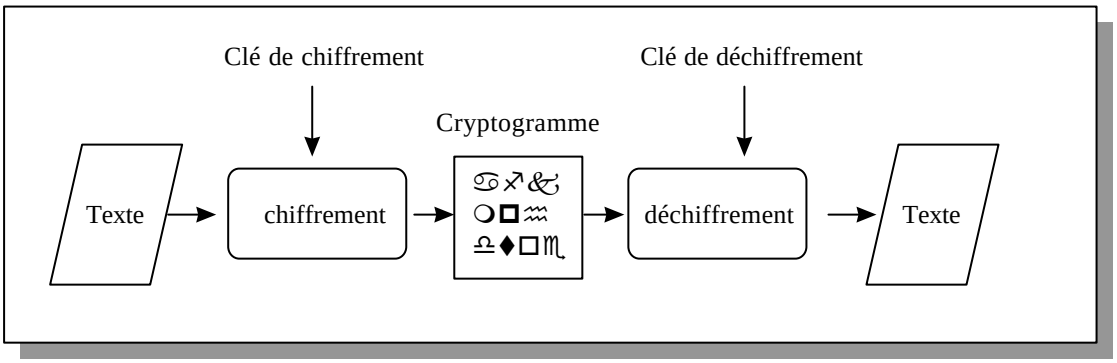
avec en plus sur un support la série 3 2 3 ... générée aléatoirement (le fameux carnet de code), le message de départ 'CDA', pris au hasard, devient 'FFD' après chiffrement. Les deux parties ayant le même carnet de code, il est alors facile de concevoir un dialogue sécurisé entre elles. Les codes de ce type sont inviolables à la première utilisation. Par contre, le fait de les utiliser deux fois permettrait de déceler les redondances, les caractères et leurs occurrences statistiques, ... et, enfin, de « casser » le principe du chiffrement. Il existe bien d'autres techniques historiques de ce type dont la description exhaustive serait ici hors de propos.

2 Définitions et problèmes

2.1.1 Définitions

Le chiffrement consiste à transformer un texte en clair en un texte chiffré. Le déchiffrement, réciproquement, c'est traduire un texte chiffré en clair en connaissant la clé de chiffrement. Décrypter par contre consiste à traduire un texte chiffré en clair tout en ne connaissant pas la clé de déchiffrement. Il est important de bien distinguer ces deux notions car tout l'intérêt de la cryptographie consiste à rendre la première opération facile et la deuxième difficile.
Une mise au point sur la terminologie peut être faite ici : les verbes « crypter » et « encrypter » ainsi que « cryptage » et « décryptage » sont des anglicismes, leurs équivalents français étant « chiffrer », « déchiffrer », ...

Le principe de ce qui vient d'être énoncé peut se résumer ainsi :



On conçoit donc facilement que dans un environnement non sûr où les usagers peuvent être multiples, le chiffrement apporte un niveau de sécurité supplémentaire.
La sécurité des communications, pourvu que la qualité du chiffrement soit suffisante, est alors garantie, même si une tierce personne intercepte le trafic échangé.

Globalement, les objectifs de l'utilisation du chiffrement sont les suivants :

- assurer la confidentialité des données : « La confidentialité est la propriété qu'une information n'est ni disponible ni divulguée aux personnes, entités ou processus non autorisés » [d'après la norme ISO 7498-2 (ISO90)]. En général, l'information n'est disponible et partagée qu'entre les deux parties de confiance que sont l'émetteur et le récepteur définis dans le cadre d'un échange,
- assurer l'intégrité des données : « L'intégrité est la prévention d'une modification non autorisée de l'information » [toujours d'après la norme ISO 7498-2 (ISO90)],
- assurer l'authentification : consiste à vérifier l'identité des différentes parties impliquées dans le dialogue. L'authentification préserve de l'usurpation d'identité et participe de la confidentialité dans le sens où elle assure que celui qui émet est bien l'entité attendue,
- assurer la non répudiation : permet de prouver qu'un message a bien été émis par son initiateur. Le message ne peut donc plus ensuite être dénié par celui qui l'a émis.

Concrètement, il existe deux types de cryptographie : la cryptographie à clé secrète ou la cryptographie à clés publiques et privées (on parle aussi de cryptographie symétrique et asymétrique, ce qui se justifiera plus loin).
Dans les deux cas, la qualité d'un système de cryptographie repose sur le secret de la clé et non sur le secret de l'algorithme et cela aura des conséquences d'une part sur le choix du système de chiffrement et d'autre part sur la manière dont les clés sont gérées, problème qui devra être traité convenablement dans toute mise en œuvre.
Il est d'ailleurs possible de remarquer à ce sujet qu'il est tout à fait envisageable de pouvoir déchiffrer un texte chiffré sans pour cela connaître l'algorithme ayant servi au chiffrement. Baser la sûreté de la méthode de chiffrement sur le secret de l'algorithme paraît donc totalement illusoire.
Cette dernière règle ne vaut d'ailleurs pas seulement pour le chiffrement : la sécurité d'un Système d'Information qui reposerait uniquement sur le secret aurait rapidement à faire face à quelques déboires !

La caractéristique essentielle d'un bon système de chiffrement réside dans le fait qu'il fait apparaître le texte obtenu comme « aléatoire » pour les méthodes basées sur des tests statistiques.
A cela s'ajoute que le système de chiffrement reste fiable tant qu'aucune méthode simple et surtout rapide de « casser » le code n'a pas été mise en évidence.
En effet, le chiffrement, étant basé sur des principes mathématiques (théorie des grands nombres par exemple), reste sûr tant qu'une théorie remettant en cause ces principes n'a pas été découverte.
Or actuellement, il n'y a aucun moyen de prouver qu'un système de chiffrement (à clés secrètes ou à clés publiques/privées) ne contient pas de failles de ce type. Des exemples célèbres ont prouvés que « casser » un code pour un expert restait souvent du domaine du faisable (des exemples existent dans le domaine télévisuel où de nombreuses chaînes « codées » n'ont gardé que très peu de temps leur caractère « confidentiel » : « cassage » de l'algorithme de brouillage vidéo VC-I par exemple).
En fait, la sécurité du chiffrement repose totalement sur la nature et l'implémentation des algorithmes, à corréler avec les puissances de calcul disponibles actuellement. Ces deux domaines étant en perpétuelle évolution, une savante adéquation doit être faite entre la complexité du chiffrement utilisé, le temps de calcul nécessaire à « casser » le chiffrement et le temps estimé pendant lequel les informations doivent être gardées confidentielles.
L'exemple peut être le plus frappant sur le sujet concerne sans doute le DES avec une clé de longueur de 56 bits qui a été déclassifié pour la défense en 1988 alors qu'auparavant, il était considéré comme sûr pour le domaine militaire.

Autre signe des temps, des « concours » sont organisés pour « casser » les principales techniques utilisées : on peut citer les divers « Challenge RSA ».
L'objectif affiché est de tester la sécurité des différents algorithmes de chiffrement autorisés par le gouvernement américain. Le premier concours concernait une clé de 40 bits et le code a été « cassé » en 3 heures environ avec une machine de puissance moyenne (1997).

Dernièrement, la factorisation de clés de 512 bits à été réalisée. Cette opération a vu la participation de six pays : Australie, Canada, Etats-Unis, France, Royaume Uni et Pays-Bas. Trois cents ordinateurs ont été nécessaires dont un Cray. L'opération s'est déroulée en deux phases (choix des polynômes et factorisation proprement dite). La première phase s'est déroulée du 27 avril 1999 au 13 juillet 1999 et la deuxième du 14 juillet 1999 au 22 août 1999. En terme de puissance de calcul, les moyens employés ont été importants : 8000 années de MIPS soit 32 années de calcul pour un processeur à 250 MHz.

Mais le point essentiel à considérer ici reste l'illustration du fait qu'il est possible de remettre en cause la sûreté d'une technique de chiffrement en intervenant dans le cas présent sur les deux facteurs qui sont l'évolution des puissances de traitement et les avancées dans les théories sous-jacentes (Crible algébrique – J Pollard – 1988).

Malgré tout et pour modérer quelque peu ce qui vient d'être dit, dans l'état actuel des connaissances et en sachant bien que les certitudes dans le domaine n'existent pas, il reste que les deux techniques, symétriques et asymétriques, si elles sont bien mises en œuvre, restent une protection tout à fait dissuasive contre « l'espionnage » ou le « piratage » des échanges informatisés en fonction du niveau de sécurité attendu.

De plus, les moyens à mettre en œuvre dans les exemples précédents ne sont tout de même pas à la portée de n'importe qu'elle organisation.

Examinons maintenant plus en détail les principes sous-jacents au chiffrement et les définitions qui en découlent. Le formalisme (simple) qui suit est emprunté aux mathématiques pour exposer avec concision les notions en jeu tout en évitant de trop longues explications.

- **le chiffrement** consiste à déterminer une fonction 'f' qui d'un texte clair 'P' fournit un texte chiffré 'C' à l'aide d'une clé E_K :

$$f_{E\_K} : P \rightarrow C$$

- **le déchiffrement** permet de faire l'opération inverse soit :

$$f_{D\_K} : C \rightarrow P \text{ avec}$$

$$f_{D\_K}(f_{E\_K}(P)) = P$$

Dit plus clairement, à partir d'un texte chiffré C et connaissant la clé de déchiffrement, on retrouve le texte en clair.
Si, dans ce cas, E_K = D_K (même clé pour le chiffrement et le déchiffrement) alors le système est dit à clés symétriques. Dans le cas contraire, le chiffrement est dit à clés asymétriques. La fonction f utilisée peut être différente pour le chiffrement et le déchiffrement : cela dépend des cas (symétriques et asymétriques) et des algorithmes utilisés.

Les propriétés des modèles symétriques et asymétriques sont les suivantes :

- **Modèle symétrique**

  - connaissant P et E_K la clé de chiffrement, il facile de calculer C,
  - connaissant C et D_K (ou E_K puisque dans ce cas E_K = D_K), il est facile de calculer P,
  - connaissant P et C, il est « très difficile » de trouver E_K ou D_K.

Le terme « très difficile » (et par déduction « facile ») dépend principalement de la taille de la clé comme cela sera illustré dans la suite (cas de « l'attaque brutale »).

- **Modèle asymétrique**

    - connaissant P et E_K, il est facile de calculer C,
    - connaissant C et D_K, il est facile de calculer P,
    - connaissant P et C, il est « très difficile » de trouver E_K ou D_K,
    - la connaissance de E_K ne permet pas ou difficilement de connaître D_K et réciproquement.

D'autres notions sont aussi très importantes à connaître en matière de chiffrement. Ces notions concernent les fonctions de hachage à sens unique et les signatures numériques.

- **Fonctions de hachage à sens unique**

Ces fonctions, appelées aussi fonctions de « condensation », condensent en quelque sorte les informations contenues dans un fichier de taille quelconque en un grand nombre qui se révèle alors être l'empreinte du fichier.
Ces fonctions sont telles que si un seul bit du message d'origine est modifié, alors l'empreinte obtenue change radicalement. Le principe est un peu le même que pour la notion de « checksum » mais en beaucoup plus élaboré.

Voyons quelles sont les caractéristiques de telles fonctions de hachage :

Considérons h = H(P) avec

h : empreinte de longueur fixe
H : fonction de hachage
P : texte de longueur quelconque

Les propriétés sont les suivantes :

- connaissant P et H, il est facile de calculer h
- connaissant h et H, il est très difficile de calculer P
- connaissant P, il est très difficile de trouver P' tel que H(P) = H(P')

Les fonctions de hachage les plus connues sont MD2, MD4, MD5 (« MD » pour « Message Digest »), SHA (Secure Hash Algorithm) et SHS (Secure Hash Standard).
Toutes ces fonctions sont quasiment similaires mais plus ou moins rapides. MD5 est plus robuste que MD4 mais environ 33% moins rapide. Actuellement, aucune attaque connue n'est possible sur MD5, si ce n'est la recherche exhaustive de clé, par principe totalement dissuasive.

Concrètement, MD5 (créé par Ronald Rivest) renvoie une valeur de 128 bits sous forme de 32 chiffres hexadécimaux quelque soit le fichier d'origine. Une autre caractéristique notoire de MD5 réside dans le fait que des données d'entrée identiques produisent toujours une même empreinte.

SHS, quant à lui, fournit des empreintes de 160 bits. Sa structure est identique à MD4 et MD5 mais environ 25% plus lent que MD5 (mais potentiellement plus fiable, la taille de la clé étant de 160 bits au lieu de 128 bits).

Il faut bien faire attention, lors de l'utilisation d'une fonction de hachage, d'utiliser des messages suffisamment longs. Dans le cas contraire, une attaque utilisant le «paradoxe des anniversaires », bien connu en probabilités, est toujours possible.

Le « paradoxe des anniversaires » exprime le fait que dans un groupe de 23 personnes choisies aléatoirement, il existe au moins une chance sur deux que deux d'entre elles aient leur anniversaire le même jour, alors que la probabilité est très faible pour qu'une personne du groupe ait son anniversaire le même jour que vous !!

D'où la nécessité, en appliquant ce constat au sujet qui nous intéresse, de prendre en compte des messages assez longs (au minimum 120 bits) avant d'appliquer une fonction de hachage avec un degré de sécurité acceptable, c'est-à-dire permettant d'éviter que deux messages « collisionnent » et aient donc le même résultat de hachage.

- **Signature électronique**

Bien que triviaux, il est bon de rappeler que les objectifs d'une signature électronique sont les suivants :

- vérifier l'intégrité d'un message : détecter toute modification éventuelle
- authentifier de manière certaine la provenance du message

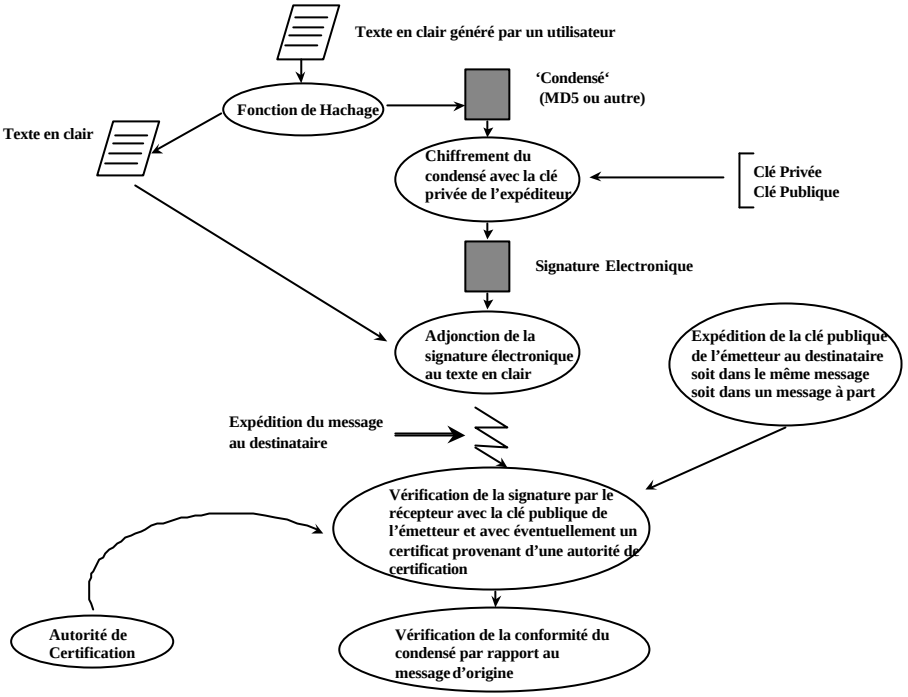
Contrairement au chiffrement qui est utilisé à des fins de confidentialité, les signatures électroniques sont, en quelque sorte, annexées aux données et laissent le texte qui vient d'être signé totalement en clair.
Pour cela, les propriétés suivantes doivent être vérifiées :

- une signature ne peut pas être falsifiée
- une signature donnée n'est pas réutilisable sur une autre document
- un document signé est inaltérable
- une signature ne peut pas être reniée

Le principe est le suivant :

Signature	:	$C = Sign_S(P)$
Vérification de la signature	:	$P = Verif_T(C)$ avec $Verif_T(Sign_S(P)) = P$ et « S » et « T » des clés

On peut illustrer ici de manière très schématique quel est le processus complet de génération d'une signature électronique :



Le mécanisme des clés publiques et des clés privées sera décrit plus amplement dans la suite.

Il existe actuellement beaucoup d'algorithmes de signature. Ces algorithmes sont souvent issus des fonctions de hachage décrites précédemment.
On peut donc encore citer : MD4, MD5 (utilisé par PGP), SHS, ... et DSS (Digital System Standard) dont le principe est basé sur le problème du logarithme discrétisé (trouver l'exposant x tel que $y=g^x \bmod p$, ce qui est un problème considéré comme « difficile » actuellement et dont l'origine remonte au projet CAPSTONE du gouvernement Américain qui visait à développer des standards en matière de chiffrement (1987).

Concrètement, ce qu'il faut retenir sur le mécanisme des signatures, c'est que les algorithmes mis en œuvre sont lents et que souvent la signature se fait en réalité sur un petit nombre de données représentatives.

Enfin, il reste à prendre en compte le problème de la non répudiation. Ce problème a deux dimensions :

- l'émetteur ne peut pas nier l'envoi d'un message
- le récepteur ne peut pas nier la réception d'un message

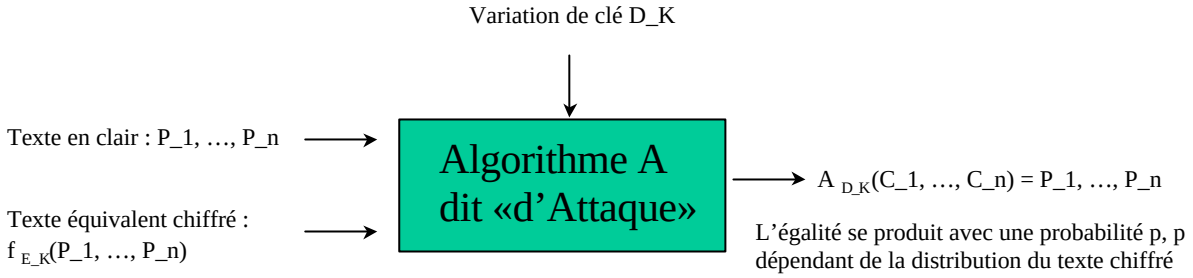
A cela, deux réponses sont envisageables et appartiennent plus au domaine juridique qu'informatique :

- chacune des parties s'engage (de manière contractuelle) à garder secret le moyen d'accès ou de chiffrement (avec conflit possible en cas de dédit de l'une des parties)
- la notarisation : tous les échanges sont consignés chez un tiers de confiance

Désormais, fort des principes de base qui viennent d'être énoncés, il reste à déterminer quels sont les types d'attaques que l'on peut faire subir à un système de chiffrement.

2.2 Quelques principes d'attaques d'un système à base de chiffrement

Modélisons ce que pourrait être une attaque visant à « casser » un code :



En général, en connaissant ou non tout ou partie des textes d'origine en clair, un algorithme « d'attaque A» fera varier la clé 'D_K' sur les textes ou portions de texte chiffrés (C_1, ..., C_n) jusqu'à retrouver le texte en clair (P_1, ..., P_n) et ceci avec une probabilité 'p' plus ou moins élevée.
Avant toute description complète du principe des «attaques », voyons ce qu'est un système de chiffrement parfait.
Dans ce type de système, on considère le texte d'origine, le texte chiffré et les clés comme des octets ou des suites d'octets de longueur m.
La fonction de chiffrement peut se réduire dans ce cas à un simple «OU Exclusif » entre le texte en clair 'P'et une clé 'K':

$$f_{E\_K} = K \text{ XOR } P$$

Si la clé est distribuée uniformément et que le message n'est envoyé qu'une seule fois dans ce mode, la clé n'étant plus réutilisée par la suite, il n'existe pas de moyen trivial de connaître le contenu du message chiffré.
Mais, il est réellement impératif de toujours respecter les deux règles suivantes :

- la clé n'est utilisée qu'une seule fois
- la distribution des clés doit être uniforme et le générateur de nombres aléatoires permettant de créer des clés uniformes doit être le moins « déterministe possible »

Ici, cette génération de nombres aléatoires revêt toute son importance car dans le cas d'une mauvaise distribution de la clé, il est possible de « casser » le code en utilisant des dictionnaires par exemple s'il est connu que le chiffrement s'effectue à partir de mots de la langue courante. De même, des méthodes statistiques peuvent être utilisées, méthodes se basant sur la fréquence d'apparition des caractères dans une langue donnée. Le seul problème inhérent à cette technique du « OU exclusif » réside dans la clé et surtout la détermination et le nombre de clés à utiliser (une par échange). A cela s'ajoute que la perte ou la rupture de la séquence d'échange rompt complètement le processus de chiffrement.

Pour mémoire, il est possible de mesurer la bonne distribution d'une clé en calculant son entropie (l'entropie rend compte de la quantité d'information et est d'autant plus élevée la probabilité d'apparition d'une information est équiprobable). Par exemple, l'entropie d'une clé K sachant que sa probabilité est p vaut -p log (p) (logarithme à base 2).

Voyons maintenant quels sont les types d'attaques qui peuvent être envisagés sur un système de chiffrement :

- **Attaque « brutale »**

Dans ce type d'attaque, il suffit de connaître un message chiffré et d'essayer successivement toutes les combinaisons pouvant conduire à déchiffrer le message : les textes chiffrés C_1, ... C_n sont connus et toutes les clés K sont essayées jusqu'à l'obtention d'une texte en clair.
Le problème principal dans ce cas concerne l'efficacité de la technique mise en œuvre à laquelle s'ajoute la définition et la reconnaissance de ce que peut être un «texte en clair » lorsque l'on ne possède aucune information le concernant, et en principe c'est le cas.
Concrètement, la difficulté d'une telle opération est perceptible immédiatement : une clé de 128 bits offre 2^{128} clés possibles.
Simplement, sur une clé de 64 bits il existe 1.844 * 10^{19} possibilités. En considérant un ordinateur testant 1 milliard de clés par seconde, il faudrait à peu près 584 ans pour trouver la clé avec certitude.

Le problème de la longueur de la clé apparaîtra donc comme un élément essentiel lors du choix d'un système de chiffrement.
D'ailleurs, voici quelques caractéristiques liées aux clés, caractéristiques issues de [ROUSSEAU1996] :

*- Nombre de clés possibles en fonction de la taille de la clé :

	Lettres minuscules (26)	Caractères alphanumériques (62)	Caractères ASCII 8 bits (256)
4 octets	460 000	$1,5 \times 10^7$	$4,3 \times 10^9$
5 octets	1.2×10^7	$9,2 \times 10^8$	$1,1 \times 10^{12}$
6 octets	$3,1 \times 10^8$	$5,7 \times 10^{10}$	$2,8 \times 10^{14}$
7 octets	$8,0 \times 10^9$	$3,5 \times 10^{12}$	$7,2 \times 10^{16}$
8 octets	$2,1 \times 10^{11}$	$2,2 \times 10^{14}$	$1,8 \times 10^{19}$

Le calcul des valeurs de ce tableau se fait simplement. Considérons le cas des lettres minuscules sur 4 octets : pour le 1^{ier} octet, il y a 26 possibilités de choix, et ainsi de suite pour les 3 octets suivants d'ou 26^4 soit 456976 possibilités au total.
Dans ce tableau, les valeurs sont arrondies.

*- Temps de calcul nécessaire dans le cas d'une recherche exhaustive avec la possibilité d'effectuer un million de tentatives par seconde :

	Lettres minuscules (26)	Caractères alphanumériques (62)	Caractères ASCII 8 bits (256)
4 octets	0,5 s	15 s	1,2 h
5 octets	12 s	15 min	13 j
6 octets	5 min	16 h	8,9 ans
7 octets	2,2 h	41 j	2300 ans
8 octets	2,4 j	6,9 ans	580 000 ans

Illustrons ces propos avec le fonctionnement du système de mots de passe du système Unix.
Dans ce cas, le mot de passe est codé sur 8 caractères. Ce mot de passe est transformé en une clé de 56 bits, clé utilisée ensuite par l'algorithme du DES (décrit plus loin).
Un texte de 64 bits à 0 est ensuite chiffré avec la clé de 56 bits. L'opération est répétée 25 fois sur chaque « crypte » obtenu successivement.

Le résultat obtenu consiste en 11 caractères stockés dans un fichier ('/etc/passwd' en général).
L'application d'une variante de « l'attaque brutale » sur les mots de passe Unix pourrait alors se traduire comme suit :

1)- récupération du fichier '/etc/passwd'
2)- Utilisation d'un ou plusieurs dictionnaires (mots usuels, prénoms féminins, ...) à partir duquel on applique l'algorithme de chiffrement qui vient d'être évoqué
3)- Comparaison du résultat obtenu avec le contenu du fichier '/etc/passwd'. En cas d'égalité, le mot de passe peut être considéré comme « cassé ».

Cela illustre le fait, si le besoin s'en faisait encore sentir, qu'il faut absolument choisir avec précaution un mot de passe et surtout mélanger lettres, chiffres et caractères spéciaux sans que le résultat n'apparaisse dans un dictionnaire.

- **Attaque par séquences connues**

Vu l'impasse à laquelle conduit la méthode précédente (à moins d'être relativement chanceux), il est possible pour une tierce personne désirant déchiffrer des communications de « fixer » un certain nombre de paramètres du message d'origine. Ainsi les entêtes de message, les « formatages » de documents normalisés permettent d'aider à la recherche de la clé.

- **Attaques par séquences forcées**

Cette technique se base quelque peu sur les principes de l'attaque par séquences connues. Il suffit dans ce cas de faire en sorte que l'émetteur chiffre à son insu un bloc de données connu, bien entendu connu par celui qui veut casser le code.
Pour contrer cette éventualité, la détection de la modification des données peut alors se faire en compliquant le contenu des données envoyées en utilisant par exemple la technique suivante :

$$T, f_{E\_K}(h(T \text{ Xor } R \text{ Xor } P), R, P) \text{ avec}$$

T : estampille de temps, R : nombre aléatoire
h : fonction de hachage, « , » : symbole de l'opération de concaténation

D'une manière générale, tout ce qui est contrôle de l'intégrité d'un flot de message (pas d'ajouts dans les messages existants, pas de rejeux d'anciens messages et pas de destruction de messages) peut se faire par l'ajout au minimum d'une estampille constituée en règle générale de la date et d'un compteur.

- **Attaque par analyse différentielle**

Cette attaque est encore une variante de l'attaque par séquences connues mais cette fois l'analyse porte sur des textes chiffrés ne comprenant que de petites variations.

Pour résumer, les méthodes utilisées en cryptanalyse se classent en deux types :

- idéale : connaissance d'une texte en clair et de son correspondant chiffré. Il reste ensuite à rechercher la clé selon diverses techniques plus ou moins complexes

- pratique : vu le type de messages généralement échangés (lettre-type, messagerie,...) et la taille de ces messages, des segments de phrases ainsi que leur position peuvent être devinés, et ce, par des essais successifs. Il reste alors à déduire la clé dans un nombre de cas relativement limité

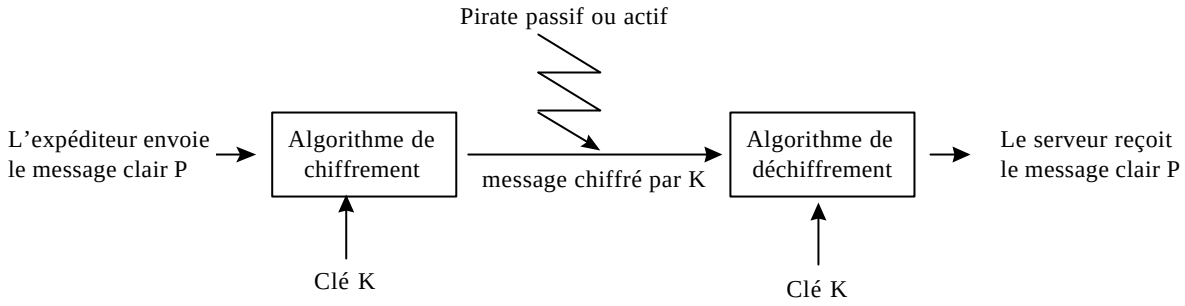
Face à ces différents types d'attaques, il existe donc deux sortes d'algorithme de chiffrement : les algorithmes dits « résistants » et les algorithmes dits « friables ».

Notre intérêt dans ce qui suit va se porter sur le premier type d'algorithme et les applications qui peuvent en être faites dans le cadre de la sécurisation du réseau.

3 Les méthodes de chiffrement

3.1 Le principe de la clé secrète

Ce principe de chiffrement est le plus ancien. Dans ce cas, toute la sécurité du dispositif repose sur le secret de la clé.
Cette clé est la même pour le chiffrement et le déchiffrement.



Le message en clair est transformé avec un algorithme de chiffrement paramètré par une ou plusieurs clés K. Ce chiffrement peut être réalisé par l'utilisateur, le système opératoire, ou un circuit spécialisé.
Le message ainsi chiffré est inexploitable pour quelqu'un qui ne possède pas la bonne clé.

Il existe de nombreux algorithmes à clés secrètes. On peut citer :

- **DES (Data Encryption Standard)**

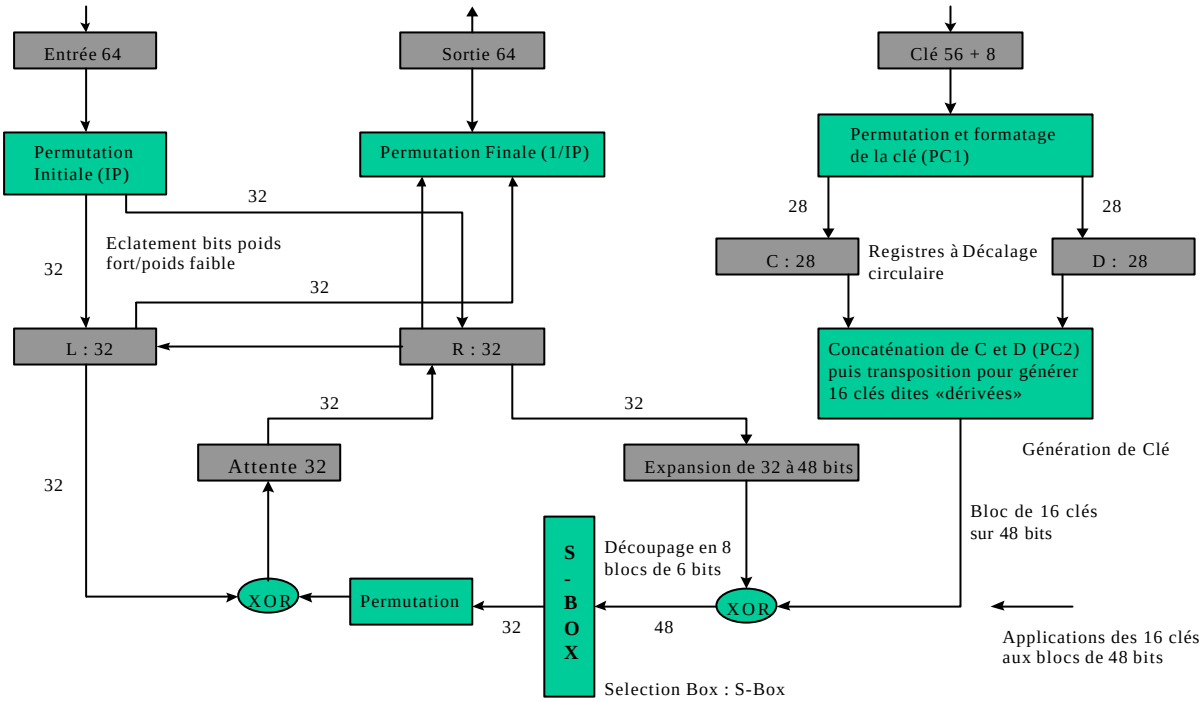
C'est un standard américain adopté en 1977 puis devenu standard ANSI X3.92 en 1981 sous l'appellation DEA (Data Encryption Algorithm). La norme DES prend son origine dans l'algorithme LUCIFER d'IBM. Elle utilise des clés de 56 bits. Cette méthode de chiffrement est actuellement considérée comme peu sûre pour une attaque disposant de très gros moyens informatiques. Elle est tout de même encore souvent utilisée dans les milieux bancaires où le niveau de sécurité apporté par cette technique est jugé suffisant.

- **Triple DES**

Cette technique double la sécurité offerte par le DES grâce à l'utilisation de 3 passes du DES de base avec 2 clés distinctes (ou 3 clés selon les implémentations). La longueur de la clé passe dans ce cas à 112 bits (2 x 56 bits).
Les performances en vitesse de chiffrement sont bonnes et le niveau de sécurité est suffisant pour des applications ne nécessitant pas un très haut niveau de sécurisation (de type militaire). Cette technique est actuellement très utilisée dans les milieux bancaires, surtout lorsque ceux-ci ont déjà l'expérience du DES ce qui conduit alors à une évolution légère des systèmes de chiffrement et des habitudes existantes.
Il faut dire qu'une « attaque brutale » sur une clé de 112 bits nécessite déjà quelques moyens pour le moins significatifs.

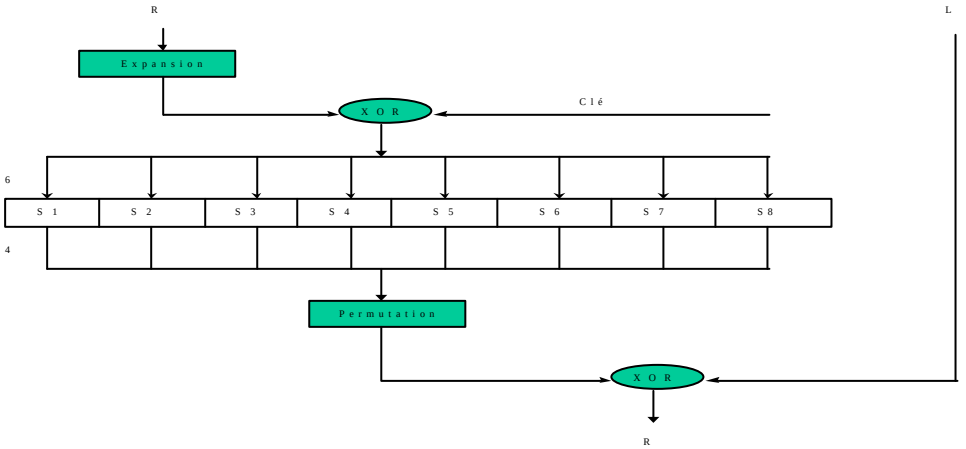
Pour illustration, la logique interne de DES est la suivante :

LOGIQUE INTERNE DE DES



Le but ici n'est pas de décrire le fonctionnement précis du DES mais de donner un aperçu des techniques mises en œuvre et de constater qu'elles reposent majoritairement sur des principes dont certains viennent d'être évoqués.
Globalement, l'algorithme fonctionne sur le principe de permutations, de substitutions et d'additions modulo 2. Le déchiffrement utilise le même principe mais en ordre inverse.

Les substitutions répondant au standard DES sont connues sous le nom de S-BOX et sont décrites par huit tables différentes. Ces S-BOX ont des entrées de 6 bits et des sorties de 4 bits et effectuent des substitutions avec les données en entrée :



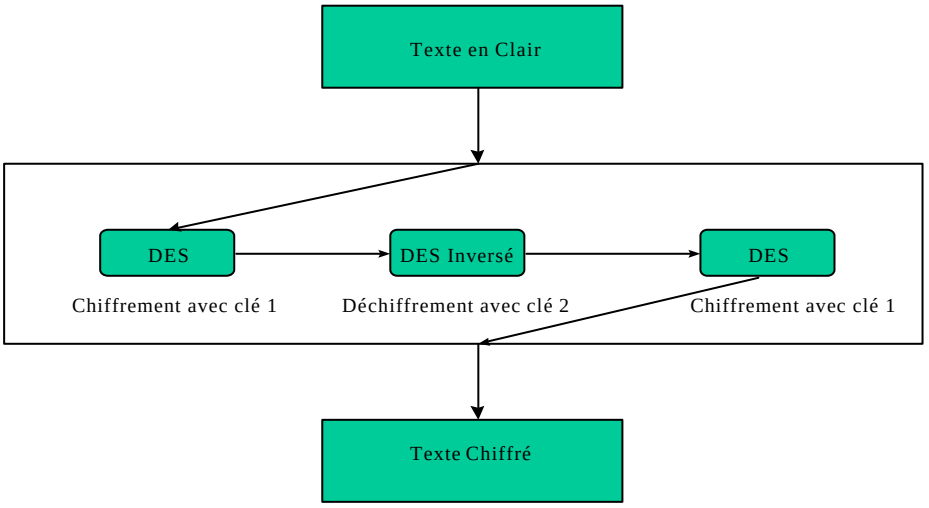
Les S-Box contiennent 8 tables de substitution prédéfinies. Lors de l'arrivée des 6 bits, les bits 1 et 6 sélectionnent la ligne dans la table de substitution et les bits intermédiaires la colonne. Une autre technique mettant en œuvre des permutations peut être utilisée. On parle alors de P-Box (Permutation Box).

Il existe dans la pratique 4 manières différentes de chiffrer et de déchiffrer avec du DES :

- le traitement des données par bloc de 64 octets, ces blocs étant traités les uns après les autres (ECB : Electronic Code Book),
- l'encodage du premier bloc de 64 octets commence par l'application d'un « OU Exclusif » sur le premier bloc chiffré avec un nombre généré aléatoirement, puis l'application d'un « OU Exclusif » entre chacun des blocs à chiffrer à venir et les blocs qui les précédent avec enfin, pour chaque bloc, le chiffrement par l'algorithme du DES après l'opération du « OU Exclusif » (CBC : Cipher Block Chaining),
- les modes OFB (Output Feedback) et CFB (Cipher Block FeedBack) plutôt utilisés dans le cas où les blocs à chiffrer sont d'une taille inférieure à 64 octets.

La méthode ECB est utilisée de préférence lorsqu'il est nécessaire d'accéder à des données sans liens avec les blocs précédents. La méthode CBC est préférable dans les autres cas.

En ce qui concerne le Triple DES et comme cela a déjà été rapidement évoqué, plusieurs techniques sont envisageables : appliquer 3 fois le DES sur les données d'entrée avec 3 clés différentes ou appliquer 2 fois le DES et une fois le DES inversé avec 2 clés différentes.
En général, c'est la deuxième solution qui est employée :



Lors de la publication du DES en 1975, deux critiques avaient alors été formulées à son égard :

- la recherche exhaustive d'une clé (« attaque brutale ») est envisageable,
- des faiblesses ont volontairement été introduites dans les S-Box par IBM sous la pression du Ministère de la Défense Américain (National Security Agency).

En ce qui concerne la première assertion, elle est crédible pour des clés de 56 bits. Par contre, l'utilisation du Triple-DES remet fortement en cause cette possibilité en portant la taille des clés à 112 bits.
La deuxième critique ne semble pas avoir été prouvée formellement bien qu'un grand nombre d'études aient été menées jusqu'à maintenant sur le sujet. La polémique reste donc ouverte.

Au niveau des performances, le DES est plus rapide que des systèmes de chiffrement de type RSA (à clés publiques et privées). Les valeurs estimées sont : 100 fois plus rapide que RSA dans le cadre d'un chiffrement logiciel et environ 1000 fois plus rapide que RSA dans le cadre d'une réalisation matérielle (données datant de 1993).

En conclusion, le DES est probablement peu sûr pour un « attaquant » ayant de gros moyens. La version Triple DES assure malgré tout au niveau de la sécurité des performances acceptables.
Au niveau des performances du chiffrement, l'ordre de grandeur est à peu près celui-ci :

- circuit électronique dédié : environ 1 Gigabit/s
- chiffrement logiciel : de l'ordre du Megabit/s

- **RC2 et RC4**

Ces systèmes sont tous les deux dus à Ron Rivest. Ce sont des alternatives à DES avec des fonctionnalités à peu près équivalentes. Le principe est basé sur un mécanisme à clés variables de 1 à 1024 bits et un flux de nombres aléatoires combinés ensuite par un « OU Exclusif ». La fiabilité de l'ensemble est, bien entendu, fonction de la longueur de la clé choisie.

- **IDEA (International Data Encryption Algorithm)**

Cet algorithme utilise des opérations arithmétiques telles que le « OU Exclusif », l'addition modulo 2^{16} et la multiplication modulo $2^{16} + 1$.
La taille des clés est de 128 bits. L'algorithme, récent (1990), a un degré de sécurité pas encore estimé complètement mais il n'existe pas actuellement de technique ou de machines capables de casser IDEA. Son principe est un peu similaire à DES (fonctionnement en étages) et fonctionne aussi en mode bloc.
Au niveau des performances [ROUSSEAU1996] :

- logiciel : environ 300 Kbit/s sur Intel 80386 à 33 Mhz
- matériel dédié : 50 à 200 Mbit/s à 25 Mhz

D'une manière générale, le DES, le Triple DES, RC2, RC4 et IDEA font actuellement partie des algorithmes considérés comme sûrs et sont souvent utilisés.

- **La démarche AES (Advanced Encryption Standard)**

L'objectif de la démarche AES consiste à fournir un algorithme de chiffrement pour protéger les informations de nature « gouvernementale » (selon la terminologie anglo-saxonne). Cet objectif vise à remplacer le DES qui joue actuellement ce rôle. C'est le NIST (National Institute of Standards and Technology) qui a été en 1997 l'initiateur de cette démarche.

Les principales caractéristiques attendues de ces algorithmes, outre bien sûr leur robustesse aux différentes attaques, sont d'une part la disponibilité de leurs spécifications pour la communauté internationale et d'autre part leur utilisation libre de droits ainsi que la facilité de leur utilisation (chiffrement de blocs de différente taille, rapidité, utilisation dans des ASICs ou des cartes à puce, etc.)

La sélection quant à elle s'est effectuée et continue à s'effectuer au cours de séries de « conférences » (AES Candidate Conference). Quinze candidats se sont déclarés au départ et à l'issue de la troisième « conférence », cinq candidats restent en liste à la date de rédaction de ce document.

Ces cinq algorithmes encore en compétition ainsi que quelques principes les caractérisant sont les suivants :

Algorithme	Initiateur(s)	Quelques principes de base	Longueur de la clé	Références
Mars	IBM	S-Boxes, Xor, rotations, multiplications, etc. fonctions de la clé et des données	128 à 448 bits	http://www.research.ibm.com/security/mars.html
RC6	RSA Laboratories	Xor, Rotations fonctions des données (issu de RC5), etc.	0 à 255 bits	http://www.rsa.com/rsalabs/aes/
Rijndael	Joan Daemen Vincent Rijmen	S-Boxes, multiplications, utilisation de polynômes, etc.	128, 192, 256 bits	http://www.esat.kuleuven.ac.be/~rijmen/rijndael/
Serpent	Ross Anderson Eli Biham Lars Knudsen	Utilisation de S-Boxes, Xor, permutations, etc.	128, 192, 256 bits	http://www.cl.cam.ac.uk/~rja14/serpent.html
Twofish	Bruce Schneier John Kelsey Doug Whitney David Wagner Chris Hall Niels Ferguson	S-Boxes, permutations, matrices, etc.	128, 192, 256 bits	http://www.counterpane.com/twofish.html

Des informations complémentaires sont disponibles aux URL fournies en référence.

Le finaliste devrait être connu courant 2000 et alors officiellement utilisé comme standard dans de nombreux protocoles comme SSL, IPSec, etc.

Problème de la gestion des clés

Une description plus détaillée de cette problématique est fournie dans le paragraphe relatif à la distribution des clés. Seul l'aspect croissance du nombre de clé en fonction du nombre de correspondants est traité ici.

En effet, dans un système à clés secrètes, le nombre de clés augmente selon le carré du nombre de correspondants à établir : $n(n-1)/2$.
Ceci est dû au fait qu'il est nécessaire d'avoir une clé secrète par paire de correspondants puisque toute communication entre individus ou sites doit rester par principe secrète.

Le tableau suivant illustre la croissance du nombre de clés en fonction du nombre de correspondants :

Nombre de correspondants	2	3	4	5	6	7	8	9	10	15	20	50
Nombre de clés	1	3	6	10	15	21	28	36	45	105	190	1225

Dans un système à clés secrètes, le gestionnaire de clés doit donc être un système souple et dont les principales caractéristiques doivent être les suivantes :

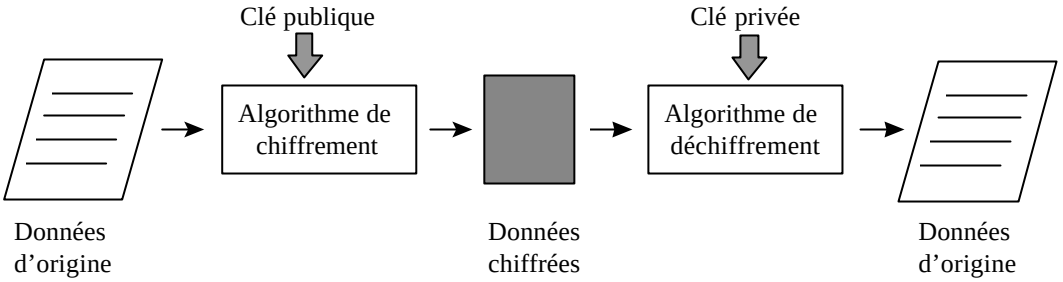
- capacité à gérer un grand nombre de clés,
- gestion des clés de manière centralisée,
- possibilité de gérer un renouvellement fréquent et automatique des clés,
- sécurisation poussée du serveur, seul garant du secret des clés.

Evidemment, la mise en place d'un tel gestionnaire ne résout pas le problème de l'initialisation du mécanisme et toute la difficulté réside dans le fait de faire parvenir sans compromission possible la 1^{ière} clé pour établir le dialogue.

Les principales implémentations industrielles du chiffrement à clés secrètes concernent les boîtiers de chiffrements, les systèmes d'authentification du type Kerberos et d'autres applications répondant à des besoins ou cas de figure spécifiques.

3.2 Le principe clé publique / clé privée

Ce mode de chiffrement prend sa source au début des années 1970. Son principe révolutionne le mode de fonctionnement traditionnel à clé secrète. Voici une illustration de ce mécanisme tel qu'il a été décrit par ses concepteurs :



De nombreux algorithmes utilisent ce principe dont le RSA inventé en 1977 (par R. Rivest, A. Shamir et L. Adleman). Historiquement, ce sont les travaux de Whitfiels Diffie et Martin Hellman qui furent à la base du principe de la clé publique même si le fondement mathématique de la méthode était différent.
Le principe de calcul était basé sur l'exponentiation de grands nombres et un calcul de congruence modulo un nombre premier.
Dans ce cas l'exponentiation est relativement facile mais l'inverse est complexe.

Le mécanisme des algorithmes de type RSA est relativement simple et le suivant :

- choisir 2 grands nombres premiers très grands p et q (100 chiffres minimum),
- calculer le produit n = p * q, n étant appelé le modulo de chiffrement,
- choisir un nombre e, plus petit que n et premier avec ((p-1) * (q-1)),
- on calcule ensuite 'd' tel que d * e = 1 mod ((p-1) * (q-1)). 'd' est calculé avec l'algorithme dit « d'Euclide étendu » (algorithme « d'Euclide étendu » : cet algorithme permet de calculer le PGCD entre 2 nombres et l'algorithme « étendu » permet, lui, de calculer l'inverse d'un nombre modulo n, le cas qui nous préoccupe ici). 'e' et 'd' sont respectivement les composants publics et privés. d est l'inverse de e dans l'arithmétique modulo (p-1)(q-1),
- la clé publique est donnée par le couple (n,e) et la clé privée par d. p et q doivent être tenus secrets ou détruits. Ici, on suppose qu'il est difficile de retrouver la clé privée 'd' à partir de la clé publique (n,e). La fiabilité du système est basée sur ce principe.

Le chiffrement d'un message est alors relativement simple :

- découper le message en blocs de même longueur n (en pratique 200 octets ou plus)
- chiffrer en utilisant la formule : $c = m^e \bmod n$ où 'c' correspond au bloc chiffré et 'm' au bloc d'origine
- pour déchiffrer, il suffit de faire $m=c^d \bmod n$

Voici une illustration simple du principe :

- soit 2 valeurs pour p et q : 47 et 59. Ici les valeurs sont faibles et ne correspondent pas à de grands nombres entiers pour faciliter la compréhension
- on calcule d'abord le produit n = p * q soit 47 * 59 = 2773
- on choisit ensuite une clé e, première par rapport au produit n. Par exemple, e=17
- on calcule ensuite d (par l'algorithme « d'Euclide étendu ») tel que d * e = 1 mod ((p1)(q-1)), soit d = 157. Il est possible de le vérifier en faisant : 17*157=2669 puis en vérifiant que 2669 mod (46 * 58) = 1
- on a donc une clé publique (17,2773) et une clé privée (157,2773). Cette dernière doit être conservée à l'abri de toute indiscrétion

Désormais, il est possible de chiffrer un message. En prenant par exemple les 4 chiffres '0003', le chiffrement de cette valeur donne (0003^{17}) mod 2773 soit 1553. On peut remarquer ici que le chiffrement s'est fait avec la clé publique et que donc seul le détenteur de la clé privée peut retrouver le message d'origine. C'est ce que l'on vérifie en faisant : (1553^{157}) mod 2773 qui donne 3, le message de départ.

En ce qui concerne l'algorithme « d'Euclide étendu », cet algorithme est basé sur l'algorithme d'Euclide « classique ».
Cet algorithme est un algorithme permettant de trouver le plus grand commun diviseur entre deux nombres entiers (PGCD).

Le principe de base est assez simple : si a et b avec a,b $\in$ N et b < a ont un diviseur commun d alors a-b, a-2b, ... sont aussi divisibles par d.
Le cas où a - nb = 0 signifie que a est divisible par b, ce qui résout le problème. La propriété se traduit ensuite par le fait que l'on prend n comme quotient entier de la division de a par b et (a -nb) est alors le reste.

Prenons un exemple : déterminer le PGCD de 522 et 453. Cela donne :

(a) 522	
(b) 453	
(c) 522 - 453 = 69	(a - b)
(d) 453 - 6*69 = 39	(b - 6c)
(e) 69 - 39 = 30	(c - d)
(f) 39 - 30 = 9	(d -e)
(g) 30 - 27 = 3	(e - 3f)
(h) 9 - 9 = 0	(f - g)

La valeur « 0 » arrête l'algorithme et 3 se trouve donc être le PGCD.

L'algorithme « d'Euclide étendu » consiste à utiliser l'algorithme d'Euclide « classique » pour trouver x tel que a * x = 1 mod n.

Voici un exemple avec les mêmes valeurs que celles prises pour RSA, à savoir d * 17 = 1 mod 2668, 2668 correspondant au produit (p-1)(q-1). Quel est dans ce cas la valeur de d ?

Et bien, comme précédemment, on a :

(a) 2668		0
(b) 17		1
(c) 2668 - 2652 = 16	(a - 156 b)	-156
(d) 17 - 16 = 1	(b - c)	157

La valeur « 1 » arrête l'algorithme et la valeur de d vaut alors 157.

Posons-nous maintenant une question toute légitime : pourquoi RSA fonctionne ?
Ce paragraphe, un peu théorique, n'est pas fondamental pour la compréhension de l'ensemble mais illustre malgré tout les principes sous-jacents au RSA.
Avant de commencer, rappelons tout d'abord deux notions essentielles à la démonstration :

- **Fonction d'Euler**:

Soit $\Phi(n)$ le nombre d'entiers inférieurs à n et premiers avec n Dans ce cas, si n est premier alors $\Phi(n) = n - 1$ et si $n = p * q$ avec p et q premiers alors $\Phi(n) = (p-1) * (q-1)$.

- **Petit théorème de Fermat généralisé par Euler**:

Si a et n sont premiers entre eux alors $a^{\Phi(n)} \bmod n = 1$.

Fort de cela et en nommant E la fonction de chiffrement et D la fonction de déchiffrement, vérifions que l'on a bien : D(E(M)) = M.

On a : $D(E(M)) = ((M)^e \bmod n)^d \bmod n$
soit $D(E(M)) = (M^e)^d \bmod n = M^{ed} \bmod n$

On peut noter ici que D(E(M)) = E(D(M)).

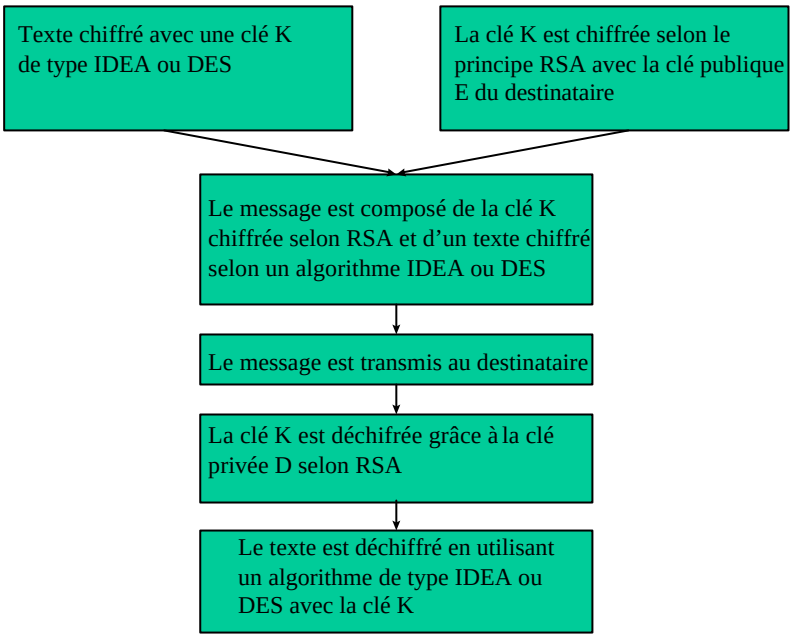
Or, $e.d = 1 \bmod ((p-1)(q-1))$ soit $e.d \equiv 1 + j*z$ avec $z = (p-1)(q-1)$
d'où $M^{ed} = M^{jz} * M \bmod n = M \bmod n$
car $M^{jz} \bmod n = (M^z)^j \bmod n = 1^j = 1$ en appliquant le théorème d'Euler

L'algorithme RSA est donc bien réversible et permet de faire deux choses essentielles :

- chiffrer un message avec une clé publique : seul le détenteur de la clé privée associée pourra alors lire le message
- chiffrer un message avec une clé secrète : tous les possesseurs de la clé publique associée pourront alors lire le message mais auront en plus la certitude que seul le possesseur de la clé privée a pu chiffrer le message

Au niveau des performances, un algorithme de type RSA est environ 1000 fois moins rapide que DES.
Pour 512 bits, l'ordre de grandeur est de 64 Kbit/s et les 1 Mbit/s sont envisageables à court ou moyen terme.

Cette mécanique est très importante à comprendre car elle est à la base de systèmes tels que PGP ou SSL, ce dernier système étant utilisé actuellement dans les navigateurs Web ou la messagerie sécurisée par exemple.
Voici un exemple d'utilisation de clés asymétriques tel qu'il est utilisé dans PGP et combiné avec des clés symétriques :



Une autre propriété utilisée par PGP mérite d'être mentionnée : le principe de la signature déjà évoqué. Ce principe consiste à :

- Générer d'un code qui identifie, par un nombre de taille fixe, le texte lui-même
- Chiffrer cette valeur par RSA avec la clé privée de l'expéditeur, permettant ainsi aux correspondants de vérifier la validité avec la clé publique de l'expéditeur.

Un autre algorithme dû à Diffie et Hellman (1976) et historiquement le premier algorithme traitant du chiffrement à clés asymétriques est fort intéressant à étudier.
Son principe peut se schématiser de la manière suivante :

A et B se mettent d'accord sur 2 nombres q et a très grands

Entité A		**Entité B**
• **Choix, au hasard, d'un grand nombre X_a**		• **Choix, au hasard, d'un grand nombre X_b**
• **Calcul de $Y_a = a^{X_a} \bmod q$**		• **Calcul de $Y_b = a^{X_b} \bmod q$**
• **Envoi de Y_a à l'entité B**	⟷	• **Envoi de Y_b à l'entité A**
• **Calcul de $K_{ab} = (Y_b)^{X_a} \bmod q$ soit $K_{ab} = (a)^{X_a X_b} \bmod q$**		• **Calcul de $K_{ab} = (Y_a)^{X_b} \bmod q$ soit $K_{ab} = (a)^{X_a X_b} \bmod q$**

K_{ab} devient désormais la clé de session entre les entités A et B. Cet algorithme est basé sur la difficulté à calculer des logarithmes discrets.
En effet, un tiers connaissant q et α et interceptant Y_a connaît alors $Y_a = \alpha^{X_a} \bmod q$.
La difficulté consiste alors à calculer X_a !

Pour résumer, les avantages et inconvénients du chiffrement asymétrique sont les suivants :

- difficulté à trouver de grands nombres premiers,
- nécessité de choisir des clés secrètes et publiques assez longues,
- difficulté à réaliser des opérations modulo n rapidement,
- complexité algorithmique de la méthode,
- malgré tout, solution assez générale et sûre si la longueur des clés et les précautions d'emploi sont respectées.

La sûreté d'un algorithme de type RSA dépend en fait de 3 facteurs :

- la non divulgation des nombres p et q,
- la difficulté qu'il existe à factoriser des grands nombres,
- l'absence de méthodes mathématiques actuellement connues permettant de calculer la clé privée d à partir du couple (n,e), à savoir la clé publique.

Les deux dernières remarques sont importantes car actuellement il existe beaucoup de tentatives concernant le « cassage » de ce type de chiffrement.
La technique la plus prometteuse est basée sur la factorisation des grands nombres.
Mais la solution reste encore hors de portée pour des longueurs de clé suffisantes. A titre d'exemple, une clé de 1024 bits nécessiterait environ 280 000 ans de calcul pour un ordinateur ayant une puissance de traitement de 10 000 MIPS [GARFINKEL1995].

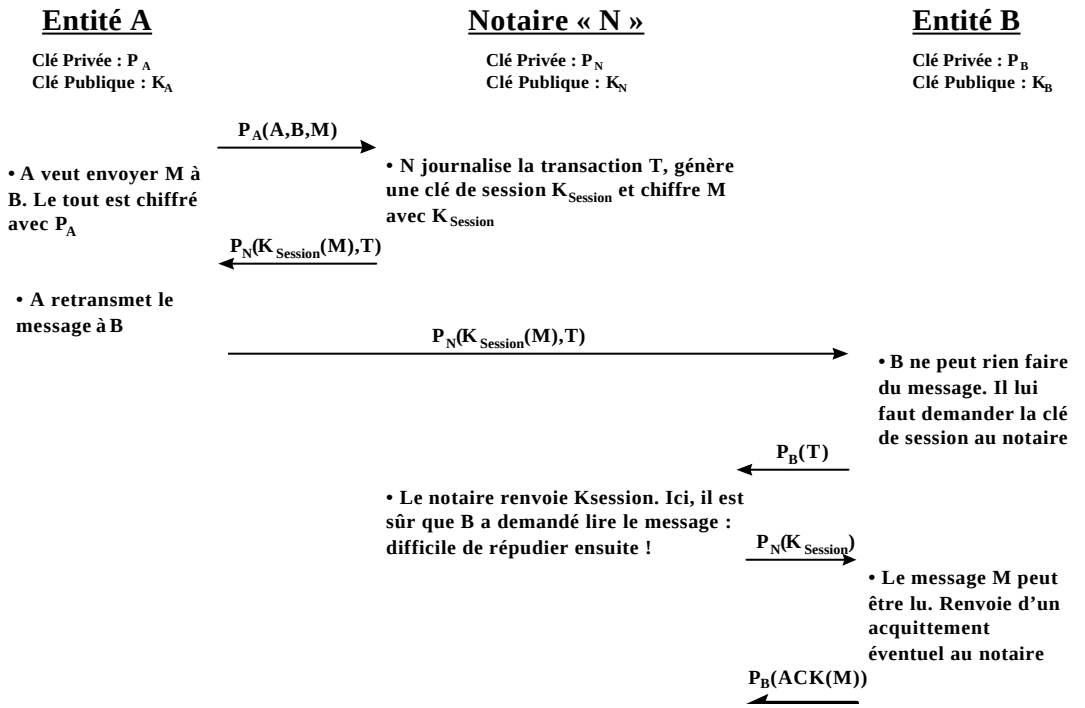
3.3 Notarisation

Le rôle de la notarisation consiste à éviter un déni de responsabilité au cours d'un échange entre deux entités. Cet aspect peut être essentiel dans certains domaines tels que le milieu bancaire par exemple.

Pour éviter tout déni, il existe deux types de réponses :

- la responsabilité du secret des clés : entièrement basé sur la bonne foi des intervenants. Dans ce cas, les conflits ne peuvent se régler en général que devant la justice,
- la notarisation.

Le premier cas ne concerne pas réellement le cadre de notre propos. Le fonctionnement du deuxième cas, quant à lui, peut être illustré comme suit :



Le passage par un tiers, le notaire, permet de consigner les échanges. Cette notarisation est plutôt fiable, sous réserve d'avoir pleine confiance en le notaire qui traite tous les échanges !

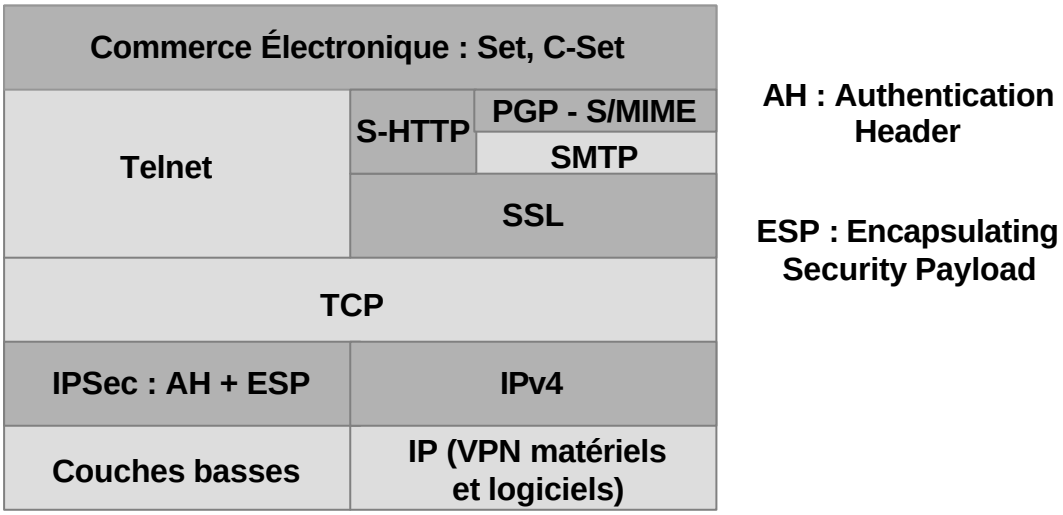
3.4 Le positionnement du chiffrement dans les protocoles réseau

Les principes de chiffrement symétrique et asymétrique qui viennent d'être décrits peuvent être utilisés à plusieurs niveau dans les couches protocolaires. L'emploi du chiffrement à un niveau donné dépend à la fois des performances et des caractéristiques attendues. Ainsi, des vitesses de chiffrement élevées sont requises pour la construction d'un équipement destiné à un VPN alors que pour de la messagerie sécurisée, des aspects plus fonctionnels sont indispensables.

Globalement, il existe quatre possibilités pour mettre en œuvre du chiffrement au niveau des coches protocolaires :

- au niveau matériel,
- au niveau des couches réseau,
- au niveau transport,
- au niveau application.

Le schéma qui suit illustre les différentes possibilités avec quelques protocoles classiques :



Un constat important dans cette répartition est que plus le chiffrement est réalisé dans les couches basses, plus ce chiffrement est transparent pour les applications.
A contrario, plus le chiffrement est positionné dans les couches hautes, plus le niveau de fonctionnalité apporté est potentiellement important mais au détriment de la transparence.

4 De la distribution des clés aux infrastructures à clés publiques

Une infrastructure à clés publiques (PKI) permet de mettre en œuvre une sécurisation des échanges sur un réseau public ou privé dont la sécurité, à la base, n'est pas garantie. Dans ce contexte, le principe consiste à employer des techniques de chiffrement asymétrique dont les clés publiques sont certifiées par une autorité de certification.
Un mode de chiffrement symétrique peut aussi être utilisé (clés de session) mais une fois la session établie par le mode asymétrique (et cela pour des raisons qui vont être détaillées dans la suite).
Pour répondre à la problématique, une PKI doit alors fournir des certificats numériques contenant des clés publiques et un service de stockage, de diffusion et de révocation des clés qui sont employées.

Fonctionnellement, une PKI est donc, de base, constituée des éléments suivants :

- une autorité de contrôle chargée de vérifier la légitimité d'une demande de certificat avant délivrance (niveau organisationnel),
- une autorité de certification avec un gestionnaire de certificats (moyen logiciel) chargée de générer et vérifier les certificats (niveau mise en œuvre),
- un ou plusieurs annuaires (ou tout autre moyen de diffuser les certificats) dont le rôle consiste à stocker et diffuser les certificats (niveau organisationnel et mise en œuvre).

Il est important de noter ici que la problématique de départ à laquelle répond une PKI réside dans la gestion des clés, leur diffusion et le contrôle de leur validité. La gestion des clés est une composante délicate de la mise en œuvre de techniques de chiffrement. C'est pourtant un élément sur lequel repose une bonne partie de la fiabilité du dispositif.

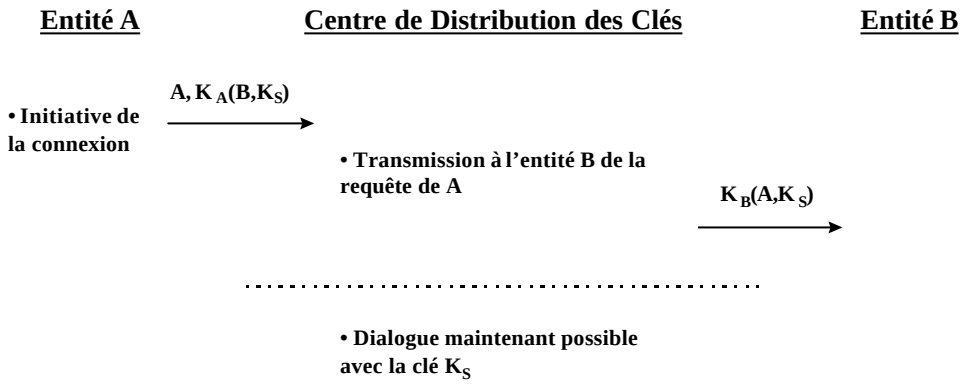
Dans ce qui suit, ces éléments sont repris, brièvement dans le cas des clés symétriques, et plus amplement dans le cas des clés asymétriques, principe au cœur du fonctionnement d'une PKI.

4.1 La gestion des clés privées

Comme il l'a déjà été montré à ce sujet, le nombre de clés à gérer en mode symétrique croit en fonction du carré du nombre de liaisons.
Pour cette raison et de part le fait que la clé doive rester strictement secrète, sa distribution est réellement problématique.
La méthode la plus sûre consiste évidemment à délivrer physiquement les clés sur chaque entité concernée, sans passer par le support d'un réseau informatique. Mais, cette méthode devient très vite inopérante en tenant compte de la possibilité d'augmenter le nombre des entités devant échanger et le nécessaire renouvellement des clés à intervalles réguliers en fonction de « l'usure » de cette clé.

Diverses autres solutions plus opératoires existent pour résoudre ce problème. Elles passent toutes par la mise en place d'une centre de distribution des clés. Kerberos est un exemple d'implémentation souvent utilisé dans le cas des clés secrètes.

Schématiquement, le principe d'un centre de distribution des clés (CDC) est le suivant :



L'entité A voulant dialoguer avec B en utilisant la clé de session K_S commence par envoyer sa demande au CDC avec la clé secrète K_A connue seulement de l'entité A et du CDC. Le CDC transmet alors la demande à l'entité B avec la clé secrète K_B.

A l'issue de ces deux échanges, les entités A et B peuvent dialoguer avec la clé de session K_S.
Ainsi, chaque entité ne connaît réellement qu'une clé : celle qui permet de dialoguer avec le CDC. La charge de la connaissance de toutes les clés est reportée sur le CDC.

Dans ce schéma un peu « idéalisé », il existe une faille classique qui consiste pour l'attaquant à se faire passer pour un tiers au niveau des échanges entre entités : l'attaque du « passeur de seau ». Cette attaque peut être évitée en rendant impossible le rejeu des connexions, ce qui se traduit en fait par une clé de session K_S ayant la propriété d'être unique.
Les techniques les plus utilisées dans ce cas sont l'estampillage de la clé de session par horodatage et ajout d'une valeur aléatoire spécifique à la session (avec un grand espace de numérotation pour rendre très problématique le rejeu).

Ce schéma mettant en œuvre un CDC est à la base de toutes les solutions concernées par le sujet de la distribution des clés symétriques.

4.2 Les cas des clés publiques

4.2.1 Nature des clés et problématique associée

Il est possible de distinguer deux types de clés aux vocations différentes :

- les clés à « court terme » : utilisée une seule fois, pour une session ou pour un seul message et ayant pour vocation d'être ensuite détruites (en général, ce sont tout de même des clés de session symétriques qui sont utilisées dans ce cas),
- les clés à « long terme » : utilisée pour la confidentialité ou l'authentification.

Dans la catégorie des clés à long terme, il est aussi possible de faire la distinction entre les clés de chiffrement et les clés destinées à l'authentification.

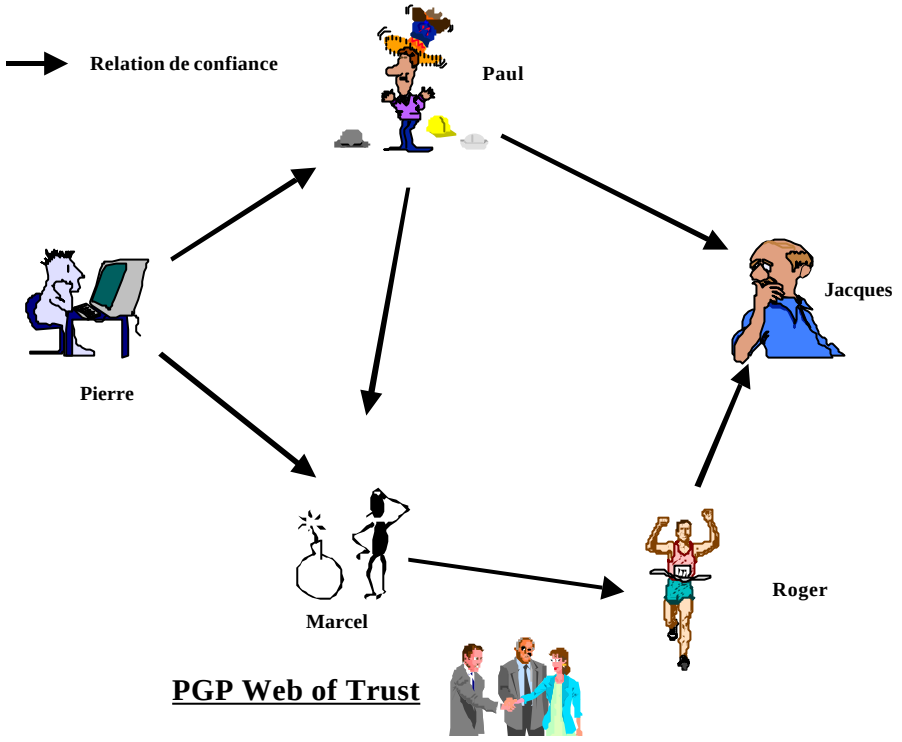
L'utilisation respective de ces clés fait que «l'usure » de chacune est différente. Ainsi la clé de chiffrement est beaucoup plus souvent utilisée que la clé d'authentification et donc en principe à renouveler à une fréquence plus élevée.

4.2.2 « Modèle PGP » et Autorités de Certification

4.2.2.1 Le « modèle » PGP

Plusieurs solutions sont envisageables pour traiter les problèmes qui viennent d'être énumérés relativement à la gestion des clés publiques, à leur validité et au degré de confiance associé. Il existe tout d'abord les solutions basées sur le «modèle » PGP (Pretty Good Privacy) qui consiste à construire un modèle de certification mutuelle dont la fiabilité repose sur la confiance réciproque d'une chaîne de certification.

Ce modèle peut être illustré par l'exemple suivant :



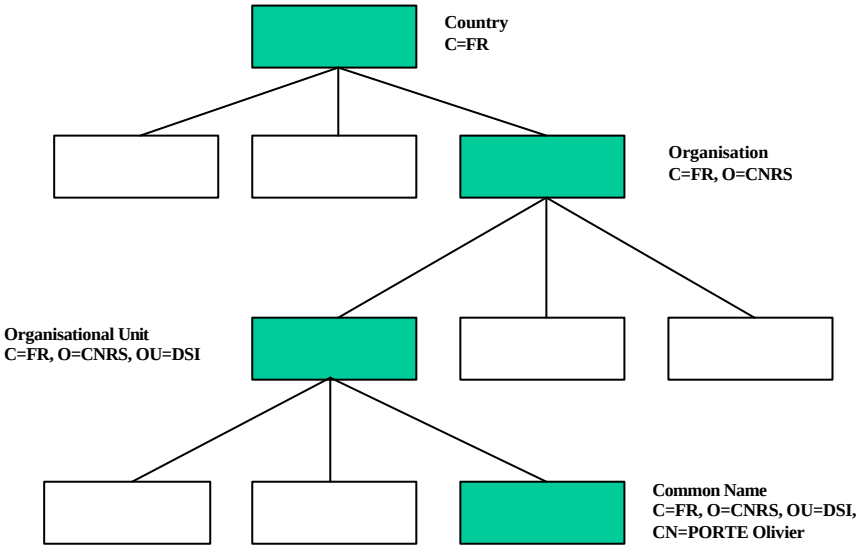
Si dans l'exemple précédent, Jacques connaît Paul et Roger qui connaissent eux-mêmes Marcel et Pierre, alors, dans la logique du modèle, Jacques est «sûr » de la clé publique de Pierre !
Il accepte cette clé publique car il a confiance en Paul et Roger. Mais comment qualifier cette confiance ?
La nature même de cette « chaîne de fiabilité » bâtie sur un principe de confiance réciproque fait que ce modèle est difficilement recevable dans une perspective qui se veut plus générale alors qu'il peut s'avérer acceptable pour un petit nombre d'individus.
De plus, PGP est à l'origine très orienté vers la sécurisation dans un contexte de messagerie (bien que désormais des extensions existent pour le chiffrement de disques durs et une mise à disposition facilitée des clés publiques).

Les PKI n'intègrent donc pas les principes de PGP dans leur mode de fonctionnement.

4.2.2.2 Les autorités de certification

Une autre solution consiste à utiliser une autorité de certification (CA : Certificate Authority).
Par définition, une autorité de certification se pose comme garante de la pérennité du lien entre une clé publique et une entité quelconque propriétaire de ladite clé publique (personne physique, organisme, serveur, etc.).
Pour répondre au problème d'échelle que peut poser le nombre d'entités pouvant posséder un certificat (point d'achoppement du chiffrement symétrique et du modèle PGP), le principe de la mise en place d'une hiérarchie d'autorités de certification est prévue et permet de garantir, en cascade, la validité du lien entre une clé publique et son détenteur. Dans ce cas, l'autorité au sommet de la hiérarchie valide après avoir pris toutes les précautions nécessaires les clés des autorités immédiatement subordonnées et ainsi de suite pour chaque étage de délégation des autorités de certification.

Globalement, une autorité de certification permet donc d'assurer l'authentification d'un serveur ou d'un utilisateur par exemple via une clé publique certifiée et contenue au sein d'un certificat X509.
Historiquement, les certificats étaient utilisés pour protéger l'accès à des annuaires de type X500.



Le DN (Distinguished Name) pour PORTE Olivier est unique : C=FR, O=CNRS, OU=DSI, CN=PORTE Olivier

De ce fait, la structure d'un certificat X509 reflète à travers ses composantes, son lien avec X500. Au niveau du nomage par exemple, X509 utilise le même nomage que X500 à travers la notion de DN (Distinguished Name) qui garantit l'unicité du nomage.

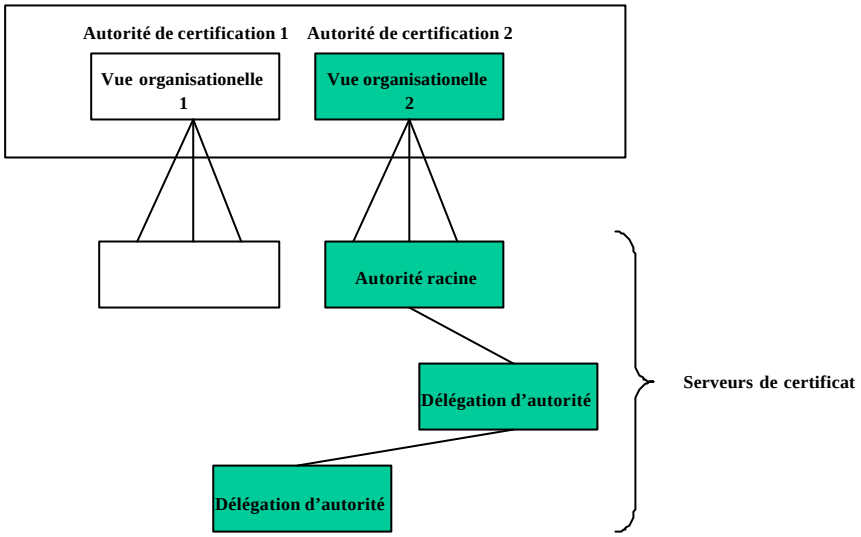
Voici un exemple de nommage au niveau X500 (et donc X509) pour ce qui concerne le DN de l'un des rédacteurs de ce document :

Le DN se retrouve aussi au niveau du certificat X509 dans les champs relatifs à l'autorité de certification et au propriétaire du certificat.

Malgré tout, il est possible de remarquer que la difficulté avec un modèle arborescent de ce type consiste à rendre compte concrètement, selon les cas, du DN d'une personne au sein d'une organisation.

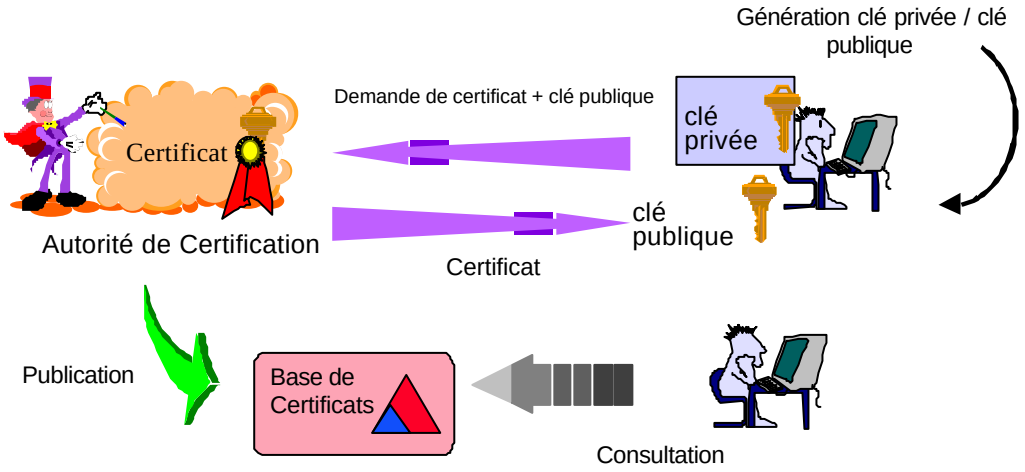
Comment, par exemple, signifier qu'une personne puisse appartenir à deux entités fonctionnelles différentes ou à une entité fonctionnelle et à une entité géographique ?

Actuellement, la meilleure réponse à ce type de question consiste à créer plusieurs vues organisationnelles à l'image de ce qui suit :



Par extension et concernant le problème relatif au stockage des clés, on peut comprendre que le choix du nommage dans le mode X500 influence voire impose la manière dont le stockage des clés peut être envisagé dès maintenant et avec certitude dans le court terme.
Actuellement, ces clés se retrouvent souvent dans des fichiers « plats », dans des bases de données relationnelles, dans des registres de type Windows ou dans des bases de données propriétaires.

L'intérêt d'un nommage de type X500 est alors de pouvoir gérer le stockage et la diffusion des clés via des annuaires de type X500 avec un protocole d'accès comme LDAP (Lightweight Directory Access Protocol).
Concernant l'interaction entre l'autorité de certification et la demande de certificat, la cinématique est la suivante :



L'autorité de certification atteste que la clé appartient bien à celui qu'il prétend être par tous les moyens jugés nécessaires.
Ces moyens peuvent aller de la simple réception d'une demande via messagerie à un contrôle très stricte de la demande effectuée (preuve de l'identité, enquête, etc.).
Le degré de confiance accordé à l'autorité de certification dépend bien évidemment du sérieux et de la rigueur de l'autorité de contrôle et conditionne la fiabilité de la PKI toute entière.

Ce point très important sera repris plus en détail dans la suite du document et mettra en évidence que fondamentalement, utiliser un certificat revient à faire confiance à l'autorité de certification.

4.2.3 Le certificat X509

Le certificat X509 fait l'objet d'une normalisation par l'ISO. Il a été réalisé par l'IETF (Internet Engineering Task Force) et est identifié par un « Distinguished Name » (DN).
C'est concrètement un document électronique attestant qu'une clé publique est bien liée à une organisation, une personne physique, etc.

Ce document électronique contient une clé publique, un certain nombre de champs à la norme X509 et une signature. C'est la liaison des attributs des champs et la clé publique par une signature qui constitue un certificat. Un certificat peut être un faux; c'est sa signature par une autorité de certification (CA) qui lui donne une authenticité.

Globalement, la composition d'un certificats x509 est la suivante :

- Version (v3 actuellement)
- Numéro de série (unique par CA)
- Algorithme de Signature du CA
- Nom du CA (DN)
- Période de validité
- Sujet du certificat (DN)
- Caractéristiques de la clé certifiée :
    - Algorithme utilisé
    - Clé Publique
    - Extensions éventuelles : CRL(liste de révocation), adresse mél, …
    - « Estampille du CA » : Hash signé

En voici un exemple complet pour illustration :

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 7 (0x7)
        Signature Algorithm: md5WithRSAEncryption
        Issuer:    C=FR,    ST=France,    L=Meudon,    O=CNRS,    OU=DSI,    CN=PORTE
Olivier/Email=porte@dsi.cnrs.fr
        Validity
            Not Before: Sep 11 09:49:27 1998 GMT
            Not After : Sep 11 09:49:27 1999 GMT
        Subject:    C=FR,    ST=France,    L=Meudon,    O=CNRS,    OU=DSI,    CN=PORTE
Olivier/Email=porte@dsi.cnrs.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d3:8a:78:15:90:bb:7f:62:50:5d:f4:37:e1:7f:
                    ee:fd:7c:0e:86:c2:1f:50:d9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Netscape CA Revocation Url:
                http://anjou.dsi.cnrs.fr/ca-crl.pem
            Netscape Comment:
                Autorite de Certification CNRS-DSI
    Signature Algorithm: md5WithRSAEncryption
        47:27:8b:b6:4e:7c:22:aa:00:93:9a:c1:e0:04:ad:55:cf:51:
        c7:11
-----BEGIN CERTIFICATE-----
MIIC7TCCAlagAwIBAgIBBzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCRlIx
fKkhXGEkWafhxb3ilCqAFxif4J7DPEX2fgmLEcwDqccR
-----END CERTIFICATE-----
```

En plus des champs « classiques » dont le rôle est relativement clair, un certificat peut posséder un certain nombre d'extensions. Ces extensions sont des couples formés d'un type et d'une valeur avec en plus un « témoin » à caractère optionnel.

Ce « témoin » permet de savoir, lorsqu'il est positionné, si l'extension doit impérativement être prise en compte ou, dans le cas contraire, tout simplement ignorée.
La nature de l'extension peut être diverse (une adresse IP, un alias, etc.) donnant ainsi la possibilité de définir des profils de certificats.

Ces profils peuvent être de plusieurs types :

- banques,
- organismes publics,
- associations,
- etc.

et donc être utilisés à de multiples fins.

Enfin, le dernier point, touchant à la fois aux certificats X509 proprement dit et aux autorités de certification, concerne la révocation des certificats.
Il est prévu dans le certificat lui-même de pouvoir fournir un pointeur sur une CRL (Certificate Revocation List) afin de contrôler auprès de l'autorité de certification la validité d'un certificat.

Le problème est que la gestion de la révocation des certificats via une CRL (qui est en général implémentée sous la forme d'une URL) ne fonctionne pas actuellement de manière satisfaisante. Les raisons de cette mauvaise gestion sont liées à la difficulté de l'implémentation, aux obstacles à l'utilisation, à la vulnérabilité aux attaques par déni de service et aux problèmes de performances.

Cet aspect traduit une réelle difficulté et une contradiction entre les principes de base de la sécurité qui impliquent de la rapidité quant à la suppression des privilèges de tous ordres en cas de problème, et l'impossibilité voire au mieux la difficulté et la lenteur qu'il y a à révoquer un certificat.

Il existe aussi une dualité certaine entre la période de validité constitutive du certificat et la liste de révocation. Une entité voulant s'assurer de la validité d'un certificat devra de toute façon contrôler si possible les deux critères.

Malgré tout, une solution, déjà sous entendue de multiples fois jusqu'ici, consiste à gérer les certificats mais aussi la révocation des certificats au sein d'un annuaire. Mais, là encore, d'autres inconvénients apparaissent tels que la nécessaire consultation de l'annuaire à chaque transaction, la disponibilité de l'annuaire, les performances, etc.

Le deuxième problème lié à la gestion de la révocation des certificats des raisons fonctionnelles telles que le changement d'adresse, de mél et en général de l'un des attributs du certificat. Ces opérations sont courantes et ne souffrent pas non plus des délais trop importants.

Une autre conséquence de ces remarques est la mise en évidence dans tous les cas de la nécessité d'une structure organisationnelle complète et adaptée au contexte pour toute PKI fiable.

Les applications du chiffrement

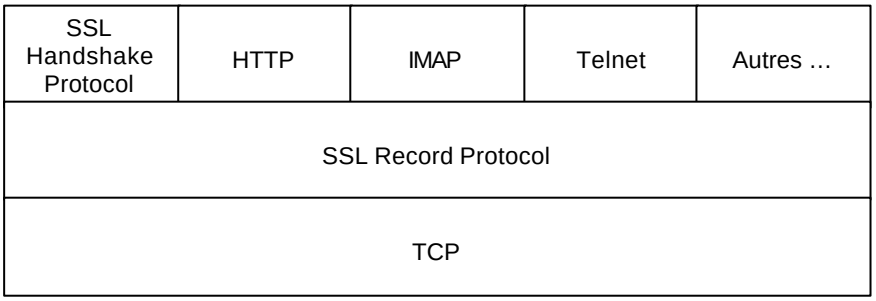
5 Le protocole SSL (Secure Socket Layer)

Le protocole SSL a été développé par Netscape Communication Inc. de façon à améliorer la confidentialité des échanges de tout protocole basé sur TCP/IP. L'application la plus développée est bien sûr la sécurisation des sessions HTTP (*Hyper Text Transfer Protocol*, [HTTP]) entre clients et serveurs WWW.

La première version diffusée est SSL v2.0 [SSLV2] en 1994. C'est un standard propriétaire accompagné d'une implémentation en en ANSI C libre pour un usage non commercial. Netscape a proposé en 1996 une amélioration du protocole, appelée SSL v3.0. La description a fait l'objet d'une proposition de standard Internet. Cette dernière est restée à l'état de *draft*. Elle a toutefois servi de base au développement du protocole TLS (*Transport Layer Security*), en cours de normalisation par l'IETF (*Internet Engineering Task Force*).

5.1 Caractéristiques du protocole SSL v2

L'un des points forts du protocole SSL est qu'il s'insère entre le protocole applicatif et le protocole TCP. Il est donc transparent pour le protocole applicatif, comme le montre la figure suivante :



SSL assure l'authentification des extrémités du circuit de communication ainsi que la confidentialité et l'intégrité des informations transmises :

- le serveur s'identifie auprès du client par un certificat vérifiable par le client. Cette authentification est obligatoire. Le client peut optionnellement s'identifier auprès du serveur par les mêmes mécanismes.

- le flux d'informations transmises entre les deux extrémités du circuit est segmenté en enregistrements (cf. *SSL Record Protocol Specification*). Chaque enregistrement est chiffré par un algorithme négocié au moment de l'ouverture du circuit.

- En outre, chaque enregistrement est accompagné d'une signature MAC (*Message Authentication Code*) garantissant l'intégrité de l'enregistrement.

5.1.1 Ouverture de la session

Les transferts de données dans une session SSL sont caractérisés, on l'a vu par la combinaison d'un algorithme de chiffrement et d'un algorithme de signature. Dans la suite, nous appellerons de telles combinaisons des "spécifications de chiffrement" (*cipher spécifications*, dans le jargon SSL). Par exemple, on utilisera IDEA avec une clé de 128 bits comme chiffrement et MD5 pour le calcul de l'empreinte. Le protocole SSL ne définit pas

une liste exhaustive des spécifications de chiffrement possibles : toute solution est acceptable, du moment qu'elle est connue à la fois du client et du serveur.

L'ouverture d'une session SSL a pour but d'authentifier le serveur auprès du client, de choisir les algorithmes de chiffrement et de signature, d'échanger une clé de chiffrement, et, éventuellement, d'authentifier le client auprès du serveur.

Le déroulement de l'ouverture d'une session SSL v2.0 obéit au *SSL Handshake Protocol*, est résumé dans le tableau ci-dessous :

Séquences du protocole	**Sens du flux** C= client, S= serveur	**Informations échangées**
`client-hello`	C → S	Défi_1, spécifs-chiffrements
`server-hello`	S → C	Id-connexion, certificat du serveur, spécifs-chiffrement
`client-master-key`	C → S	$C_{clé\_publique\_du\_serveur}$(clé_de_session)
`client-finish`	C → S	$C_{clé\_client}$(id-connexion)
`server-verify`	S → C	$C_{clé\_serveur}$(défi_1)
`request-certificate`	S → C	$C_{clé\_serveur}$(type-d'authentification, défi_2)
`client-certificate`	C → S	$C_{clé\_client}$(type_de_certificat, certificat_du_client, données)
`server-finish`	S → C	$C_{clé\_serveur}$(id-session)

5.1.2 Choix des algorithmes de chiffrement et d'empreintes

Le client et le serveur s'accordent sur une spécification de chiffrement lors des deux premières séquences du protocole, les séquences `client-hello` et `server-hello`, décrites ci-dessous.

Le client envoie au serveur la liste des spécifications de chiffrement qu'il supporte. Il accompagne cette liste d'un défi qui sera utilisé ultérieurement (séquences `server-verify`).

Le serveur renvoie un nombre aléatoire, qui sera utilisé comme identificateur de la connexion courante, son certificat X.509, et les spécifications de chiffrement supportées à la fois par le client et le serveur.

Le client est donc en mesure de sélectionner dans cette liste la spécification de chiffrement qui sera utilisée le reste de la session.

5.1.3 Génération des clés de session

Le client génère une clé, dite clé-maître de la session. Il chiffre œtte clé-maître avec la clé publique du serveur et envoie le résultat au serveur (séquence `client-master-key`). Notons que pour s'harmoniser avec les réglementations sur l'exportation des technologies de chiffrement, il est possible que seule une partie de la clé-maître soit chiffrée (par exemple 40 bits, dans le cas des versions exportables de RC4 ou RC2). Le reste de la clé transite en clair.

A partir de la clé-maître, le client et le serveur vont déterminer chacun une clé de chiffrement. Ainsi, les enregistrements émis par le client ne sont pas chiffrés avec la même clé que ceux émis par le serveur. La manière dont sont calculées ces clés dépendent de la spécification de chiffrement choisie à l'étape précédente. Par exemple, dans le cas de

chiffrements symétriques par RC2, RC4 ou IDEA et d'une empreinte par MD5, les clés sont générées comme suit :

- la clé de chiffrement du client est formée de l'empreinte MD5 de la clé-maître, suivie du caractère ASCII "0" (0x30), suivi du défi envoyé par le client lors du `client-hello`, suivi de l'identificateur de connexion envoyé par le serveur lors du `server-hello`. Cette clé est également utilisée par le serveur pour déchiffrer les informations transmises par le client.

- la clé de chiffrement du serveur est formée de l'empreinte MD5 de la clé-maître, suivie du caractère ASCII "1" (0x31), suivi du défi envoyé par le client lors du `client-hello`, suivi de l'identificateur de connexion envoyé par le serveur lors du `server-hello`. Cette clé est également utilisée par le client pour déchiffrer les informations transmises par le serveur.

Le client a terminé sa phase d'initialisation et il le notifie au serveur (séquence `client-finish`). Pour ce faire, il chiffre l'identificateur de connexion et envoie le résultat au serveur.

Symétriquement, le serveur renvoie dans la séquence `server-verify` le résultat du chiffrement du défi proposé par le client dans la séquence `client-hello`. Cette étape permet au client de vérifier qu'il est bien en communication avec le serveur qui détient la clé privée associée à la clé publique présente dans le certificat transmis lors de la séquence `server-hello`. En effet, seul ce serveur est capable de déchiffrer le message contenant la clé-maître et d'en déduire la clé correcte pour chiffrer le défi.

5.1.4 Authentification du client

L'authentification du client est optionnelle. Si le serveur souhaite que le client s'authentifie, il lui transmet une requête de certificat (séquence `request-certificate`). Cette requête contient un défi et le type d'authentification souhaité par le serveur (par exemple, chiffrement du défi par RSA et signature par MD5).

Le client renvoie le type de son certificat, son certificat et la signature d'une combinaison d'informations permettant au serveur de vérifier que le client est bien le propriétaire du certificat.

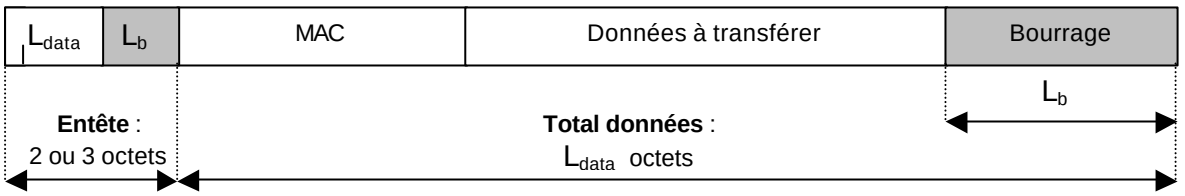
5.1.5 Fin de la phase d'ouverture de la session

L'ouverture de la session est se termine par la séquence `server-finished`. Le serveur génère un identificateur de session et le transmet de façon chiffrée au client. Le client et le serveur peuvent gérer un cache dans lequel ils associent à ce numéro de session la clé-maître.

Pour toutes les connexions ultérieures entre le client et le serveur, les séquences `client-hello` contiendront l'identificateur de session. Si le serveur dispose toujours dans son cache cet identificateur de session, alors le transfert du certificat du serveur (séquence `server-hello`) et le transfert de la clé-maître (séquence `client-master-key`) sont supprimés. La phase d'ouverture de session en est accélérée d'autant.

5.1.6 Le flux des données

Le flux de données entre le serveur et le client est segmenté en enregistrements, selon le *SSL Record Protocol*. Chaque segment est composé d'un entête de 2 ou 3 octets suivi d'un MAC (*Message Authentication Code*), suivi des données à transférer et suivi éventuellement de caractères de bourrage. Ces derniers sont utilisés pour certains algorithmes de chiffrement lorsque la taille des données à chiffrer n'est pas un multiple de la taille du bloc de chiffrement défini par l'algorithme.



Le bit de poids fort du premier octet indique le type d'entête : s'il vaut 1, l'entête contient deux octets et les 15 bits suivants donnent la longueur totale de la partie réservée aux données. S'il vaut 0, l'entête a trois octets, le bit suivant n'a pas de signification en SSL v2.0, les 14 bits suivants donnent la longueur de la partie réservée aux données, les 8 bits suivants donnent la taille de la zone de bourrage.

Le MAC est l'empreinte, calculée selon l'algorithme négocié à l'ouverture de la session, de quatre valeurs : la clé de chiffrement de l'émetteur, les données à transférer, éventuellement les caractères de bourrage, puis le numéro de séquence de l'enregistrement.

La sécurité du protocole SSL réside dans le fait que, mis à part dans les deux premières séquences de l'ouverture de la session, les données transitent toujours de façon chiffrée. En outre, elle sont toujours accompagnées d'une empreinte qui permet au destinataire de vérifier que les données n'ont pas été délibérément altérées en cours de transfert.

5.2 Caractéristiques du protocole SSL v3

Le protocole SSL v3 [SSLV3] corrige un certain nombre de faiblesses de la version 2 et apporte de nouvelles fonctionnalités.

En plus du protocole d'ouverture de session (*SSL Handshake Protocol*), SSL v3 prévoit deux protocoles supplémentaires : le protocole d'échange de spécifications de chiffrement (*SSL Change Cipher Spec*) et le protocole d'alerte (*SSL Alert Protocol*). Tous deux, à l'instar du *SSL Handshake Protocol*, s'appuient sur le protocole de transfert d'enregistrement (*SSL Record Protocol*). Le messages échangés par ces protocoles sont donc chiffrés dès que possible.

Le protocole d'alerte est utilisé pour la transmission de codes d'erreurs entre le client et le serveur. Les codes d'erreurs sont accompagnés d'un indice de sévérité de l'erreur (avertissement, erreur fatale). Le protocole SSL v3 spécifie que si une erreur fatale est détectée lors d'une session, alors la session doit être immédiatement interrompue et doit être effacée du cache des sessions, à la fois sur le client et sur le serveur. Ainsi, une erreur dans le protocole SSL force, lors de la prochaine ouverture de session, la renégociation des spécifications de chiffrement et la génération d'une nouvelle clé-maître.

Le protocole d'alerte est également utilisé pour indiquer la fin d'une session entre le client et le serveur. Cette notion n'existe pas dans SSL v2, où l'une des extrémités indique la fin

d'une session SSL en fermant la connexion TCP/IP sous-jacente. Ce mécanisme rend SSL v2 vulnérable aux attaques par "troncature" : en effet, on peut très bien imaginer un homme du milieu provoquant la fin prématurée d'une session SSL en envoyant les paquets ad-hoc au client et au serveur.

Dans le protocole d'ouverture de session, le serveur et le client ont la possibilité de négocier un algorithme de compression des données avant leur chiffrement. En outre, lorsqu'ils transmettent leur certificat, ils transmettent également la chaîne d'autorités de certification qui valide le certificat en question. De même, lorsque le serveur demande le certificat du client, il lui transmet également la liste des autorités de certification qu'il accepte. Le client est donc en mesure de sélectionner le certificat le plus approprié.

6 Le protocole HTTPS

6.1 Comment utiliser HTTPS ?

Le protocole HTTPS est l'implémentation du protocole HTTP (*Hyper Text Transfer Protocol*) au dessus de SSL.

Du fait que le démarrage d'une session SSL se fait à l'initiative du client, l'utilisation de HTTPS pour le transfert de pages WWW doit être spécifié dans l'URL (*Uniform Resource Locator*). Ainsi, pour une URL de la forme `http://www.domaine.fr`, le client établira une connexion HTTP classique, alors que pour une URL de la forme `https://www.domaine.fr`, le client ouvrira une session SSL avec le serveur, et utilisera ensuite le protocole HTTP au dessus de cette session.

La plupart des navigateurs informent l'utilisateur lorsqu'une page a été obtenue par HTTPS. Ainsi, dans Netscape Communicator, le bouton "Sécurité" s'entoure d'un liseré jaune. Microsoft Internet Explorer affiche un cadenas dans le bas de la page. En outre, lorsqu'en cours de navigation on passe d'une page obtenue par HTTP à une page obtenue par HTTPS, ou inversement, le navigateur affiche un avertissement à l'utilisateur.

Le protocole HTTPS est communément utilisé dans les applications de commerce électronique, au moins dans la phase de transfert des coordonnées bancaires (numéros de cartes bancaires, notamment). Ce n'est bien sûr pas la seule application, comme on le verra dans la suite.

Notons qu'il est fréquent d'entendre parler de "serveur WWW sécurisé" dès qu'il est en mesure d'échanger des pages par HTTPS. Si HTTPS est, probablement, la meilleure solution pour échanger des informations en toute sécurité entre un client et un serveur, il faut garder à l'esprit que seul ce transfert est réellement sécurisé. L'utilisation de HTTPS n'augmente en rien la sécurité du serveur lui-même, ni de l'application basée sur HTTPS : comme on le verra dans le paragraphe concernant les scripts CGI, une application au dessus de HTTPS peut présenter un degré de sécurité acceptable vis-à-vis d'un utilisateur extérieur, mais un degré de sécurité nul pour un utilisateur ayant un compte (par exemple, un *login* Unix) sur le serveur.

6.2 Démarrer un serveur Apache avec mod_ssl

La mise en œuvre de HTTPS dans un serveur WWW est très dépendante du serveur lui-même. Nous avons donc choisi de décrire la procédure pour installer et configurer le module mod_ssl v2.2 [MODSSL] dans un serveur Apache 1.3.x.

L'installation du module mod_ssl nécessite au préalable d'installer une version relativement récente de OpenSSL [OPENSSL]. De plus, mod_ssl s'insère au niveau source dans Apache, si bien que la distribution de mod_ssl est dépendante de la version d'Apache utilisée.

Les exemples ci-dessous décrivent l'installation pour une version 2.3.5 de mod_ssl, s'insérant dans une version source 1.3.6 d'Apache et utilisant la version 0.9.3a de OpenSLL. Les sources de ces diverses distributions sont localisés dans le répertoire `/sources`. Vous pouvez bien sûr changer le nom de ce répertoire en fonction de l'arborescence des sources sur votre système.

Nous utiliserons les conventions suivantes : les commandes commençant par l'invite '%' peuvent s'exécuter dans l'environnement d'un utilisateur standard, les commandes commençant par l'invite '#' doivent s''exécuter en tant qu'utilisateur privilégié (*root*).

Avant toute chose, effectuez une sauvegarde de votre distribution courante d'Apache.

6.2.1 installer OpenSSL

Aller dans le répertoire `/sources/openssl-0.9.3a` contenant la distribution de OpenSSL. Taper les commandes suivantes (pour un système Linux 5.2 ou supérieur) :

```
% ./Configure linux-elf ; make
# make install
```

Ces commandes installent OpenSSL dans le répertoire `/usr/local/ssl`.

6.2.2 installer mod_ssl dans les sources d'Apache

L'installation de mod_ssl s'effectue en deux temps. Dans un premier temps, on modifie les sources d'Apache pour rajouter le module, puis on recompile et on réinstalle Apache.

Allez dans le répertoire `/sources/mod_ssl-2.3.5-1.3.6` contenant la distribution de mod_ssl. Tapez les commandes ci-dessous. Vérifiez les arguments `--with-apache` et `--with-ssl` qui indiquent dans quels répertoires trouver les sources d'Apache et de OpenSSL ; vérifiez également que les répertoires de destination décrits dans le fichier `config.layout` d'Apache sont conformes à votre installation. Dans les exemples qui suivent, nous supposerons que le répertoire contenant les fichiers de configuration du serveur Apache (paramètre `prefix` du fichier `config.layout`) est `/etc/httpd`.

```
% ./configure linux-elf --with-pache=/sources/apache_1.3.6 \
               --with-ssl=/sources/openssl-0.9.3a \
               --prefix=/etc/httpd --enable-shared=ssl \
               --enable-shared=max \
          --with-layout=config.layout:Apache
```

On peut ensuite reconstruire Apache par les commandes suivantes :

```
% cd /sources/apache_1.3.6
% make
# make install
```

6.2.3 configurer Apache et mod_ssl

La procédure d'installation ci-dessus a fabriqué dans le répertoire `/etc/httpd/conf` un fichier d'exemple de configuration qui s'appelle `httpd.conf.default`. Dans les

exemples qui suivent, nous indiquons les lignes minimales à rajouter dans le fichier de configuration `httpd.conf` de façon à démarrer un serveur Apache incluant SSL.

Les directives ci-dessous peuvent être rajoutées n'importe où dans le fichier `httpd.conf`. Nous renvoyons le lecteur au manuel de référence de mod_ssl pour la signification de ces directives.

```
LoadModule ssl_module /usr/lib/apache/libssl.so
AddModule mod_ssl.c

Listen 80
Listen 443

AddType application/x-x509-ca-cert .crt .der

SSLSessionCache dbm:/var/run/ssl_scache
SSLSessionCacheTimeout  300
SSLMutex  file:/var/run/ssl_mutex
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
SSLLog /var/www/logs/ssl_engine_log
SSLLogLevel info

<VirtualHost _default_:443>
DocumentRoot /var/www/ssl_htdocs
ErrorLog /var/www/logs/ssl_error_log
TransferLog /var/www/logs/ssl_access_log

SSLEngine             on
SSLCertificateFile    conf/ssl.key/server.crt
SSLCertificateKeyFile conf/ssl.key/server.key
SSLCACertificatePath  conf/ssl.crt
SSLCACertificateFile  conf/ssl.crt/ca-bundle.crt

SSLVerifyClient       require
SSLVerifyDepth        5

SetEnvIf   User-Agent   ".*MSIE.*"   nokeepalive   ssl-unclean-
shutdown
CustomLog /var/www/logs/ssl_request_log \
  "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
```

6.2.4 générer et installer le certificat du serveur

La séquence d'ouverture d'une session SSL nécessite le certificat du serveur. Ce certificat, obtenu auprès d'une autorité de certification s'installe dans le fichier `/etc/httpd/conf/ssl.key/server.crt` du serveur Apache. De même, la clé privée non chiffrée du certificat du serveur s'installe dans le fichier `/etc/httpd/conf/ssl.key/server.key`.

Bien souvent, on ne souhaite pas faire l'acquisition d'un certificat de serveur auprès d'une autorité de certification reconnue, comme CertPlus, Belsign, Verisign, Thawte, etc. La procédure ci-dessous décrit comment installer un certificat de serveur auto-signé. Bien sûr, ce certificat ne sera pas reconnu comme valide par les clients connectés au serveur. Toutefois, les navigateurs demanderont à l'utilisateur l'autorisation pour accepter ou non le

certificat en question. Notez bien que le nom dans le champ CN (*Common Name*) doit nécessairement être celui du serveur WWW, ou bien être une expression (par exemple : "`*.lbv.fr`") lorsqu'on utilise le même certificat pour plusieurs serveurs WWW.

```
# cd /etc/httpd/conf
# mkdir ssl.key
# /usr/local/ssl/bin/openssl req -new -x509 -days 1000 -nodes \
        -keyout ssl.key/server.key -out ssl.key/server.crt
```

Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Gironde
Locality Name (eg, city) []:Talence
Organization Name (eg, company) [Internet Widgits Pty Ltd]:La Boutique Virtuelle
Organizational Unit Name (eg, section) []:Diffusion Multimedia
Common Name (eg, YOUR name) []:www.lbv.fr
Email Address []:webmaster@lbv.fr

```
# chmod 600 ssl.key/server.key
```

6.2.5 installer les certificats des autorités de certification

Le répertoire `/etc/httpd/conf/ssl.crt` accueille les certificats des autorités de certification reconnue par le serveur WWW. Ce dernier n'est en effet en mesure de valider le certificat d'un client que s'il dispose du certificat de l'autorité qui a signé le certificat de ce client. Par défaut, l'installation de Apache et de mod_ssl n'inclut aucun certificat d'autorités de certification.

Normalement, le répertoire `/etc/httpd/conf/ssl.crt` contient un fichier `ca-bundle.crt` et un fichier `Makefile`, copiés à partir des sources de mod-ssl. Le premier contient les certificats des principales autorités de certification reconnues. C'est la même liste d'autorités que celle qui apparaît dans les navigateurs usuels comme Netscape Navigator ou Internet Explorer. Un serveur HTTPS configuré comme indiqué ci-dessus est donc en mesure de reconnaître les certificats signés par n'importe laquelle de ces autorités.

Si vous disposez du certificat d'une autorité de certification qui n'est pas dans cette liste, vous pouvez le rajouter par la procédure suivante : nommez le certificat avec une extension "`.crt`", placez le dans le répertoire `/etc/httpd/conf/ssl.crt` et exécutez depuis ce répertoire la commande `make` :

```
# cp nouveau-ca.crt /etc/httpd/conf/ssl.crt/nouveau-ca.crt
# cd /etc/httpd/conf/ssl.crt
# make
```

La commande `make` ci-dessus balaie la liste des fichiers ayant l'extension "`.crt`" et ne contenant pas la chaîne de caractères "`SKIPME`". Pour chacun de ces fichiers, elle fabrique un *hash-code* à partir du DN de l'autorité de certification. Elle crée ensuite un lien symbolique entre le fichier d'origine et le résultat du *hash-code*. Cette technique permet d'accélérer la recherche du certificat de l'autorité qui a signé le certificat du client.

Nous attirons l'attention du lecteur sur la confiance qu'il peut ou doit accorder aux autorités de certification contenues dans le fichier `ca-bundle.crt`. La procédure qui permet à l'autorité de certification de vérifier l'identité du requérant est peut-être sujette à caution, surtout dans le cas des autorités qui proposent des certificats gratuitement.

Ainsi, contrairement à un passeport, le certificat d'un utilisateur ne prouve pas son identité. Il prouve simplement qu'une autorité a accepté de certifier l'identité proposée par cet utilisateur. L'identité du détenteur du certificat est donc indissociable de l'autorité de certification qui a signé le certificat.

6.3 Les autorisations

Les paramètres décrits précédemment permettent d'accéder de façon chiffrée et authentifiée à des pages WWW. Dans l'état actuel, l'intérêt se limite à l'authentification du serveur : le client est en mesure de vérifier qu'il dialogue bien avec le serveur qu'il croit, et non pas avec un serveur tiers qui usurpe le nom ou l'adresse Internet du serveur en question.

En ce qui concerne le chiffrement des pages, il est peut utile si les pages en question sont en accès public. HTTPS, grâce aux mécanismes d'authentifications de SSL, peut permettre au serveur d'identifier de façon fiable le client avec lequel il dialogue.

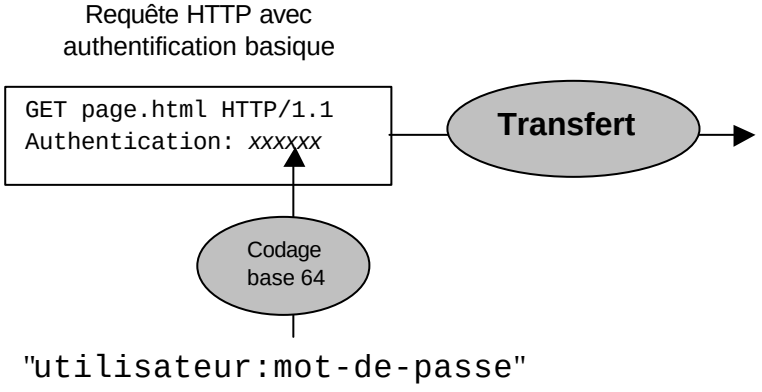
6.3.1 Par mot de passe

On utilise pour HTTPS exactement les mêmes paramètres dans le fichier `httpd.conf` que pour HTTP, par exemple :

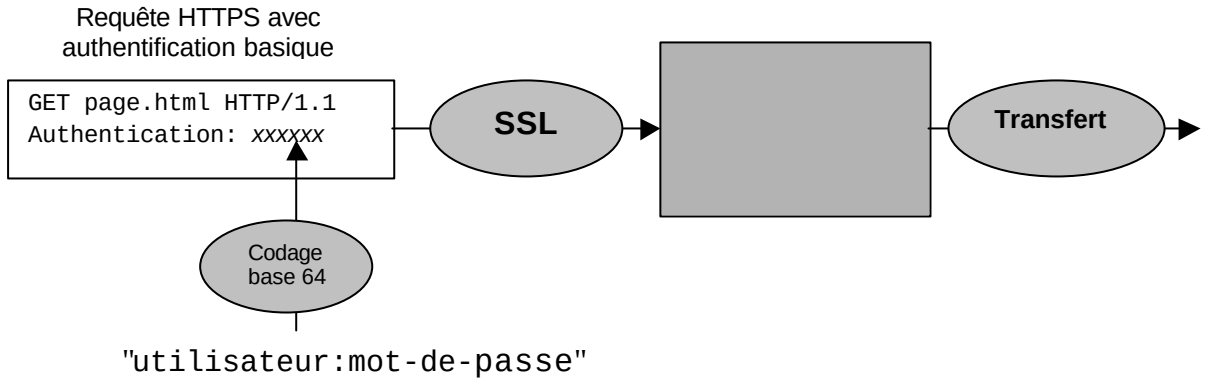
```
<Location /pages/confidentielles>
  AuthType      Basic
  AuthName     Pages-Confidentielles
  AuthUserFile /etc/httpd/conf/httpd.passwd
  require      valid-user
</Location>
```

Les directives ci-dessus indiquent que toutes les pages localisées dans le répertoire `/pages/confidentielles` ne sont accessibles que pour les couples utilisateur, mot-de-passe indiqués dans le fichier `/etc/httpd/conf/httpd.passwd`. Selon la partie du fichier `httpd.conf` où se trouvent ces directives, les transferts des informations d'authentification et des pages auront lieu via HTTP où HTTPS.

Dans le cas d'une authentification classique, dite "basic authentication" dans HTTP, le client encode au format Base 64 le nom de l'utilisateur et le mot de passe associé, et rajoute le résultat dans le champ "`Authentication`" de l'entête de la requête HTTP. Ce codage est quasiment transparent pour toute personne à l'écoute de la communication et récupérer le couple utilisateur, mot de passe est trivial :



Dans le cas de HTTPS, la procédure reste exactement la même, mais l'ensemble de la requête envoyée par le client au serveur est chiffrée, comme le montre la figure ci-dessous :



L'observateur sur le réseau n'est, en principe, ni en mesure de déterminer si la requête contient des informations d'identification, ni de la déchiffrer, ni de la rejouer.

6.3.2 En fonction du certificat du client

La directive "`SSLVerifyClient require`" du fichier `httpd.conf` indique au serveur qu'il doit demander le certificat du client (séquence `certificate-request`) lors de l'ouverture de la session SSL. Le serveur extrait alors les divers champs du DN (*Distinguished Name*) contenu dans le certificat du client. On peut construire des conditions que ces champs doivent satisfaire pour accéder à telles où telles pages.

Les directives ci-dessous permettent de ne donner l'accès à certaines pages qu'aux titulaires d'un certificat indiquant qu'ils font partie de l'organisation "*La Boutique Virtuelle*" en France :

```
<Location /pages/confidentielles>
  SSLVerifyClient  require
  SSLVerifyDepth   5
  SSLRequireSSL
  SSLRequire       %{SSL_CLIENT_S_DN_C} eq "FR" and \
              %{SSL_CLIENT_S_DN_O} eq "La Boutique Vituelle"
</Location>
```

Les variables qu'on peut utiliser sont de la forme "`%{SSL_CLIENT_S_DN_`*xx*`}`" pour décrire le DN contenu dans le certificat du client, et de la forme "`%{SSL_CLIENT_I_DN_`*xx*`}`" pour décrire le DN de l'autorité qui a signé ce certificat. Les *xx* représentent les divers composants du DN, à savoir C pour le pays, O pour l'organisation, L pour la localisation géographique, CN pour le nom et Email pour l'adresse électronique du titulaire du certificat.

Si vous êtes amené à écrire de telles directives portant sur le certificat du client, vérifiez également que le DN de l'autorité qui a signé ce certificat. En effet, plusieurs autorités peuvent avoir signé des certificats différents pour des organisations (champ *O*) ou des personnes (champ *CN*) homonymes.

Bien sûr, la directive "`SSLRequire`" peut devenir très complexe s'il n'y pas de critère simple permettant de décrire les DN autorisés. Apache et le module SSL permettent de

définir des listes exhaustives de DN autorisés, de la même façon qu'on gère une liste de mots de passe pour des accès HTTP classiques. La déclaration est modifiée comme suit :

```
<Location /pages/confidentielles>
  SSLVerifyClient  require
  SSLVerifyDepth   5
  SSLRequireSSL
  SSLOptions       +FakeBasicAuth
  AuthType         Basic
  AuthFile         /etc/conf/httpd/https.passwd
</Location>
```

Les trois dernières lignes indiquent au serveur Apache de simuler une authentification basique et de trouver la liste des comptes utilisateurs et des mots de passe associés dans le fichier `https.passwd`. Ce dernier contient la liste des DN des utilisateurs. On associe à chacun d'eux le même mot de passe qui correspond au chiffrement du mot "`password`". On fabrique donc un fichier du style :

```
/C=FR/O=La Boutique Virtuelle/CN=Joe Dalton:xxj31ZMTZzkVA
/C=FR/O=CNRS/OU=UMR9999/CN=Gus Tave/Email=tave@umr9999.cnrs.fr:xxj31ZMTZzkVA
…
```

6.4 Les scripts CGI et HTTPS

HTTPS est transparent vis-à-vis des scripts CGI ou autres : HTTPS garantit que les requêtes de formulaires sont chiffrées par le client lorsqu'elles sont soumises au serveur et que la réponse du serveur est elle aussi chiffrée. Le serveur transmet au script les versions déchiffrées des paramètres du formulaire. De même, le script génère une page HTML en clair, et c'est le serveur qui la chiffre avant de la transmettre au client.

Dans le cas d'Apache, le serveur appelle le script en enrichissant son environnement d'un certain nombre de variables. Ces variables permettent au script de connaître un certain nombre de paramètres sur la session, notamment la version du protocole SSL utilisé, les spécifications de chiffrement, les informations contenues dans le certificat du serveur (DN du serveur et DN de l'autorité qui a signé le certificat).

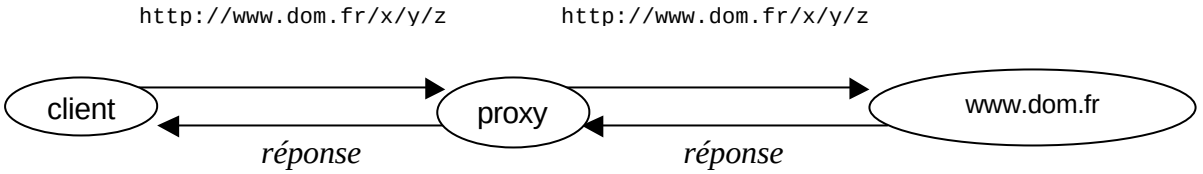
On peut bien sûr, en utilisant les mêmes mécanismes d'autorisations que ceux décrits au paragraphe 2.2.2, faire en sorte que le serveur demande le transfert du certificat du client. Dans ces conditions, l'environnement passé au script CGI ou autre s'enrichit des informations contenues dans le certificat du client. Les variables de l'environnement intéressantes à exploiter sont de la forme "`SSL_CLIENT_S_DN_`*xx*" où *xx* a la même signification que dans le paragraphe 2.2.2.

Avec un tel mécanisme, on peut construire des applications qui requièrent une authentification forte du client, quelle que soit la localisation de ce client sur l'Internet.

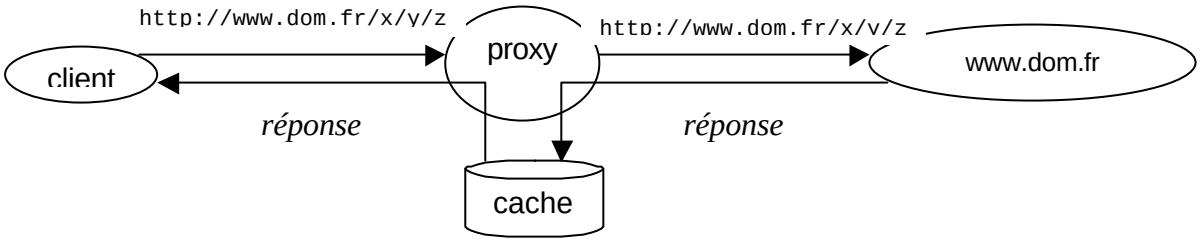
Attention : du côté du script, l'authentification n'étant réalisée que par le contenu de variables d'environnement, il faut vérifier qu'un utilisateur ayant un compte sur le serveur n'est pas en mesure d'exécuter directement le script après avoir reconstitué un environnement valide.

6.5 HTTPS et les serveurs proxys

Un serveur *proxy-HTTP* est un serveur qui relaye des requêtes HTTP pour le compte de clients WWW : le client adresse la requête HHTP au *proxy*, qui la fait suivre vers le serveur de destination, récupère la réponse de ce dernier et la renvoie vers le client :



Parfois, le serveur *proxy* fait aussi office de *cache* : dans ce cas, le serveur *cache* stocke les pages qu'il récupère depuis un serveur distant. Ainsi, si la page requise par un client est dans le cache, le serveur la renvoie directement au client, sinon, il fait suivre la requête.



On utilise les *proxys* dans le cas où les stations du site n'ont pas toutes un accès direct à l'Internet. On utilise les fonctionnalités de cache pour économiser la bande passante vers les serveurs distants.

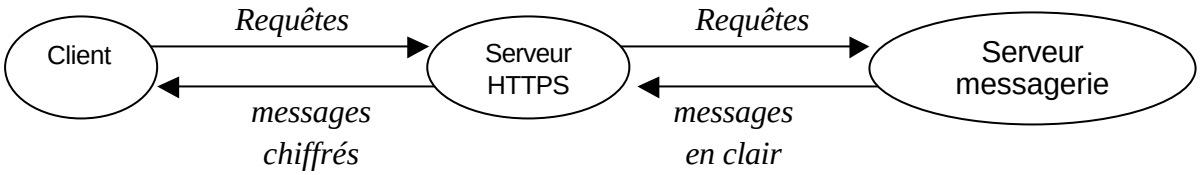
Les solutions de *caches* sont totalement inopérantes en ce qui concerne HTTPS, pour deux raisons. D'une part, chaque enregistrement transmis par SSL contient un numéro de séquence, qui le rend difficilement rejouable. D'autre part, le dialogue entre le client est le serveur est chiffré dans sa totalité, y compris les requêtes soumises par le client. Un serveur *cache*, inséré entre le client et le serveur d'une connexion HTTPS, n'est donc pas en mesure de détecter les requêtes HTTP ni de vérifier si elles sont présentes dans le cache.

Les solutions de *proxys* pour HTTPS sont possible : Apache, configuré avec le module mod_ssl dispose d'un *proxy* capable de gérer les connexions HTTPS pour le compte de clients. Le mécanisme est nécessairement différent de celui d'un *proxy* HTTP classique. En effet, lors d'une connexion HTTPS, le client et le serveur établissent en premier lieu une connexion SSL. Ensuite, le client envoie la requête HTTP sur cette connexion. La requête est donc chiffrée et un *proxy* HTTP classique n'est pas en mesure de s'insérer dans ce schéma. Le proxy HTTPS doit donc reconnaître, soit par un port particulier, soit par tout autre moyen, que le client souhaite établir une connexion HTTPS vers un serveur donné. Le *proxy* HTTPS relaye donc entre le client et le serveur des enregistrements SSL sans les interpréter au lieu de faire suivre des requêtes HTTP.

6.6 Une application : le Webmail

Le Webmail est une application de plus en plus répandue. Elle à tout utilisateur d'accéder à sa messagerie dans des conditions sûres depuis un poste de travail pouvant être localisé dans un environnement peu sûr, comme un cybercafé, une salle en libre service d'un campus universitaire, etc.

L'idée est simple : il suffit de disposer d'un serveur HTTPS et, sur ce serveur, d'une application IMAP cliente du serveur de messagerie du site :



Le serveur HTTPS et le serveur de messagerie peuvent bien entendu résider sur la même machine. Il existe de nombreuses implémentations de *Webmails*. Pour plus d'informations on consultera les pages du Comité réseau des Universités sur ce thème [WEBMAIL].

L'intérêt de HTTPS dans ce schéma est que le transfert du mot de passe IMAP circule de façon chiffrée entre le client WWW et le serveur HTTPS, de la même façon que le contenu des messages.

Cette solution est particulièrement adaptée aux utilisateurs itinérants. En effet, elle permet sans installer de logiciel ou de configuration particulière, d'accéder de façon relativement sûre à sa messagerie.

7 SSL et la messagerie

Les technologies de "*Webmails*" décrites ci-dessus souffrent parfois d'une ergonomie un peu austère du fait de l'utilisation de pages HTML pour accéder aux boîtes aux lettres. La plupart des clients de messagerie, comme Netscape Messenger, offrent la possibilité d'encapsuler le protocole SMTP et le protocole IMAP dans des sessions SSL. L'idée est donc la même que pour HTTPS : le client et le serveur établissent en premier lieu une connexion SSL et déroulent ensuite le protocole classique au dessus de cette connexion. Les protocoles qui en résultent ont reçu des numéros de ports officiels par l'IANA : il s'agit de 993 pour S/IMAP et … pour S/SMTP.

Le fait que SSL soit transparent au protocole sous-jacent fait qu'il est en général facile de modifier un client ou un serveur pour utiliser SSL.

On trouvera dans [WRAPSSL] un "*wrapper SSL*" permettant de sécuriser par SSL tout service réseau implémenté par un serveur lancé par `inetd` sur un système Unix et ce, sans modifier le code source du serveur en question.
Par exemple, pour mettre en œuvre un serveur S/IMAP sur Linux, on procède en trois étapes, comme indiqué ci-dessous.

La première étape est de déclarer le service "`simap`" dans le fichier `/etc/services` en y rajoutant la ligne ci-dessous :

```
simap tcp/993
```

La deuxième étape consiste à compiler et installer le "*wrapper SSL*". Pour ce faire, il suffit de modifier le fichier `Makefile` de la distribution de WrapSSL 8.5 pour rajouter le

répertoire `/usr/local/ssl/include/openssl` dans la liste des fichiers d'inclusion :

```
CFLAGS = -Wall -DSMART -I/usr/local/ssl/include/openssl
```

On lance ensuite la commande "`make`" pour fabriquer le binaire `wrapssl8`. Nous avons choisi e l'installer ensuite dans le répertoire `/usr/sbin`.

La troisième étape consiste à déclarer le nouveau service dans `/etc/ined.conf`. La commande demandant de nombreux arguments, nous avons préféré créé un script trivial qui lance `wrapssl8`. Ce script appelé `/usr/sbin/simapd` contient les lignes suivantes :

```
#!/bin/sh
exec /usr/sbin/wrapssl8 -in -out -verify 0 \
     -key_file /etc/httpd/conf/ssl.key/server.key \
     -cert_file /etc/httpd/conf/ssl.key/server.crt \
     -ca_path /etc/httpd/conf/ssl.crt/ca-bundle.crt \
     /usr/sbin/imapd
```

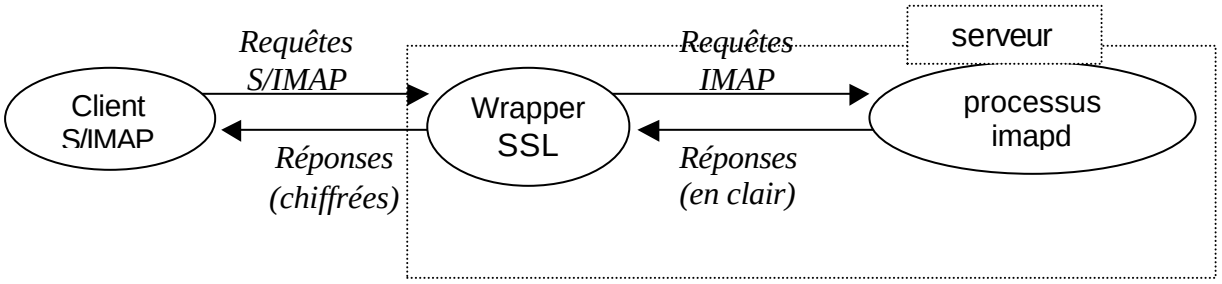
On rajoute ensuite la ligne ci-dessous dans `/etc/inetd.conf` et on relance inetd par un signal 1.

```
simap stream tcp nowait root /usr/sbin/simapd simapd
```

Notons que dans la version 8.5, `wrapssl8` ne gère que le protocole SSLv2. Le client de messagerie doit être configuré pour utiliser SSLv2.

C'est le *wrapper* qui effectue l'ouverture de la session avec le client. Le *wrapper* et le client négocient donc les spécifications de chiffrement.

Le *wrapper* lance ensuite le serveur IMAP classique comme l'aurait fait `inetd`. Le *wrapper* génère des enregistrements SSL pour toutes les données reçues du serveur et les envoie au client. Le *wrapper* déchiffre les enregistrements SSL reçus du client et les transmet en clair vers le serveur IMAP. Dans une configuration classique, le *wrapper* SSL et le serveur IMAP tournent sur la même machine. Les messages ne transitent en clair qu'à l'intérieur de cette machine :



8 Le format S/MIME

Le format S/MIME [SMIME] est une extension du format MIME [MIME]. Il définit un certain nombre de types MIME pour décrire un message chiffré ou un message signé.

S/MIME s'appuie sur PKCS7 [PKCS7] qui décrit une syntaxe générale pour des données auxquelles on a appliqué des techniques de cryptographie, c'est-à-dire essentiellement, la signature digitale et le chiffrement.

Les clients de messagerie Netscape Messenger et Microsoft Outlook sont en mesure de générer et interpréter correctement des messages au format S/MIME. On regrettera que la version 4.0 d'Eudora ne dispose pas d'extensions pour S/MIME.

Les exemples ci-dessous décrivent l'utilisation de S/MIME avec Netscape Messenger. L'utilisation avec Microsoft Outlook est tout à fait similaire et laissée à titre d'exercice.

8.1 La signature d'un message

S/MIME définit plusieurs format pour des messages signés. Le plus couramment utilisé correspond au type MIME `multipart/signed`. Dans ce format, le message transmis est la réunion de deux parties MIME. La première correspond au message à signer. Il a un type MIME et un encodage classique. La deuxième partie est l'empreinte correspondante chiffrée avec la clé privée du signataire. Elle contient également le certificat du signataire et les informations sur les algorithmes utilisés pour calculer l'empreinte du message et chiffrer cette emreinte. Plus précisément, cette partie est un objet PKCS7 de type `signedData` avec un champ `contentInfo` vide. On lui attribue le type MIME `application/pkcs7-signature`, comme le montre l'exemple suivant :

```
MIME-Version: 1.0
Subject: message =?iso-8859-1?Q?sign=E9?=
Content-Type: multipart/signed;
       protocol="application/x-pkcs7-signature"; micalg=sha1;
       boundary="----------ms27BFF969A2EBC8AE8D0C0A62"

This is a cryptographically signed message in MIME format.

--------------ms27BFF969A2EBC8AE8D0C0A62
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

Ceci est un message sign=E9.
--------------ms27BFF969A2EBC8AE8D0C0A62
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIIF2wYJKoZIhvcNAQcCoIIFzDCCBcgCAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3DQEHAaC
C
…
cT/jBEirPG0M3z+4O9idrhSHftctWDCgaxyo1tNw4pQ/rGylXkcCgrfLGBuT
--------------ms27BFF969A2EBC8AE8D0C0A62--
```

L'intérêt de ce format est que le message d'origine reste lisible pour un agent de messagerie qui ne connaît pas le format S/MIME. Par exemple, dans le cas d'Eudora Pro, le destinataire d'un message signé est en mesure de le lire. Il voit un document attaché de type `application/pkcs7-signature`, (suffixe `p7s`) qu'il ne peut décoder.

8.2 Le chiffrement d'un message

Le type MIME `application/pkcs7-mime` est utilisé dans le cas d'un message ou d'une partie de message chiffré. La partie chiffrée du message est un objet PKCS7 de type

`envelopedData`. Il correspond au résultat du chiffrement du message par la clé publique du destinataire.

L'exemple ci-dessous montre le format S/MIME correspondant à un message chiffré.

```
MIME-Version: 1.0
Subject: message =?iso-8859-1?Q?chiffr=E9?=
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

MIAGCSqGSIb3DQEHA6CAMIACAQAxgf0wgfoCAQAwgaMwgZ0xCzAJBgNVBAYTAkZSMRAwDgY
VQQHEwdUYWxlbmNlMTgwNgYDVQQKEy9DTlJTIC0gRGVsZWdhdGlvbiBBcXVpdGFpbmUgZXQ
…
9RZB2CTfXWCGHOCw0ZdVxfmK9zJoUSkSjwQoXCHb2Y8b7UIyFPW2W2lYB4G53KkMk9PR+17
wqA8znuswA55CzxsIgQIJ0L/K5GtRn4ECE4tyuUVF1zwAAAAAAAAAAAAA==
```

8.3 Le chiffrement et la signature

Dans le cas d'un message à la fois signé et chiffré, il y a deux possibilités. Soit on chiffre le message signé, soit on signe le message chiffré. Les deux solutions ont chacune leur intérêt. Dans le premier cas, la liste des signataires est masquée par le chiffrement. Dans le deuxième cas, on peut vérifier les signataires sans être en mesure de déchiffrer le message.

C'est donc l'application de messagerie qui décide quel mécanisme employer. Dans le cas de Netscape Messenger, un message est d'abord signé et le résultat est chiffré.

8.4 Installation du certificat de l'utilisateur

Pour pouvoir utiliser S/MIME, il est nécessaire d'être titulaire d'un certificat d'authentification. L'autorité signataire de ce certificat doit être connue de tous les correspondants. Dans le cas contraire, le destinataire d'un message ne pourra pas être en mesure de vérifier la signature de l'émetteur du message.

En principe, un certificat d'utilisateur ne contient que la clé publique. La manière habituelle de transmettre à l'utilisateur son certificat et sa clé privée est de chiffrer l'ensemble par un algorithme symétrique standard (triple-DES, par exemple) et de communiquer la clé de chiffrement par un moyen sûr (communication téléphonique chiffrée, courrier en main propre, etc.) Le format dans lequel est chiffré le certificat et la clé privée s'appelle PKCS12 [PKCS12].

Importer un certificat au format PKCS12 depuis un navigateur standard est trivial.

Depuis Netscape Navigator, on clique sur le bouton "*Sécurité*", puis sur le lien "*Vos certificats*". Dans la fenêtre qui s'affiche, on clique sur le bouton "*Importer un certificat*". On sélectionne alors le fichier contenant le certificat. L'importation se termine après une suite de bouton "*Suivant*". Notons que l'un des tableaux affichés demande pour quels types de services on pense utiliser le certificat. Nous vous recommandons de cocher toutes les cases.

Dans la phase d'importation de votre premier certificat, Netscape vous demandera un mot de passe pour protéger les données confidentielles, notamment la clé privée. Netscape stocke ces données dans un fichier chiffré qu'il appelle Netscape Communicator DB. Ce mot de passe sera utilisé pour toutes les opérations liées à l'authentification et à la sécurité

des transferts, à la fois pour Navigator et pour Messenger. Il est demandé une fois par session de Netscape. Dans les exemples qui suivent, la fenêtre vous demandant le mot de passe en question ne sera jamais mentionnée.

Pour Microsoft Outlook et Explorer, les clés privées sont stockées dans la base de registres propre à l'utilisateur, et sont donc protégées par le mot de passe de connexion Windows.

8.5 Comment obtenir le certificat d'un correspondant ?

Si on désire établir une communication chiffrée avec un correspondant, il est nécessaire de détenir une copie de certificat. En effet, ce dernier contient la clé publique de son titulaire, et c'est avec cette clé qu'on va chiffrer le message.

Les paragraphes suivants décrivent les deux manières usuelles de récupérer le certificat d'un correspondant.

8.5.1 En recevant un message signé

Dès qu'on dispose d'un certificat, il est possible d'envoyer des messages signés à des correspondants.

Depuis Netscape Messenger, lorsqu'on tape un message, il suffit d'utiliser le bouton "*Options d'envoi de messages*" situé à gauche de la liste des destinataires du message. On coche la case "*Signé*". La signature électronique est rajoutée au message.

Dans le cas où l'on est titulaire de plusieurs certificats, Netscape Messenger permet de spécifier le certificat qui sera utilisé pour la signature. Pour cela, cliquez sur le bouton "*Sécurité*", puis sur le lien "*Messenger*". Sélectionnez le certificat dans le paragraphe "*Certificat pour vos messages signés et chiffrés*".

Comme on l'a vu précédemment, la signature électronique contient une copie du certificat de l'émetteur du message. Les outils de messagerie S/MIME stockent le certificat de tout émetteur de message signé.

Dès lors qu'on a reçu un message signé d'un correspondant, il est possible de démarrer une communication chiffrée.

8.5.2 En interrogeant un annuaire

Il est possible de demander le certificat d'un utilisateur à un annuaire en utilisant le protocole LDAP [LDAP]. Dans Netscape, on procède comme suit. On clique sur le bouton "*Sécurité*" puis sur le lien "*Autres certificats*". Dans la fenêtre qui s'affiche, on clique sur le bouton "*Rechercher dans l'annuaire*". On sélectionne un annuaire, on tape l'adresse de messagerie recherchée et on clique sur le bouton "*Rechercher*".

Il est possible de rajouter un annuaire LDAP dans la liste. Par exemple, si l'annuaire à rajouter est `ldap.lbv.fr`, on demande à Netscape Navigator d'afficher l'URL `ldap://ldap.lbv.fr`. Netscape Navigator demande si on souhaite rajouter cette adresse dans la liste des annuaires.

8.6 Exporter un certificat

Compte tenu des évolutions prévisibles de la législation française en matière de preuve, le certificat d'un utilisateur, dès lors qu'il est certifié par une autorité de qualité, est assimilé à sa signature originale. Détenir le certificat d'un tiers et sa clé privée est donc une

responsabilité que les autorités de certification ne souhaitent pas nécessairement assumer. Pour la même raison, les logiciels courants protègent les clés privés avec des mécanismes de chiffrement optimaux.

Il est donc nécessaire de prévoir une procédure de sauvegarde des certificats d'utilisateurs. La manière optimale est de sauvegarder le certificat sur un support amovible (disquette, ou mieux encore cédérom). La sauvegarde est particulièrement recommandée dans le cas où les procédures de transferts entre l'autorité de certification et l'utilisateur n'ont pas utilisé de support physique ou bien lorsque l'utilisateur change de navigateur ou de station de travail.

La sauvegarde est aisée. Depuis Netscape Navigator, il suffit de cliquer sur le bouton "*Sécurité*", puis sur le lien "*Vos certificats*". On sélectionne le certificat à sauvegarder et on clique sur le bouton "*Exporter*".

Netscape demande un mot de passe pour protéger le certificat. Ce mot de passe est choisi par l'utilisateur et connu de lui seul. Netscape demande ensuite confirmation de ce mot de passe. Une fenêtre apparaît. Elle permet à l'utilisateur d'indiquer dans quel fichier sauver le certificat au format PKCS12.

8.7 S/MIME et les listes de diffusion

L'utilisation de la messagerie S/MIME doit pourvoir se faire en particulier dans des listes de diffusion.

8.7.1 La signature

C'est l'application la plus simple de S/MIME aux listes de diffusion. Rappelons qu'un message signé via S/MIME est simplement un message SMTP contenant en attachement une partie de corps spécifique (`Content-type : x-pkcs7-signature`). Ce message peut donc être relayé sans altération par tout MTA conforme à SMTP. L'exploitation de cette signature ne dépend que des capacités du client de messagerie destinataire. Diffuser un message signé est donc possible avec n'importe quel logiciel de listes de diffusion sans modification de celui-ci.

Dans le cas des messages de service (abonnement, accès aux opérations privilégiées du responsable de liste), il peut s'avérer très précieux d'exploiter la signature S/MIME du demandeur. En effet, ces opérations demandent une identification, il serait dommage de ne pas profiter de l'authentification forte S/MIME. Pour en bénéficier, il doit être possible de configurer le robot de listes de diffusion liste par liste pour imposer ou non l'authentification S/MIME pour chaque opération. Le robot doit alors décoder les signatures comme n'importe quel client de messagerie S/MIME.

8.7.2 La confidentialité

Dans ce cas le but est de diffuser à chaque abonné un message chiffré par l'émetteur. Notre hypothèse est celle d'une liste de diffusion centralisée et gérée par un robot. L'émetteur chiffre donc son message à destination de l'adresse de la liste, pour le faire, celui-ci doit accéder à **la clef publique de la liste**[1]. Le robot doit alors décoder le message (étant le seul à pouvoir accéder à la clef privée associée, il est le seul à pouvoir le faire). La diffusion s'effectue alors en chiffrant le message à l'intention de chaque destinataire. Le robot de listes de diffusion doit donc détenir la clef publique de chaque abonné (ou être capable d'accéder à celle-ci via un annuaire LDAP) .

[1] Sera-t-il possible d'obtenir auprès d'une autorité un certificat pour un service et non pas pour un individu ?

Cette opération n'a de sens que si le contrôle de la liste des abonnés est strict, c'est l'objectif même de l'opération de chiffrement et il convient alors que le contrôle des opérations de gestion de la liste des abonnés soit aussi solide que le chiffrement S/MIME. Il serait stupide de chiffrer au sein d'une liste de diffusion à laquelle un petit malin se serait abonné par simple usurpation de l'adresse SMTP du propriétaire de liste.

A noter que si l'abonnement se fait exclusivement au moyen d'une commande *subscribe* dans un message signé, le robot peut stocker à ce moment la clef publique de chaque nouvel abonné en prévision du chiffrement. Réciproquement, en signant le message de bienvenue dans la liste, il remet à chaque nouvel abonné la clef publique de celle-ci.

9 IP Security (IPSec)

IP Security (IPSec) est un protocole fournissant un mécanisme de sécurisation au niveau IP. Ce protocole, composante indissociable d'IPV6 et utilisable aussi avec IPV4, permet de chiffrer et/ou d'authentifier les échanges entre deux équipements réseau (équipements actifs du réseau ou postes de travail).
Son origine remonte à 1995, date des premiers RFC sur le sujet, une seconde version des RFC étant disponible depuis fin 1998.
Les RFC importants sont essentiellement :

- RFC 2409 concernant l'Internet Key Exchange (IKE),
- RFC 2401 concernant IPSec,
- RFC 2402 concernant l'Authentication Header (AH),
- RFC 2406 concernant l'Encapsulating Security Payload (ESP).

Au niveau des fonctionnalités, IPSec permet donc de faire du chiffrement et de l'authentification tout en offrant la possibilité de sélectionner les algorithmes utilisés ainsi que les modalités de leur mise en œuvre. Ces modalités sont alors définies dans une « Association de Sécurité » (Security Association (SA)).

IPSec est totalement indépendant des couches protocolaires de niveau supérieur (TCP, UDP, etc.) et peut être décomposé en deux sous ensembles bien distincts :

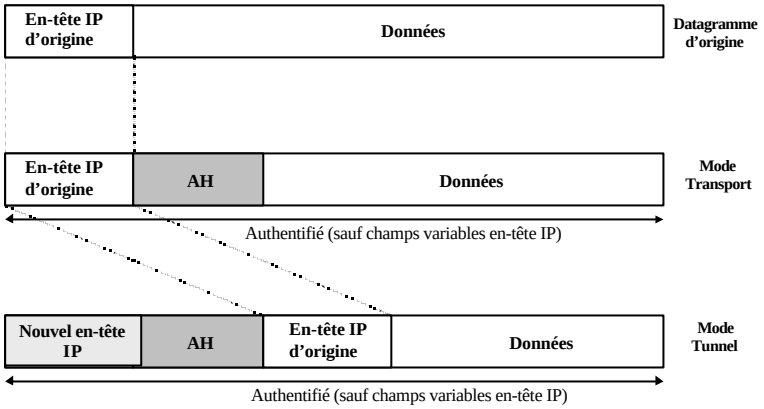
- le chiffrement et l'authentification déjà évoqués,
- la gestion des clés.

Dans ce qui suit, la description d'IPSec s'appuie uniquement sur le protocole IPV4. Pour une description plus complète sur le sujet, le lecteur intéressé pourra se reporter à la bibliographie.

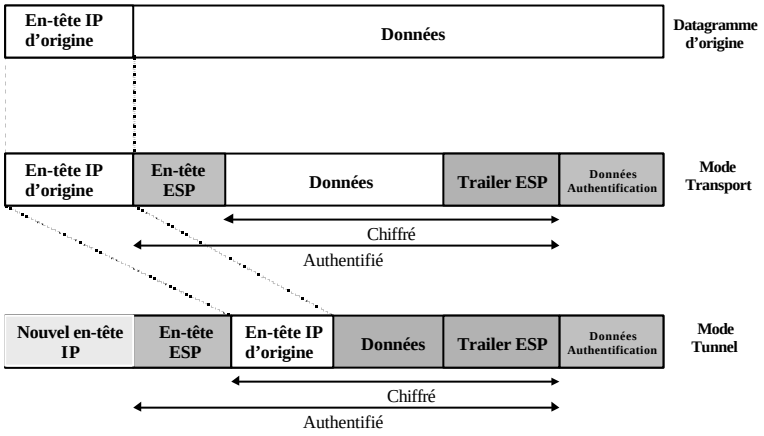
9.1 Fonctionnement d'IPSec

9.1.1 Modes de fonctionnement

Concernant IPSec, deux modes de fonctionnement sont envisageables : le mode tunnel et le mode transport. Ces deux modes sont applicables aussi bien pour le cadre de l'authentification (Authentication Header (AH)) que dans le cadre du chiffrement (Encapsulation Security Payload (ESP)).
Voici les modes de fonctionnement en mode « transport » et « tunnel » concernant AH :



Voici les modes de fonctionnement concernant ESP :



Les champs variables de l'en-tête IP (TTL, etc.) ne sont pas pris en compte pour réaliser l'authentification et sont comptabilisés comme valant zéro dans le calcul.

La différence majeure entre les deux modes réside dans le fait que dans un cas, le mode « transport », l'en-tête IP du datagramme d'origine est conservée alors que dans l'autre cas, elle est modifiée.

En conséquence, le mode « tunnel » a l'avantage concernant ESP de chiffrer totalement l'identité des entités impliquées dans l'échange et donc de masquer au maximum les caractéristiques des flux d'information.

9.1.2 Les Associations de sécurité (SAs)

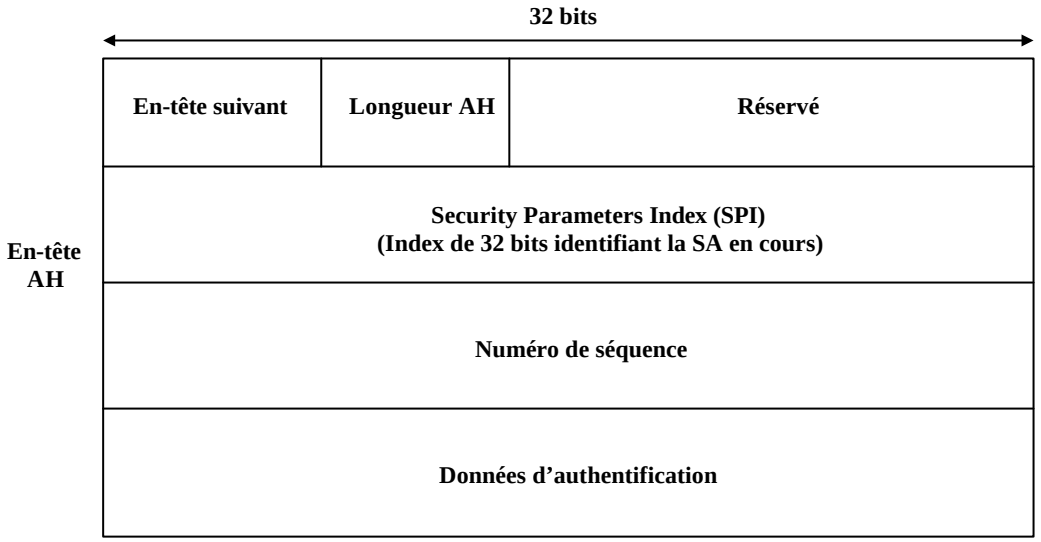
La définition qui suit est reprise de la définition de [LABOURET1999] et fournit une vision synthétique de la notion de SA :

« Une association de sécurité (SA) se définit comme une connexion simplexe qui offre des services de sécurité au trafic qu'elle transporte. Les services de sécurité fournis par une SA mettent en œuvre l'un ou l'autre des protocoles AH ou ESP, mais pas les deux. Si les deux protocoles doivent être appliqués à un flux de données, il est nécessaire de créer deux SAs distinctes. Une communication bidirectionnelle typique entre deux entités nécessite l'ouverture de deux SAs, une dans chaque sens de communication. »

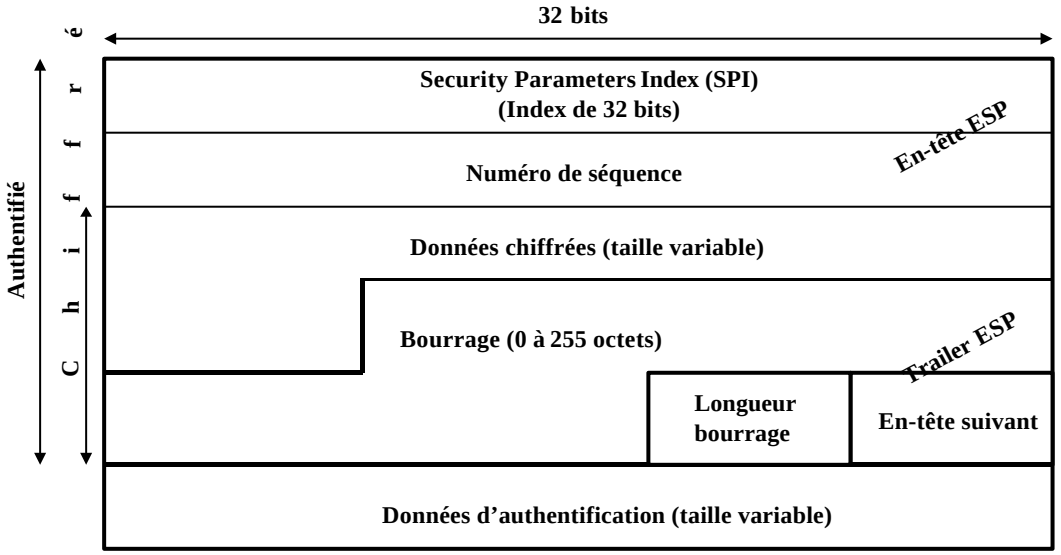
Concrètement, une SA est identifiée de manière unique par un triplet constitué d'un indice de paramètre de sécurité (Security Parameter Index ou SPI, bloc de 32 bits), d'une adresse IP de destination, et d'un identifiant de protocole de sécurité (AH ou ESP).

Pour être plus explicite, voyons comment ces notions se traduisent au niveau AH et ESP.

L'implémentation de AH est la suivante :



Pour ce qui concerne ESP :



Datagramme avec ESP

Les champs « En-tête suivant » font référence au type du contenu après le paquet AH ou ESP et sont conformes au RFC 1340 décrivant et référençant les protocoles.
Le champ SPI est une valeur arbitraire de 32 bits, utilisée avec l'adresse IP de destination et le type de protocole utilisé (AH ou ESP) pour caractériser une SA.
Le champ « Données d'authentification » contient la valeur de contrôle du paquet, c'est-à-dire une signature garantissant l'intégrité, à la réception, du paquet qui a été émis.
Quant au champ «Numéro de séquence », il est utilisé comme protection contre le rejeu des connexions. C'est une option négociée lors de la création de la SA.

9.1.3 IPSec et la gestion des clés

Au delà des aspects purement implémentation d'IPSec qui viennent d'être décrits à travers AH et ESP, le principal problème à résoudre réside dans la gestion des clés et des SAs.

Pour cela, le principe retenu par l'IETF a été de gérer ces SAs (et donc les algorithmes de chiffrement, les clés, etc.) comme une deuxième composante bien distincte au sein du protocole.

De base, il existe bien entendu la possibilité de gérer manuellement les clés utilisées dans IPSec. Mais, pour des raisons évidentes de performance et de mise en œuvre, un protocole de négociation des SAs a été développé.

Ce protocole de négociation des SAs s'appelle *Internet Security Association and Key Management Protocol* (ISAKMP).

ISAKMP fournit un cadre générique à la négociation des SAs. D'autres protocoles tels que SKEME et Oakley (protocoles de gestion des clés antérieurs à ISAKMP, dont la description dépasse le cadre de cet article) ont été repris certaines de leurs fonctionnalités et combinés à ISAKMP pour donner en final le protocole IKE (Internet Key Exchange).

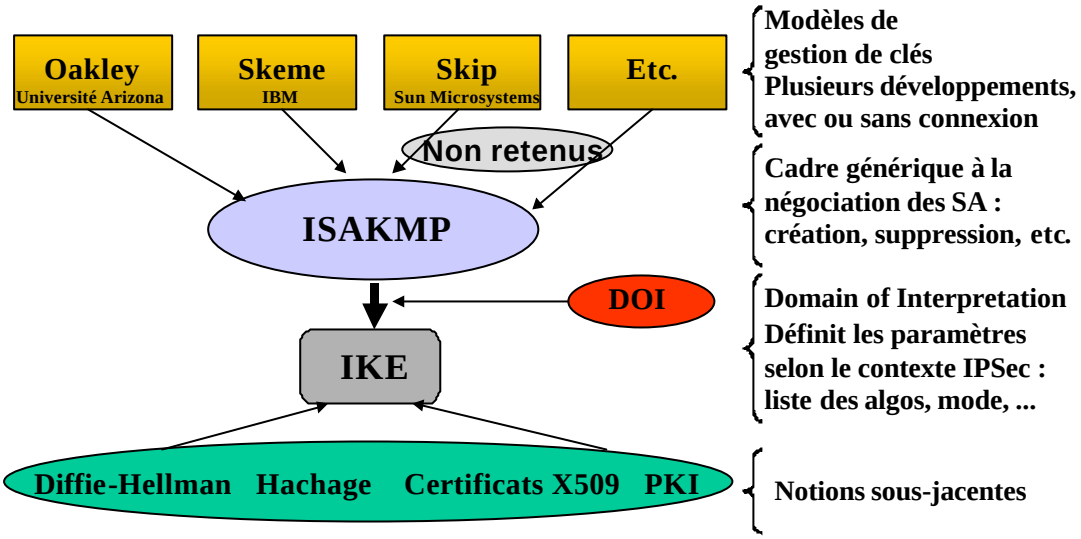
IKE est en fait un protocole hybride fédérant SKEME et Oakley au sein d'ISAKMP. Pour le décrire sommairement, on peut dire que IKE est utilisé pour construire un échange de clés sécurisé et une authentification des entités voulant entrer en communication, et ce, en amont de tout échange de données.

Cette fonction peut être réalisée manuellement, via une autorité de certification ou DNSSec par exemple. Dans IPSec, c'est IKE (ou autrement dit ISAKMP/Oakley) qui assure cette fonction.

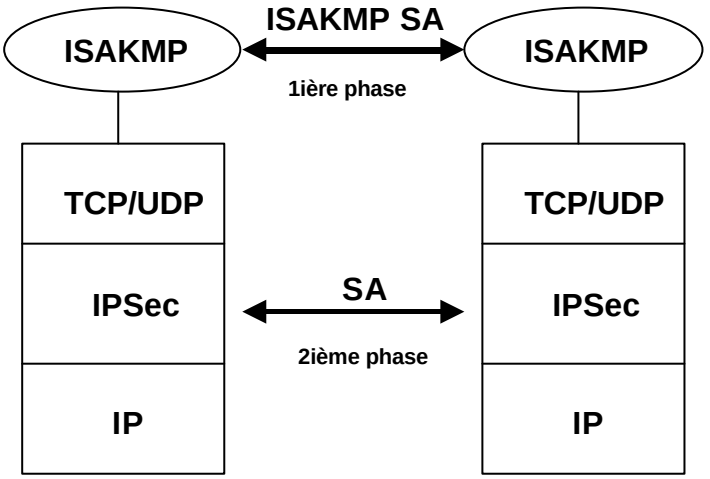
De plus, ISAKMP ne fournissant qu'un cadre générique, s'ajoute à tout cela la notion de *Domain of Interpretation* (DOI). Concrètement, c'est le DOI qui définit les paramètres négociés dans les échanges ISAKMP.

Sans entrer plus avant dans la description sur ces aspects de gestion des clés, il est important de retenir que les notions mises en jeu sont toujours basées sur des concepts connus, à savoir des algorithmes asymétriques (Diffie-Hellman), des fonctions de hachage de type MD5, des certificats X509, etc. ainsi que des infrastructures à clés publiques (PKI).

Voici un schéma synoptique résumant quelque peu ces notions pour lesquelles la littérature sur le sujet est, en général, rarement très explicite :



Concernant la notion de SA, sa mise en œuvre au niveau IPSec « couches basses » se retrouve aussi au niveau ISAKMP/Oakley lui-même comme l'indique le schéma qui suit :



Ainsi, pour construire une SA de niveau IPSec, il est nécessaire au préalable d'établir une SA de niveau ISAKMP.

C'est à ce niveau que sont utilisés des mécanismes de type Diffie-Hellman et certificats X509 pour amorcer et établir un échange sécurisé. On peut noter qu'à ce niveau, contrairement au niveau IPSec, la SA est bi-directionnelle (RFC 2409).

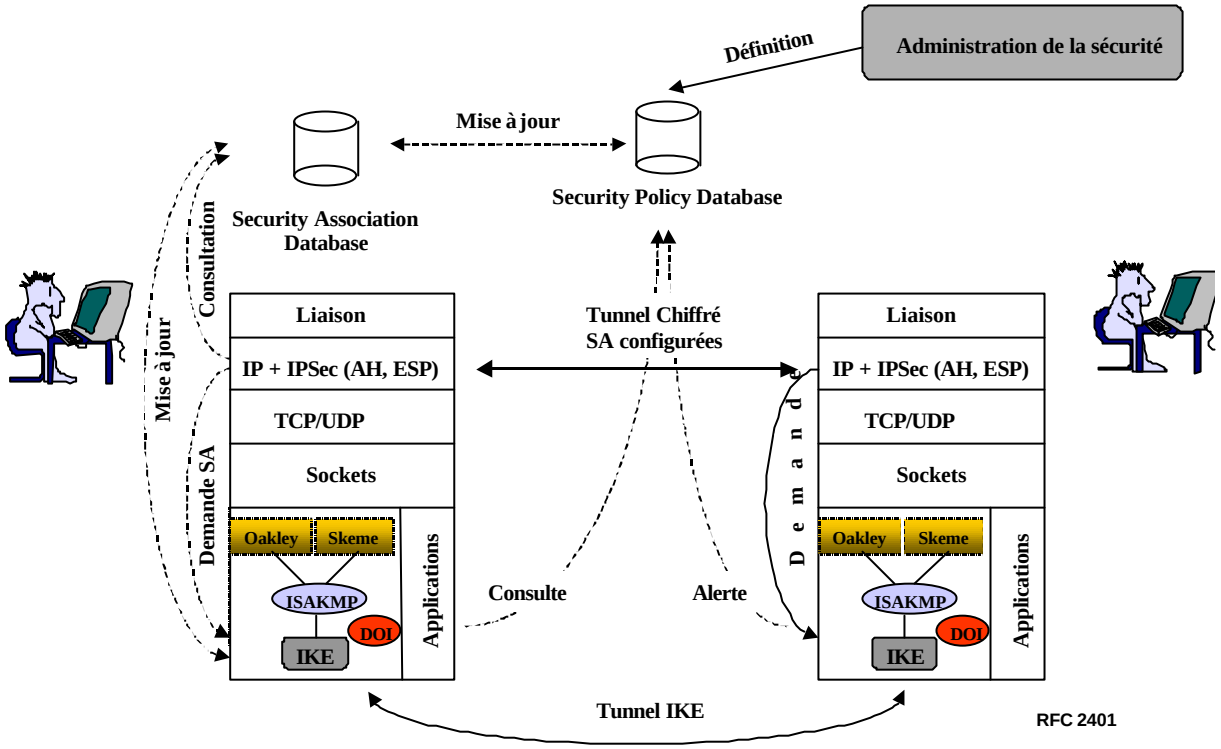
Le protocole ISAKMP préconise l'emploi d'un format de données bien spécifique avec, lors de l'échange, l'utilisation de « cookies » pour identifier les participants à la connexion. L'IANA a même normalisé le port et le protocole requis pour ISAKMP : protocole UDP et port 500.

Comme le précise la définition d'une SA, l'établissement d'une communication bidirectionnelle au niveau IPSec nécessite la création de deux SAs, une dans chaque sens.
Il peut donc être nécessaire aux couches réseau de solliciter par deux fois un protocole de niveau applicatif.

Les conséquences en sont donc à la fois une perte de performance lors de l'établissement d'une connexion par exemple mais aussi un gain fonctionnel important pour garantir la sécurisation de la communication.

Le schéma qui suit illustre de manière très simplifiée les composantes nécessaires à l'établissement d'une connexion sécurisée via IPSec.

On distingue bien deux éléments dissociés qui sont IPSec proprement dit (ESP et AH) et la gestion des clés (IKE).



Concernant le fonctionnement du protocole, la couche IPSec fait donc normalement appel à IKE pour construire une SA. Dans ce cas, si la SA n'existe pas au préalable, elle est soit construite directement, soit extraite d'une Security Association Database (SAD).

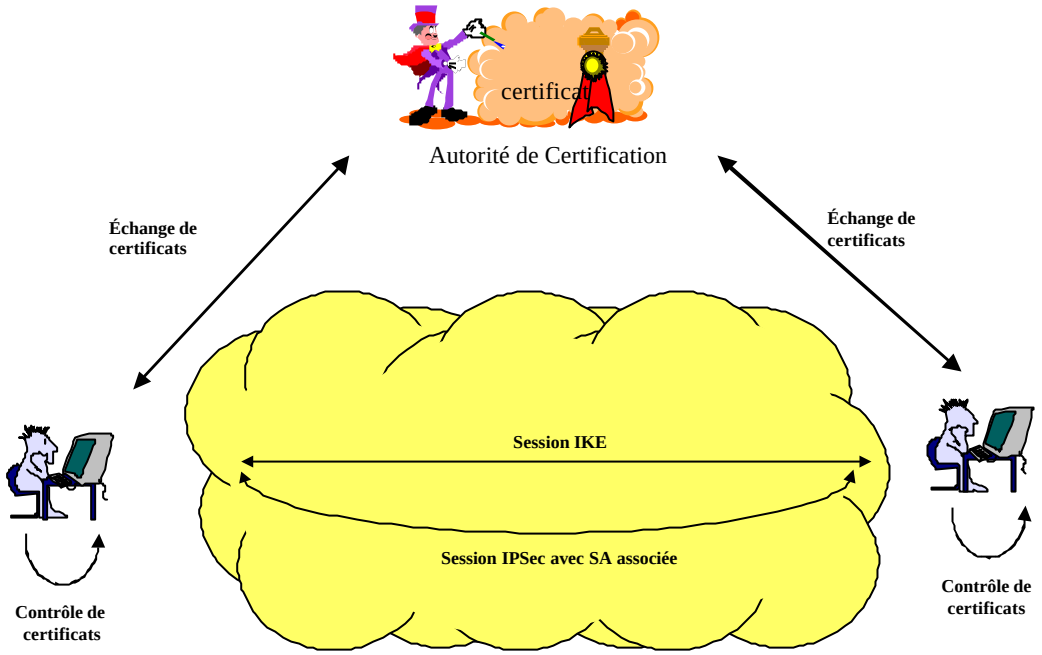
En effet, le RFC 2401 définit la notion de bases de données de sécurité. Ces bases sont au nombre de deux : la SAD, qui contient tous les paramètres associés à chaque SA active et la Security Policy Database (SPD) qui décrit la politique de sécurité à travers la liste des flux entrants et sortants autorisés.

La politique sécurité appliquée aux datagrammes peut être alors de trois types :

- rejet du datagramme,
- passage sans protection (équivalent à un filtre passant),
- l'application d'IPSec (AH et ESP).

L'application de cette politique se base sur les critères classiques que sont l'adresse IP de destination, l'adresse IP source, le nom DNS ou un DN (au sens X500), le protocole de transport utilisé (TCP ou UDP) et les ports source et destination.

La SAD contient quant à elle les SAs actives et les caractéristiques associées telles que la durée de vie, les paramètres de ESP et AH utilisés, le mode tunnel ou transport, etc.
Le schéma qui suit, inspiré de [CISCO1998], illustre l'interaction entre IPSec, IKE et une infrastructure à clés publiques (PKI).



Dans ce cas, le démarrage de la session IKE est assuré par un échange de certificats X509 entre les entités voulant communiquer.
Ces certificats sont garantis par une autorité de certification acceptée par l'ensemble des parties voulant entrer en communication.

Globalement, concernant le mécanisme d'échange des clés, IPSec prévoit une souplesse maximale au niveau du choix des mécanismes à employer via ISAKMP.

Comme il l'a déjà été suggéré, le mode manuel, les PKI, etc. sont envisageables, mais aussi d'autres protocoles tels que DNSSEC par exemple. Pour mémoire, DNSSEC permet d'étendre les Ressources Record (RR) du DNS classique avec de nouveau champs (certificats, etc.) et d'utiliser les fonctionnalités du DNS ainsi modifié pour échanger des certificats.
Dans ce dernier cas, la difficulté consiste à résoudre les problèmes liés à l'emploi de fonctions telles que Network Address Translation (NAT), DHCP, etc.

9.1.4 Les modes de fonctionnement de IKE

Le protocole IKE fonctionne en deux phases : la première permet d'établir la SA pour ISAKMP et la deuxième d'établir la ou les SAs pour IPSec.
Il est possible de noter ici que la SA construite dans la première phase peut servir à construire plusieurs SAs dans la deuxième phase.

La première phase se décline en deux modes qui sont le « Main Mode » (6 messages échangés) et l'« Agressive Mode » (3 messages échangés).

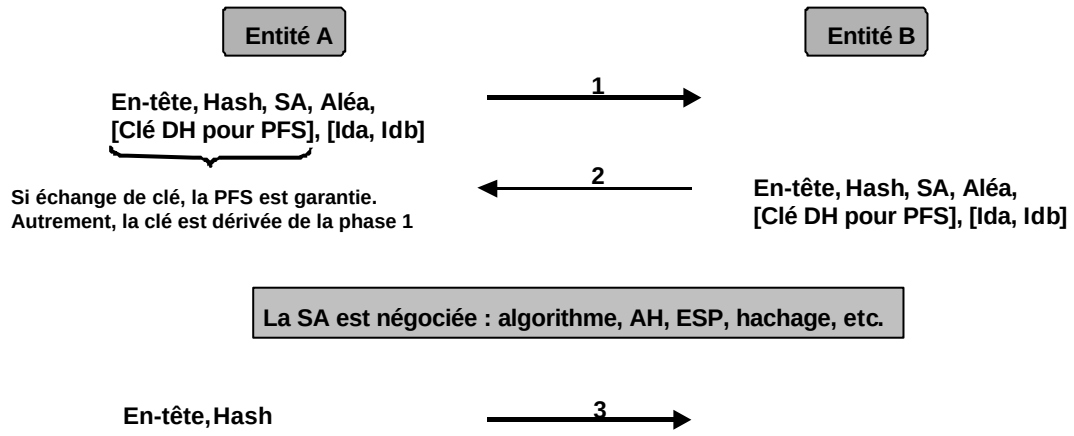
La deuxième phase comprend deux options, le «Quick Mode » avec PFS (Perfect Forward Secrecy) et le « Quick Mode » sans PFS.

La propriété PFS caractérise le fait que la découverte d'une clé ne remet pas en cause l'utilisation des autres clés (par exemple, la découverte d'une clé maître ne compromet pas les clés de session, ce qui garantit dans le temps la validité des échanges ayant déjà eu lieu).
Un autre mode existe, le « New Group Mode », mais ce mode est un peu à part dans le sens où il n'est utilisé dans aucune des phases 1 ou 2.

Durant la première phase, en « Main Mode » ou « Agressive Mode », les informations suivantes sont échangées :

- un algorithme de chiffrement et une fonction de hachage,
- une méthode d'authentification,
- les valeurs publiques pour Diffie-Hellman,
- trois clés : chiffrement, authentification et une clé pour la dérivation d'autres clés.

Durant la deuxième phase, les messages sont protégés grâce aux paramètres échangés lors de la première phase. En plus, des aléas sont utilisés pour éviter le rejeu des sessions.
Voici une illustration de ce mode d'échange :



Dans l'exemple ci-dessus, la propriété PFS n'est garantie que lorsque des clés Diffie-Hellman sont de nouveau échangées. L'entête ISAKMP quant à elle fixe le « formatage » des échanges : place du Hash, de l'aléa, etc. dans le paquet. Toutes les données sont échangées chiffrées dès le premier échange grâce à la SA du niveau ISAKMP.
Une description plus complète de ce mode et des autres modes disponibles est fournie dans le RFC 2409.

9.2 Implémentations actuelles et exemple de mise en œuvre avec des routeurs

Il existe aujourd'hui bon nombre d'implémentations d'IPSec. Ces implémentations sont soit de nature commerciale, soit du domaine public, sachant que les deux environnements sont relativement dynamiques sur le sujet.

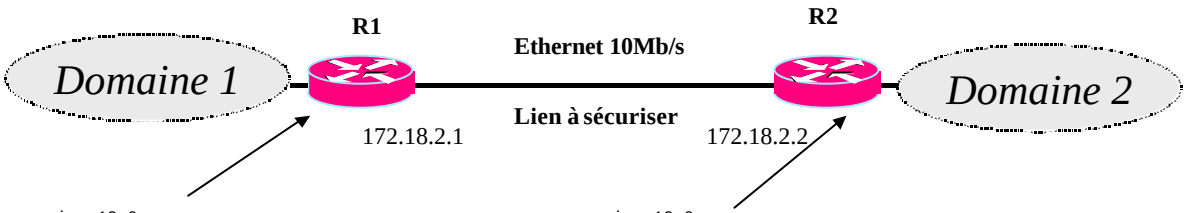
Au niveau des réalisations, il est possible de citer :

- FreeS/Wan pour Linux,
- L'implémentation native dans FreeBSD,
- L'implémentation dans AIX pour les versions supérieures à la 4.3,
- Des implémentations dans les firewall (Firewall-1 de Checkpoint par exemple),
- Et des implémentations dans des équipements tels que des équipements dédiés au chiffrement ou tout simplement des routeurs (Cisco à partir de la version IOS 11.3 par exemple).

Dans tout ces exemples, les modalités de mise en œuvre sont plus ou moins complexes et vont de l'implémentation native (FreeBSD, IOS Cisco, etc.) à la nécessaire recompilation du noyau (FreeS/Wan, etc.).

La maquette qui suit va permettre d'illustrer de manière plus opérationnelle une petite mise en œuvre en employant les fonctionnalités IPSec de l'IOS de Cisco.
Cet exemple n'est qu'une illustration technique, un déploiement réel nécessitant bien sûr une étude préalable quant aux flux à sécuriser et à leur intégration au sein d'une politique sécurité.

Voici le schéma de principe ainsi que la configuration de base des routeurs, sans IPSec, assurant la simple interconnection des équipements :




```
version 12.0
service config
hostname r1
!
enable secret 5 $1$C/Zh$uIE9MzyayxHMcTjB0qPnu0
!
!
interface Ethernet0/0
 ip address 172.18.2.1 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable
!
!
interface Ethernet0/1
 ip address 172.19.2.1 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable

line con 0
 exec-timeout 0 0
 transport input none
line aux 0
line vty 0 4
 password password
 login
!
end
```

```
version 12.0
serviceonfig
hostname r2
!
enable secret 5 $1$C/Zh$uIE9MzyayxHMcTjB0qPnu0
!
!
interface Ethernet0/0
 ipaddress 172.18.2.2 255.255.255.252
 noipdirected-broadcast
 no ip mroute-cache
 no cdp enable
!
!
interface Ethernet0/1
 ip address 172.20.2.1 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable

line con 0
 exec-timeout 0 0
 transport input none
line aux 0
line vty 0 4
 password password
 login
!
end
```

Au niveau des configurations physiques, les caractéristiques des routeurs R1 et R2 sont les suivantes :

- routeur Cisco 3640,
- 8 Mo de mémoire flash, 32 Mo de RAM,
- IOS version 12.0.3(T) avec IP, IP Plus et IPSec 56 bits.

Dans ce qui suit, la mise en œuvre d'IPSec n'utilise pas, pour des raisons de simplicité, d'autorité de certification (CA) et de certificats X509 pour établir une SA de niveau ISAKMP. En lieu et place seront utilisées des « pre-shared keys » déjà configurées et existantes au niveau du routeur.

Ces clés sont issues d'une fonctionnalité de Oakley et fixent les paramètres de base d'un échange Diffie-Hellman (groupe 1 pour les clés de 768 bits et groupe 2 pour les clés de 1024 bits. Par défaut le groupe 1 est employé).

Les configurations qui suivent font référence au routeur R1, la configuration du routeur R2 étant symétrique.

Plusieurs étapes sont nécessaires pour mettre en œuvre IPSec dans le cadre de l'exemple et avec les restrictions précédentes :

1- Configuration des paramètres ISAKMP :
Sur le routeur R1, cette configuration est la suivante :

```
crypto isakmp policy 1
 hash md5
 authentication pre-share
 lifetime 3600
```

Dans l'exemple, elle permet de fixer l'algorithme de hachage utilisé, la durée de vie de la SA négociée et le fait d'utiliser des « pre-shared keys ».

2- Configuration de la clé ISAKMP

```
crypto isakmp key motdepasse address 172.18.2.2
```

Le mot de passe « motdepasse » est associé à l'adresse 172.18.2.2, le routeur R2.

Dans le cas de l'utilisation d'une autorité de certification et non de la fonction «pre-shared keys », il aurait fallu réaliser les étapes suivantes, en lieu et place des étapes 1 et 2 :

- créer un couple RSA sur le routeur (**`crypto key gen rsa`** …)
- récupérer le certificat d'une autorité de certification (**`crypto ca identity <MaCA>`** **puis** **`enrollment url http://<MaCA>`**)
- faire certifier la clé publique du routeur par l'autorité de certification (**`crypto ca enroll <MaCA>`**)
- enfin, configurer les paramètres ISAKMP comme précédemment mais sans le paramètre « **`authentication pre-shared`** ».

3- Mise en œuvre d'une « access-list »

L'étape suivante consiste à créer une « access-list » étendue permettant de déclencher IPSec selon la nature des flux d'information :

```
access-list 102 permit ip host 172.18.2.1 host 172.18.2.2
```

On peut noter que cette «access-list » prend comme première adresse IP la source, c'est-à-dire le routeur R1 lui-même en l'occurrence. Elle doit se concevoir en prenant en compte le flux sortant.

4- Configuration des paramètres des SAs

Il reste ensuite à configurer les paramètres négociés lors de l'établissement des SAs :

```
crypto ipsec transform-set trans1 esp-rfc1829
 mode transport
crypto ipsec transform-set trans2 ah-md5-hmac esp-des
 mode transport
```

Les paramètres à négocier peuvent être différents selon les cas et les configurations voulues. Ici, ces paramètres relèvent purement du choix de l'auteur et sont souvent montrés à titre d'exemple dans des documents de configuration pour illustrer le fait que même les anciennes spécifications (RFC 1829 en l'occurrence) sont prises en compte et restent compatibles avec les implémentation récentes.

C'est aussi à ce niveau qu'est précisé le mode de fonctionnement d'IPSec : mode tunnel ou mode transport.

5- Création de la « crypto map »

Puis, il faut ensuite créer une « crypto-map » fournissant les caractéristiques quant au lien IPSec lui-même :

```
crypto map e1 1 ipsec-isakmp
 set peer 172.18.2.2
 set transform-set trans1 trans2
 match address 102
```

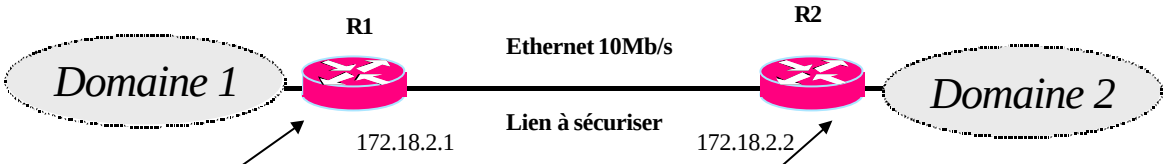
A ce niveau sont précisés le routeur symétrique au routeur R1, l'ordre de préférence dans la négociation des paramètres de SAs (trans1 et trans2) et «l'access-list » associée permettant de déclencher IPSec.

6- Activation d'IPSec

La dernière étape consiste à appliquer la « crypto-map » précédemment définie sur une interface pour rendre opérationnel les configurations (ordre **`crypto map e1`**) :

```
interface Ethernet0/0
 ip address 172.18.2.1 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable
 crypto map e1
```

Les configurations des routeurs avec IPSec sont alors les suivantes :




```
!
version 12.0
hostname r1
!
enable secret 5 $1$C/Zh$uIE9MzyayxHMcTjB0qPnu0
!
crypto isakmp policy 1
 hash md5
 authentication pre-share
 lifetime 3600
crypto isakmp key motdepasse address 172.18.2.2
!
crypto ipsec transform-set trans1 esp-rfc1829
 mode transport
crypto ipsec transform-set trans2 ah-md5-hmac esp-des
 mode transport
!
crypto map e1 1 ipsec-isakmp
 set peer 172.18.2.2
 set transform-set trans1 trans2
 match address 102
!
interface Ethernet0/0
 ip address 172.18.2.1 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable
 crypto map e1
!
no ip http server
!
access-list 102 permit ip host 172.18.2.1 host 172.18.2.2
!
end
```

```
!
version 12.0
hostname r2
!
enable secret 5 $1CaxkIhtzyayxHMcTjB0qlppt
!
crypto isakmp policy 1
 hash md5
 authentication pre-share
 lifetime 3600
crypto isakmp key motdepasse address 172.18.2.1
!
crypto ipsec transform-set trans1 esp-rfc1829
 mode transport
crypto ipsec transform-set trans2 ah-md5-hmac esp-des
 mode transport
!
crypto map e1 1 ipsec-isakmp
 set peer 172.18.2.1
 set transform-set trans1 trans2
 match address 102
!
interface Ethernet0/0
 ip address 172.18.2.2 255.255.255.252
 no ip directed-broadcast
 no ip mroute-cache
 no cdp enable
 crypto map e1
!
no ip http server
!
access-list 102 permit ip host 172.18.2.2 host 172.18.2.1
!
end
```

A ce niveau, les configurations de base étant effectives, il est possible d'en visualiser le résultat (à partir du routeur R1) :

- **R1#sh crypto map**

```
Crypto Map "e1" 1 ipsec-isakmp
        Peer = 172.18.2.2
        Extended IP access list 102
            access-list 102 permit ip host 172.18.2.1 host 172.18.2.2
        Current peer: 172.18.2.2
        Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ trans1, trans2, }
```

- **R1#sh crypto ip transform-set**

```
Transform set trans1: { esp-rfc1829  }
   will negotiate = { Var len IV, Transport,  },
   IV length: 8 bytes

Transform set trans2: { ah-md5-hmac  }
   will negotiate = { Transport,  },
   { esp-des  }
   will negotiate = { Transport,  },
```

```
-  R1#sh crypto ip sa

interface: Ethernet0/0
    Crypto map tag: e1, local addr. 172.18.2.1

   local  ident (addr/mask/prot/port): (172.18.2.1/255.255.255.255/0/0)
   remote ident (addr/mask/prot/port): (172.18.2.2/255.255.255.255/0/0)
   current_peer: 172.18.2.2
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 0, #pkts encrypt: 0, #pkts digest 0
    #pkts decaps: 0, #pkts decrypt: 0, #pkts verify 0
    #send errors 0, #recv errors 0
     local crypto endpt.: 172.18.2.1, remote crypto endpt.: 172.18.2.2
     path mtu 1500, media mtu 1500
     current outbound spi: 0
     inbound esp sas:
     inbound ah sas:
     outbound esp sas:
     outbound ah sas:
```

Le contrôle des configurations permet de corriger les erreurs éventuelles et de prendre en compte, si besoin, de nouveaux paramètres (PFS, lifetime, etc.).
L'étape suivante consiste à activer le mode « debug » pour permettre de visualiser le résultat de la mise en œuvre d'IPSec :

- **R1#debug crypto isakmp**
- **R1#debug crypto ipsec**
- **R1#debug cryto engine**

Ensuite un ping du routeur R1 vers le routeur R2 permet de visualiser les phases ISAKMP et IPSec. Au préalable, la commande « **sh cryto engine connection active** » permet de confirmer l'établissement d'une connexion sécurisée :

```
R1#ping 172.18.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.18.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms
R1#sh cryp en conn a

  ID Interface       IP-Address      State  Algorithm           Encrypt  Decrypt
   1 no idb          no address      set    DES_56_CBC                0        0
   2 Ethernet0/0     172.18.2.1      set    DES_56_CBC                0        5
   3 Ethernet0/0     172.18.2.1      set    DES_56_CBC                5        0
```

A ce niveau, on constate d'une part que les routeurs R1 et R2 sont atteignables entre eux et, d'autre part, que 5 paquets de part et d'autre (paquets issus du ping) ont bien été échangés avec chiffrement et déchifffement dans les deux sens vu de R1).

Conformément à la définition des SAs, deux SAs sont bien crées pour l'échange (une dans chaque sens) :

```
R1#sh cryp ip sa

interface: Ethernet0/0
    Crypto map tag: e1, local addr. 172.18.2.1

   local  ident (addr/mask/prot/port): (172.18.2.1/255.255.255.255/0/0)
   remote ident (addr/mask/prot/port): (172.18.2.2/255.255.255.255/0/0)
   current_peer: 172.18.2.2
```

```
     PERMIT, flags={origin_is_acl,transport_parent,}
    #pkts encaps: 2, #pkts encrypt: 2, #pkts digest 0
    #pkts decaps: 2, #pkts decrypt: 2, #pkts verify 0
    #send errors 8, #recv errors 0

     local crypto endpt.: 172.18.2.1, remote crypto endpt.: 172.18.2.2
     path mtu 1500, media mtu 1500
     current outbound spi: E772311
     inbound esp sas:
      spi: 0x52924B8(86582456)
        transform: esp-rfc1829 ,
        in use settings ={Var len IV, Transport, }
        slot: 0, conn id: 2, crypto map: e1
        sa timing: remaining key lifetime (k/sec): (4607999/3377)
        IV size: 8 bytes
        replay detection support: N
     inbound ah sas:
     outbound esp sas:
      spi: 0xE772311(242688785)
        transform: esp-rfc1829 ,
        in use settings ={Var len IV, Transport, }
        slot: 0, conn id: 3, crypto map: e1
        sa timing: remaining key lifetime (k/sec): (4607999/3359)
        IV size: 8 bytes
        replay detection support: N
     outbound ah sas:
```

Quant aux traces issues du mode « debug » (les éléments importants figurent dans une police de caractères plus grande) :

```
/* REQUETE IPSEC POUR CREATION DE SAs */
00:03:42: IPSEC(sa_request): ,
  (key eng. msg.) src= 172.18.2.1, dest= 172.18.2.2,
    src_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    dest_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-rfc1829 ,
    lifedur= 3600s and 4608000kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4000
00:03:42: IPSEC(sa_request): ,
  (key eng. msg.) src= 172.18.2.1, dest= 172.18.2.2,
    src_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    dest_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    protocol= AH, transform= ah-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4000
00:03:42: IPSEC(sa_request): ,
  (key eng. msg.) src= 172.18.2.1, dest= 172.18.2.2,
    src_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    dest_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-des ,
    lifedur= 3600s and 4608000kb,

/* DEBUT DE NEGOCIATION ISAKMP */
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4000
00:03:42: ISAKMP (1): beginning Main Mode exchange
00:03:57: ISAKMP (1): retransmitting phase 1...
00:03:57: ISAKMP (1): processing SA payload. message ID = 0
00:03:57: ISAKMP (1): Checking ISAKMP transform 1 against priority 1 policy
00:03:57: ISAKMP:      encryption DES-CBC
00:03:57: ISAKMP:      hash MD5
00:03:57: ISAKMP:      default group 1
00:03:57: ISAKMP:      auth pre-share
00:03:57: ISAKMP:      life type in seconds
00:03:57: ISAKMP:      life duration (basic) of 3600
00:03:57: ISAKMP (1): atts are acceptable. Next payload is 0
00:03:57: Crypto engine 0: generate alg param

00:03:57: CRYPTO_ENGINE: Dh phase 1 status: 0
00:03:57: ISAKMP (1): SA is doing pre-shared key authentication
00:03:57: ISAKMP (1): processing KE payload. message ID = 0
00:03:57: Crypto engine 0: generate alg param

00:03:57: ISAKMP (1): processing NONCE payload. message ID = 0
00:03:57: Crypto engine 0: create ISAKMP SKEYID for conn id 1
00:03:57: ISAKMP (1): SKEYID state generated
00:03:57: ISAKMP (1): processing vendor id payload
```

```
00:03:57: ISAKMP (1): speaking to another IOS box! /* :-) */
00:03:57: generate hmac context for conn id 1
00:03:57: ISAKMP (1): processing ID payload. message ID = 0
00:03:57: ISAKMP (1): processing HASH payload. message ID = 0
00:03:57: generate hmac context for conn id 1
00:03:57: ISAKMP (1): SA has been authenticated with 172.18.2.2
00:03:57: ISAKMP (1): beginning Quick Mode exchange, M-ID of 355270384
```

/* LA SA de NIVEAU IKE EST CREE. LE NIVEAU IPSEC COMMENCE POUR SES SAs */

```
00:03:57: IPSEC(key_engine): got a queue event...
00:03:57: IPSEC(spi_response): getting spi 86582456ld for SA
        from 172.18.2.2      to 172.18.2.1      for prot 3
00:03:57: IPSEC(spi_response): getting spi 516494449ld for SA
        from 172.18.2.2      to 172.18.2.1      for prot 2
00:03:57: IPSEC(spi_response): getting spi 26219091ld for SA
        from 172.18.2.2      to 172.18.2.1      for prot 3
00:03:58: generate hmac context for conn id 1
00:03:58: generate hmac context for conn id 1
00:03:58: ISAKMP (1): processing SA payload. message ID = 355270384
00:03:58: ISAKMP (1): Checking IPSec proposal 1
00:03:58: ISAKMP: transform 1, ESP_DES_IV64
00:03:58: ISAKMP:   attributes in transform:
00:03:58: ISAKMP:      encaps is 2
00:03:58: ISAKMP:      SA life type in seconds
00:03:58: ISAKMP:      SA life duration (basic) of 3600
00:03:58: ISAKMP:      SA life type in kilobytes
00:03:58: ISAKMP:      SA life duration (VPI) of  0x0 0x46 0x50 0x0
00:03:58: ISAKMP (1): atts are acceptable.
```

/* EXAMEN DES PARAMETRE ECHANGES */

```
00:03:58: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) dest= 172.18.2.2, src= 172.18.2.1,
    dest_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    src_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-rfc1829 ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x0
00:03:58: ISAKMP (1): processing NONCE payload. message ID = 355270384
00:03:58: ISAKMP (1): processing ID payload. message ID = 355270384
00:03:58: ISAKMP (1): processing ID payload. message ID = 355270384
00:03:58: generate hmac context for conn id 1
00:03:58: ISAKMP (1): Creating IPSec SAs
00:03:58:         inbound SA from 172.18.2.2      to 172.18.2.1      (proxy 172.18.2.2
to 172.18.2.1     )
00:03:58:         has spi 86582456 and conn_id 2 and flags 0
00:03:58:         lifetime of 3600 seconds
00:03:58:         lifetime of 4608000 kilobytes
00:03:58:         outbound SA from 172.18.2.1      to 172.18.2.2      (proxy 172.18.2.1
to 172.18.2.2     )
00:03:58:         has spi 242688785 and conn_id 3 and flags 0
00:03:58:         lifetime of 3600 seconds
00:03:58:         lifetime of 4608000 kilobytes
00:03:58: IPSEC(key_engine): got a queue event...
00:03:58: IPSEC(initialize_sas): ,
  (key eng. msg.) dest= 172.18.2.1, src= 172.18.2.2,
    dest_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    src_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-rfc1829 ,
    lifedur= 3600s and 4608000kb,
    spi= 0x52924B8(86582456), conn_id= 2, keysize= 0, flags= 0x0
00:03:58: IPSEC(initialize_sas): ,
  (key eng. msg.) src= 172.18.2.1, dest= 172.18.2.2,
    src_proxy= 172.18.2.1/255.255.255.255/0/0 (type=1),
    dest_proxy= 172.18.2.2/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-rfc1829 ,
    lifedur= 3600s and 4608000kb,
    spi= 0xE772311(242688785), conn_id= 3, keysize= 0, flags= 0x0
```

/* LES SAs DE NIVEAU IPSEC VONT ETRE CREES. LES ECHANGES VONT COMMENCER */

```
00:03:58: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.18.2.1, sa_prot= 50,
    sa_spi= 0x52924B8(86582456),
    sa_trans= esp-rfc1829 , sa_conn_id= 2
00:03:58: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.18.2.2, sa_prot= 50,
    sa_spi= 0xE772311(242688785),
    sa_trans= esp-rfc1829 , sa_conn_id= 3
```

On retrouve bien dans ces traces les deux phases concernant ISAKMP et IPSec. De même tous les paramètres fixés dans les configurations des routeurs R1 et R2 se retrouvent, sans surprise, lors de la négociation.

Pour conclure sur le sujet, il est possible de faire un certain nombre de constats à l'issue de cette petite expérimentation et d'ajouter des informations complémentaires :

- la mise en œuvre d'IPSec dans le contexte d'un environnement intégré comme IOS de Cisco a été relativement simple dans le cadre la maquette,
- les performances obtenues semblent acceptables. Au niveau mise en œuvre, des Cisco 2500 peuvent suffire et dans ce cas, il faut environ 4 secondes pour réaliser l'exponentiation Diffie-Hellman avec des clés de 1024 bits (source Cisco),
- au niveau mémoire, la table des SAs dans le routeur prend environ 300 octets auxquels s'ajoutent 120 octets par entrée (540 octets si deux SAs sont utilisées pour une communication, en entrée et en sortie),
- au niveau des échanges, les performances sont évidemment moindres pour les flux chiffrés,
- de même, les performances des flux non chiffrés sont un peu à la baisse à cause du passage de chaque paquet IP à l'examen de la « crypto map ».

Vers une infrastructure de gestion de clefs publiques

```
Jacques : Hé ! Dis-moi donc, comment être sûr que personne n'espionne
          notre correspondance ?
 Pierre : Je sais ! J'utilise un algorithme de chiffrement avec une clef
          secrète, tu es le seul à pouvoir lire le courrier que je te
          poste car je te passe la clef secrète

Jacques : Oui mais, comment être sûr que personne ne va espionner le
          réseau et s'emparer de la clef secrète ?
 Pierre : Je sais ! J'utilise une nouvelle clef secrète pour chaque
          message, je te la poste avec le message mais je la crypte avec
          un algorithme asymétrique. Pour le faire, j'utilise ta clef
          publique et tu es le seul, avec ta clef privée, à pouvoir
          retrouver la clef secrète pour déchiffrer le message.

Jacques : Oui mais, comment peux-tu être sûr que la clef publique que tu
          utilises n'est pas un faux ?
 Pierre : Je sais ! Tu fais signer ta clef publique par une autorité de
          certification. En vérifiant la signature de ta clef publique,
          (avec la clef publique de l'autorité de certification), je
          sais que l'autorité de certification a bien vérifié que tu es
          le titulaire de cette clef publique. Tu me suis ?

Jacques : Oui mais, comment puis-je être sûr que la clef publique que tu
          utilises n'est pas un faux certifié par une autorité de
          certification bidon ?
 Pierre : Pour cela Jacques, il nous faut gérer une infrastructure de
          clefs publiques
```

Les spécifications de protocoles et les implémentations de solutions de chiffrement existent pour un grand nombre d'applications. Des serveurs et des clients compatibles avec le chiffrement par certificat sont même déjà largement déployés dans nos établissements (Apache, IIS, Netscape, Internet-Explorer, …).
Pourquoi ces techniques ne peuvent-elles se généraliser que maintenant alors que l'on dispose depuis longtemps d'algorithmes de chiffrement dont on peut admettre qu'ils sont largement assez solides pour la sécurité requise dans nos applications ?
Les algorithmes de chiffrement symétrique ne peuvent être déployés dans un contexte ouvert comme l'Internet parce que la question du partage du secret (envoyer la clef de chiffrement) n'est pas facile à résoudre à une grande échelle. Les algorithmes de chiffrement asymétrique visent à résoudre ce problème en particulier parce qu'une infrastructure non sécurisée peut être utilisée pour diffuser des clefs publiques.
La question délicate dans ces techniques est celle de la vérification de la validité des clefs publiques. En effet, il est facile de forger un couple clef publique et clef privée contenant l'identité que l'on souhaite usurper. La solution adoptée est de faire signer sa clef publique par un tiers. Cette signature d'une clef appelée certificat peut être vérifiée, ce qui permet de valider la provenance de la clef publique sous réserve que deux conditions soient respectées :

- pouvoir accéder à la clef publique de l'entité qui a certifié la clef publique que l'on souhaite vérifier,
- savoir quel degré de confiance on peut accorder dans ce tiers certificateur. Tous les certificats n'ont pas la même valeur. De même qu'il est facile de forger une fausse clef publique, il est facile de faire certifier celle-ci par une autorité de certification complaisante.

Jusqu'à présent, c'est ce qui a empêché le déploiement de PGP pourtant déjà ancien (première version stable de PGP en 1991) dont l'usage est resté limité ou presque à des informaticiens.

Quels sont les facteurs nouveaux qui nous font penser que S/MIME ou HTTPS peuvent se déployer largement ?

La législation du cryptage a considérablement évolué en France et partout dans le monde, essentiellement sous la pression du commerce électronique.

Dans PGP la diffusion des clefs publiques n'est pas spécifiée. Il existe cependant des serveurs de clefs publiques de toutes sortes (robot de messagerie, serveurs FTP ou HTTP ; les serveurs de clefs publiques ne garantissent absolument pas la validé des clefs qu'ils diffusent).
Les certificats X509 utilisés sont basés sur des Distinguished Name (DN) et trouvent donc leur place naturellement dans des annuaires LDAP dont le déploiement est en cours. L'absence de cette infrastructure d'annuaire a lourdement handicapé l'utilisation de PGP. On peut penser qu'un nombre relativement important de serveurs HTTPS peut être mis en place dans le seul but de chiffrer les échanges entre le navigateur et le client sans s'appuyer sur un annuaire, par contre, il sera quasiment impossible d'utiliser S/MIME dans une communauté ouverte de correspondant avant que des annuaires LDAP ne soit généralisés.

10 Infrastructure de gestion de clefs

10.1 Composantes et services d'une PKI

La disponibilité d'autorités de certification commerciales ou administratives est l'opportunité pour mettre en œuvre une **infrastructure de gestion de clefs ou Public Key Infrastructure** (PKI).
L'infrastructure de gestion de clefs recouvre l'ensemble des services mis en œuvre pour assurer la gestion complète des clefs publiques.

10.1.1 Autorité d'enregistrement

C'est l'autorité qui reçoit des utilisateurs les demandes de certificats ou CSR (Certificate Signing Request) et en vérifie la teneur. Les méthodes utilisées pour ces vérifications dépendent de la nature du certificat demandé et de la politique de certification choisie. La vérification peut être limitée à l'identité du demandeur, mais on peut aussi vérifier s'il possède bien la clef privée associée, s'il a bien l'autorisation de son organisation pour demander ce type de certificat, etc. Les moyens mis en œuvre pour assurer cette vérification peuvent aller du simple échange de courrier électronique à une véritable enquête effectuée par exemple par les renseignements généraux.

L'enregistrement de la demande de certificat étant accepté, la demande est passée à l'autorité de certification qui elle n'a connaissance que des informations strictement indispensables à l'établissement du certificat.

10.1.2 Autorité de certification

Elle fabrique le certificat en signant la clef publique du demandeur (avec sa clef privée). Le certificat contient le Distinguished Name du demandeur, sa clef publique, une date d'expiration et la destination du certificat, par exemple certificat destiné à un serveur HTTPS ou bien certificat pour signature S/MIME. Bien entendu, la clef privée de l'autorité de certification est un élément vital de l'ensemble de l'infrastructure de gestion de clefs, quiconque accède à cette clef privée peut établir de faux certificats. L'autorité de certification n'est pas obligatoirement connectée sur le réseau Internet.

10.1.3 Service de publication

Le service de publication permet l'accès des utilisateurs aux clefs publiques de leurs correspondants.
L'utilisation du service de publication n'est pas requise pour toutes les applications de chiffrement asymétrique. En particulier, l'accès à un serveur HTTPS dans le but de chiffrer les échanges ou d'authentifier le serveur ne requiert ni un accès au service de publication car le serveur HTTPS communique lui-même son certificat au client, ni un accès à l'autorité de certification dont la clef publique peut accompagner le certificat du serveur HTTPS.
De même, il est possible d'échanger des messages S/MIME sans utiliser le service de publication (l'envoi d'un message signé permet de faire parvenir automatiquement à son correspondant son certificat).
Toutefois, l'utilisation du service de publication est un élément déterminant dès que le nombre d'utilisateurs augmente. L'identité de la personne certifiée est définie dans un Distinguished Name, elle constitue donc une clef d'accès dans l'annuaire LDAP. Par ailleurs LDAP est la seule API normalisée et donc utilisable dans le contexte hétérogène de l'Internet.

10.1.4 Service de révocation

L'utilisation des cartes bancaires serait-elle réellement possible si l'on ne pouvait pas faire opposition sur les cartes perdues ou volées ? De même, il est vital de pouvoir révoquer une clef publique. Dans le cas de la perte de sa clef privée, une personne doit bloquer l'usage de sa clef publique faute de quoi elle ne peut plus lire les messages confidentiels qui lui sont adressés par ses correspondants.
Dans le cas plus grave du vol de la clef privée, le «voleur» peut contrefaire la signature de sa victime et déchiffrer ses messages confidentiels. Le service de révocation doit enregistrer, vérifier les demandes de révocation (l'auteur de la demande est-il bien la personne titulaire de la clef publique ?) et publier une liste des certificats révoqués.

Il appartient aux clients utilisateurs de vérifier les listes de révocation pour les certificats qu'ils utilisent. La révocation est un élément du service de publication. L'accès aux listes de révocation peut être spécifié dans le certificat sous forme d'une URL, mais peu de produits implémentent l'accès aux listes de révocation.
Contrairement aux autres éléments de la PKI, la liste de révocation doit être disponible en «temps réel», de ce fait, cet élément de service de la PKI est le seul obligatoirement connecté à Internet.

10.1.5 Politique de certification

La politique de certification décrit l'ensemble de la procédure qui conduit à certifier une clef publique. Cette politique prend en considération les moyens mis en œuvre pour vérifier les informations constituant le certificat et la destination de celui-ci (certificat personnel pour la signature S/MIME, certificat de serveur HTTPS, etc). Une autorité de certification peut appliquer plusieurs politiques de certification selon les populations et les usages concernés.

Quelques exemples de politiques de certification :
Thawte personnal freemail : certificat gratuit obtenu quasiment sans vérification, le seul élément de preuve de l'identité du demandeur est acquis par échange de mot de passe par courrier. On a donc la preuve que le demandeur a accès à la boite aux lettres dont il prétend être titulaire.
Thawte personnal basic : il faut 100 points pour obtenir ce certificat. Les points sont obtenus auprès d'un «notaire Thawte» qui peut vous attribuer 50 points si vous vous présentez physiquement avec une copie de votre pièce d'identité. Avec 150 points vous devenez notaire. Cet exemple de politique de certification est assez intéressant ; si trois complices malhonnêtes prouvent leur identité, ils peuvent devenir notaires et dès lors enregistrer de fausses demandes de certificats (Thawte propose un troisième niveau de certification *«Thawte personnal premium»).*

Swisskey : ce service suisse vous demande de produire une pièce d'identité en cours de validité auprès d'un représentant de son autorité d'enregistrement c'est à dire auprès de certaines banques (milieu dans lequel on a l'habitude de vérifier l'identité des personnes).

Ces exemples montrent bien que la solidité des algorithmes de chiffrement ou la longueur des clefs utilisées est relativement secondaire devant les aspects organisationnels de la PKI. L'IETF a défini dans le RFC-2560 un formalisme de description d'une politique de certification.

10.1.6 Protection de la PKI

Outre la politique de certification qui est un élément majeur de la confiance que l'on peut accorder à un certificat, la protection de la PKI est fondamentale. Le SCSSI a élaboré des profils de protection correspondant aux impératifs de l'administration française pour les niveaux les plus basiques jusqu'au niveau classifié «secret défense».

Le fait par exemple que la société de certification Certplus confie son *blockhaus* informatique à un fabriquant de cartes bancaires illustre bien les enjeux qui dépendent de cette sécurité.

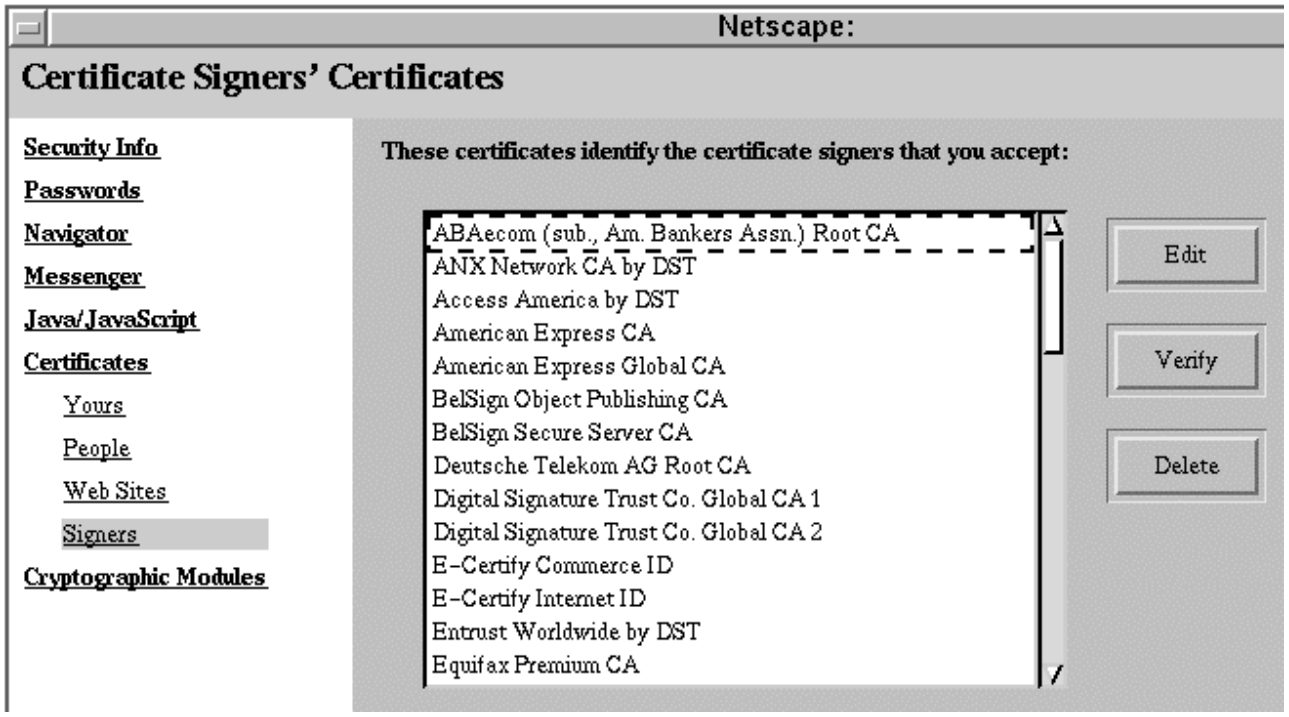
10.1.7 Interopérabilité et PKI

Nous avons montré comment l'infrastructure de gestion de clefs publiques dont la pièce maîtresse est l'autorité de certification permet de résoudre le délicat problème des échanges et vérification des clefs publiques. Comment des correspondants ayant obtenu leurs certificats personnels de messagerie auprès d'autorités de certification distinctes peuvent-ils échanger des courriers signés ou confidentiels ?

Pour vérifier la validité d'un certificat, une application doit :

- soit connaître directement et accorder confiance à l'autorité de certification ayant émis ce certificat (et donc disposer dans sa configuration de la clef publique de cette autorité),
- soit accorder confiance à une autre autorité ayant elle-même certifié la clef publique de l'autorité émettrice du certificat à valider (certification hiérarchisée ou croisée).

Les logiciels de messageries Netscape et Outlook sont pré-configurés pour reconnaître et accepter comme digne de confiance un certain nombre d'autorités de certification. La figure ci-dessous montre le menu «security» de la version française de Netscape Messenger.



Un menu permet d'éditer la liste des autorités reconnues. Il est possible d'ajouter de nouvelles autorités en chargeant dans son navigateur un certificat par exemple via un cgi. L'entête HTTP doit contenir «`Content-Type:application/x-x509-ca-cert`».

Les autorités pré-configurées dans les deux principaux navigateurs du marché le sont parce que des accords commerciaux lient ces autorités et les éditeurs de ces logiciels. Ces accords ne sont absolument pas une garantie de fiabilité de ces autorités, ni du point de vue de leur profil de protection, ni de celui de la politique de certification mise en œuvre. Pourtant, les clients de messagerie traitent indifféremment tous les certificats, quelle que soit l'autorité émettrice. Très peu de personnes disposent d'une information suffisante pour choisir les autorités de certification dignes de confiance.
Du point de vue d'un serveur HTTPS Apache, il est possible de spécifier dans la configuration du serveur quelles sont les autorités reconnues. En pratique, il semble impossible pour un serveur destiné à une large population de refuser les certificats clients émis par des autorités largement déployées. Mais surtout, ce serveur sera tenu d'obtenir (donc de payer) son certificat auprès d'une de ces autorités, faute de quoi, le client est prévenu dans des termes très inquiétants qu'il communique avec un serveur dont on ne peut valider le certificat. Seules des applications d'intranet peuvent peut-être échapper à ce dicta du marché.

10.2 Les navigateurs et serveurs HTTPS (source Thawte)

Navigateurs HTTPS	
Internet Explorer 5.x	Windows 95/98, NT 4.0
Navigator 4.x Communicator 4.x	Toute plate-forme
Internet Explorer 4.x	Windows 95/98, NT 4.0 Windows 3.1, NT 3.51
Internet Explorer 3.02	Windows 95/98, NT 4.0 SP3, Windows 3.1 (IE 3.02a.2916)
WebTV Classic et plus	
Opera 3.x et plus	Windows 3.1, Windows95/98
FrontPage 2000	FrontPage 2000

Répartition des serveurs HTTPS en France	
IIS	46.46%
Apache	22.31%
Netscape	14.17%
Non identifiés	4.20%
Domino	3.41%
Webstar	2.36%
Website pro	2.10%
Stronghold	1.84%
Autres	3.15%

Ces chiffres ont été obtenus en testant systématiquement les URLs de la forme `https://www.domaine.fr` et `https://secure.domaine.fr` soit 350 certificats testés. Seulement 27% des serveurs français utilisent des clefs de 128 bits, les autres étant toujours limités à 40 bits. Apache est utilisé pour 91% des serveurs utilisant les clefs de 128 bits.

10.3 Le projet OpenCA

Le but de ce projet est de fournir à la communauté un ensemble logiciel permettant la mise en œuvre des services de bases d'une infrastructure de gestion de clefs publiques. L'ensemble de ces logiciels étant disponible en «open source». OpenCA est basé en particulier sur Apache, OpenSSL, OpenLDAP et mod_ssl.
Massimiliano Pala, un des auteurs écrit : "this code is completly unfinished, unusable, but it works: the hard part is to guess how to set up the whole thing because of the lack of documentation. I would not release such an incomplete code normally, anyway let's see. Don't expect you'll get easily a working CA ... ".

S'il fallait encore prouver qu'on ne peut faire confiance à n'importe quelle autorité de certification, ce logiciel libre montre :

- que la technique de certification est à la portée de tous les pirates de l'internet ;

- que le nombre d'autorités de certification va augmenter considérablement et que faute d'inscrire des dispositifs techniques dans des projets globaux de PKI, les problèmes d'interopérabilité seront nombreux.

11 Les évolutions législatives

Le Premier Ministre avait annoncé au printemps 99 une libéralisation du chiffrement concernant en particulier :

- augmentation de la taille des clefs utilisables (sauf exportation) ;
- suppression de l'obligation de recourir à un «tiers de confiance» (le tiers de confiance est chargé du séquestre des clefs privées) ;
- instauration d'un label de qualité attribué aux autorités de certification ;
- instauration d'une obligation (assortie de sanctions) de remise aux autorités judiciaires des transcriptions en clair des documents chiffrés ;
- modification du droit de la preuve pour la prise en compte de la signature électronique.

Une partie de ces évolutions était possible par simple décret ; celui du 19 mars 1999 permet l'utilisation de moyens de chiffrement avec des clefs de 128 bits librement pour les personnes privées et sous régime déclaratif pour les personnes morales. [2]

Cette modification spectaculaire du cadre réglementaire est pourtant moins importante que la suppression des tiers de confiance. A noter toutefois que l'esprit de ce qui a été annoncé semble imposer un service d'auto séquestre des clefs privées pour nos établissements. En effet, comment respecter l'obligation de remise aux autorités judiciaires de la transcription lisible des échanges chiffrés sans mettre en œuvre un service de séquestre des clefs privées ?
Cette obligation est cependant beaucoup moins contraignante que les dispositions précédentes, par ailleurs cette obligation permet la mise en œuvre d'un service de restauration de clef privée.

12 Des projets au sein nos administrations

Il n'est pas prévu d'autorité de certification interministérielle mais au sein du MENRT, la direction de l'administration travaille sur trois projets complémentaires.

1. Utilisation de signature S/MIME dans les rectorats. Dans les différents rectorats, vingt mille personnes environ ont des prérogatives administratives pour lesquelles leur signature papier est acceptée. Ce premier projet dont les études se termine (démarrage en janvier 2000) a pour objectif de distribuer vingt milles certificats de messagerie. La structure de la PKI est basée sur des autorités de certification de niveau rectoral et une autorité de certification de l'administration.

2. Projet CPEN Carte à Puce de l'Education Nationale. Ce projet concerne 1 300 000 personnels. Les applications de cette carte à puce concernent l'accès aux données personnelles des agents, l'accès aux mises à jour des annuaires, le partage de ressources pédagogiques ou de gestion etc . Il sera étudié la possibilité de passerelle

[2] Vous pouvez utiliser le formulaire https://www.fortify.net/sslcheck.html pour vérifier la taille des clefs utilisables par votre navigateur

avec les services de fournisseurs d'accès. Le fait d'utiliser une carte à puce permet de stocker d'autres informations que le seul certificat de la personne, par ailleurs, c'est un atout important dans le cas d'un usage «nomade». Cet objet qui ressemble à une carte bancaire, ne devrait pas être perçu comme un nième mot de passe. Cette image de la carte à puce devrait faciliter la prise de conscience des responsabilités liées à son usage. Comme dans le premier projet, la diffusion des clefs publiques est assurée par un annuaire LDAP dont l'étude est très avancée.

3. VPN sur Renater. Le but de ce troisième projet est de remplacer un certain nombre de liens privés par la notion de VPN appliquée sur l'infrastructure de Renater. Ce réseau privé virtuel disposera de service de chiffrement.

Actuellement, les trois projets ne concernent pas directement les universités, cependant l'existence dans chaque académie d'une autorité de certification accréditée par celle du ministère et l'organisation en parallèle de serveurs LDAP constitueront un moteur important pour les projets de PKI que les universités pourraient adopter.

BIBLIOGRAPHIE

[BOUCQUEAU1997] BOUCQUEAU J.-M, DELAIGLE J .-F, DHEM J.-F , JOYE M., KOEUNE F., MASSIAS H., MESTRE P., QUISQUATER J.-J, Comment jouer à pile ou face sur Internet sans tricher, Université Catholique de Louvain, 1997

[CISCO1998] CISCO Systems, White Paper IPSec, 1998

[COHEN1995] COHEN Jo, La cryptographie au coeur des réseaux, O1 Réseaux, Juin 1995

[DELEURENCE1996] DELEURENCE Guillaume, Une sécurité presque parfaite, Réseaux & Télécoms, Avril 1996

[EUROPE1997] Commission Européenne, Direction générale XII, Télécommunication, Marché de l'information et valorisation de la recherche, Assurer la sécutité et la confiance dans la communication électronique, Vers un cadre Européen pour les signatures numériques et le chiffrement, 1997

[FAHN1993] FAHN Paul, Answers to frequently asked questions about today's cryptography, RSA laboratories, version 2.0, draft 2f, Septembre 1993

[FERRET1997] FERRET Bruno, L'authentification, clé de voute de la sécurité, Décision Micro & Réseaux N° 293, Avril 1997

[FISCHER1994] FISCHER M, How to implement the Data Encryption Standard (DES) - A step by step tutorial version 1.3, 1994

[FLORIN] FLORIN Gérard NATKIN S., La sécurité, CNAM - CEDRIC

[GARFINKEL1995] GARFINKEL Simson, PGP Pretty Good Privacy, O'Reilly & Associates, Août 1995

[HTTP] BERNERS-LEE T., FIELDING R. and FRYSTYK H., The Hypertext Transfer Protocol, RFC 1945, Mai 1996.

[KARVE1997] KARVE Anita, Public Key Cryptography, LAN Magazine, page 23, Avril 1997

[KUPARINEN1998] KUPARINEN Martin, ISAKMP and IKE, 27 novembre 1998

[LABOURET1999] LABOURET Ghislaine, IPSec : présentation technique, Hervé Schauer Consultants, Avril 1999

[MIME] FREED, N. and BORENSTEIN N., "MIME Part 1: Format of Internet Message Bodies", RFC 2045, Novembre 1996.

[PKCS7] RSA Laboratories, "PKCS #7: Cryptographic Message Syntax Standard", Novembre 1993.

[PKCS12] RSA Laboratories, "PKCS #12: Personnal Information Exchange Syntax", Juin 1999.

[ROUSSEAU1996] ROUSSEAU Ludovic, Cours Sécurité V1.1, Bordeaux 20 et 21 janvier 1997, CNAM/CEDRIC, Décembre 1996

[SCHNEIER] SCHNEIER Bruce, Cryptographie appliquée, 2ième Edition, New York : John Wiley, 1996

[SMIME] DUSSE S., HOFFMAN P., RAMSDELL B., LUNDBLADE L. and REPKA L., "S/MIME Version 2 Message Specification", RFC2311, Mars 1998.

[TANENBAUM1997] TANENBAUM A., Réseaux - Architecture, Protocoles, Applications, InterEditions, 3ième Edition, Juillet 1997

[SCHNEIER] SCHNEIER Bruce, Cryptographie appliquée, 2ième Edition, New York : John Wiley, 1996

[SSLV3] FREIER A., KARLTON P. and KOCHER P., "The SSL Protocol Version 3.0", Internet Draft, Transport Layer Security Working Group, Novembre 1996.

URL

[MODSSL] http://www.modssl.org

[OPENCA] http://www.openca.org

[OPENSSL] http://www.openssl.org

[SSLV2] http://www.netscape.com/eng/security/SSL_2.html

[WEBMAIL] http://www.cru.fr/http-mail

[SCSSI] http://www.SCSSI.gouv.fr/document/igc.html

[WRAPSSL] http://www1.kappa.ro/~drazvan