Sécurité des échanges. Conception et validation d'un nouveau protocole pour la sécurisation des échanges.

Ibrahim Hajjeh

ECOLE NATIONALE SUPERIEURE DES TELECOMMUNICATIONS
PARIS

MEMOIRE

Présenté en vue d'obtenir le grade de docteur de l'Ecole Nationale Supérieure des Télécommunications

Spécialité informatique et réseaux

Par

Ibrahim HAJJEH

Sécurité des échanges.
Conception et validation d'un nouveau protocole pour la sécurisation des échanges.

Soutenue le 7 décembre 2004 devant le jury composé de

Frédéric Cuppens (ENST Bretagne)	Président
Pascal Lorenz (Université de la Haute Alsace)	Rapporteurs
Jean-Pierre Thomesse (Institut Nationale Polytechnique de Loraine)	
David Arditti (France TELECOM R&D)	Examinateurs
Hossam Afifi (INT Evry)	
Michel Diaz (LAAS CNRS)	
Vincent Roca (INIRA Rhones Alpes)	
Ahmed Serhrouchni (ENST Paris)	Directeur de thèse

A la mémoire de mon père,
A ma mère,
A ma sœur et mes frères.

Learning is a treasure that will follow its owner everywhere.
Proverbe Chinois

Remerciements

C'est avec mon enthousiasme le plus vif et le plus sincère que je voudrais rendre mérite à tous ceux qui à leur manière m'ont aidé à mener à bien cette thèse.

Mes premiers remerciements sont adressés tout d'abord à mon directeur de thèse Ahmed Serhrouchni pour avoir accepté de me diriger patiemment et pour son soutien constant pendant mes trois années de thèse. Il fut pour moi un directeur de thèse attentif et disponible malgré ses charges nombreuses. Sa compétence, sa clairvoyance, son charisme, son dynamisme m'ont beaucoup appris. Ils ont été et resteront des moteurs dans mon travail.

Je souhaite exprimer ma gratitude envers Monsieur Frédéric Cuppens, Professeur à l'Ecole Nationale Supérieure des Télécommunications de Bretagne, qui m'a honoré en acceptant d'être président de mon jury de thèse.

Je remercie Monsieur Pascal Lorenz, Professeur à l'Université de Haute Alsace et Monsieur Jean Pierre Thomesse, Professeur à l'Institut Nationale Polytechnique de Loraine, pour avoir accepté la charge d'être rapporteurs de ma thèse.

Je remercie également Monsieur Michel Diaz, Directeur de Recherche au Centre National de Recherche Scientifique LAAS, Vincent Roca, Maître de Conférences à l'INRIA Rhône Alpes, Hossam Afifi, Professeur à L'INT Evry et à David Arditti, Directeur de Recherche et de Développement à France Télécom, qui m'ont honoré en acceptant d'être examinateurs de ma thèse.

Je remercie très vivement tous les membres du département Informatique et Réseaux de Télécom Paris (ENST). Merci à mes camarades, Ashraf, Dany, Dalila, Fadi, Habib, Mejdi, Rana, Rami, Rani, Tarek, Tranh et Xiaouyn, qui ont contribué à rendre mon séjour et mes études agréables et enrichissants.
Merci à mes amis pour l'affectueuse amitié dont ils ont toujours fait preuve. Je pense particulièrement à Ahmed, Rola et Salim.

Un merci tout particulier à mes chers amis Mohamad, Ouahiba et Jacques. Leur amitié, leur confiance en moi et leur soutien généreux m'ont été indispensables. Je garde pour eux et pour toujours, une place particulière dans mon coeur.

Les mots les plus simples étant les plus forts, j'adresse toute mon affection à ma famille, et en particulier à ma mère, à ma soeur et à mes frères. Malgré mon éloignement depuis de nombreuses années, leur confiance, leur tendresse, leur amour me portent et me guident tous les jours.

Enfin, je réserve une reconnaissance particulière à mon cher défunt père, avec la plus grande douleur de ne plus l'avoir parmi nous, pour l'amour et le soutien incomparable dont il m'a fait preuve depuis ma naissance.

Résumé

De nombreux mécanismes de sécurité ont été proposés pour les réseaux fixes et mobiles. Bien que ces mécanismes aient pu répondre à un ensemble d'exigences de sécurité, ils demeurent uniquement efficaces dans un contexte spécifique lié aux hypothèses et aux exigences restrictives qui ont été émises lors de la conception.

Dans un premier temps, nous définissons une liste d'exigences en sécurité qui permet d'analyser les solutions de sécurité les plus déployées, à savoir les protocoles SSL/TLS, IPsec et SSH. Ceci va nous permettre de comparer ces protocoles et de pouvoir aborder d'une part, leur avantages et inconvénients et d'autre part, les exigences qu'ils ne peuvent pas satisfaire.

Dans un second temps, nous proposons d'étendre le protocole SSL/TLS avec de nouveaux services. SSL/TLS est une solution de sécurité générique et transparente aux applications basées sur le protocole de transport TCP. Ainsi, il ne couvre pas les besoins spécifiques à certaines classes d'application telles que les applications de paiement sur Internet. Il ne fournit pas non plus ni le service de protection d'identité des usagers ni le service de contrôle d'accès.

Notre propositions d'intégrer le protocole d'échange des clés ISAKMP (Internet Security Association and Key Management Protocol) avec SSL/TLS permet de fournir, entre autres, la protection d'identité des utilisateurs et l'unification des associations de sécurité. Nous prouvons les capacités non exploitables dans le protocole ISAKMP telles que l'utilisation des certificats d'attributs pour l'authentification.
Afin d'étendre l'utilisation de SSL/TLS vers les systèmes de paiement sur Internet, nous intégrons un module générique pour la génération d'une preuve de non répudiation dans le protocole SSL/TLS. Ce module est interopérable avec les deux standards SSL/TLS et TLS Extensions.

Par ailleurs, toutes ces propositions restent, cernées par le problème d'interopérabilité avec les anciennes versions de chaque protocole. Ce qui rend la satisfaction de tous les besoins à travers un des protocoles existants irréalisables. Pour cela, nous proposons finalement de concevoir et de valider un nouveau protocole de sécurité qui intègre nativement l'évolution des protocoles de sécurité des échanges et des réseaux d'une manière performante. Nous avons appelé ce protocole SEP pour *Secure and Extensible Protocol*. SEP permet l'établissement d'une session sécurisée par l'intermédiaire d'une autorité de confiance (IA), la mise en œuvre d'un VPN (Virtual Private Network) au niveau de données et l'autorisation.

Mots clés

Protocole de sécurité, services de sécurité, sécurisation des échanges, sécurisation des réseaux, protocole de gestion des clés, protocole d'authentification.

Abstract

Many security mechanisms have been proposed for wired and wireless networks. Although these mechanisms have answered some security requirements, they remain efficient in a specific context related to the assumptions and the restrictive requirements which have been emitted at the time of their design.

Firstly, we define a list of security requirements which make it possible to analyze the most deployed security solutions, namely, SSH, SSL/TLS and IPsec protocols. This will allow us to compare these protocols and to be able to present their advantages, limitations and the security requirements that they cannot satisfy.

Secondly, we propose to extend the SSL/TLS protocol with new services. SSL/TLS is a transparent security solution to applications which are natively based on the TCP transport protocol. Thus, security services provided to applications are the same. SSL/TLS does not meet specific needs to some classes of applications such as internet payment applications. Also, it does not provide either identity protection or access control.

We integrate the Internet Security Association and Key Management Protocol (ISAKMP) in SSL/TLS to provide, among others, identity protection and unification of security associations. This work allows us to exhibit some non exploitable capacities of the ISAKMP protocol such its use of attribute certificates in authentications.
In order to extend the use of SSL/TLS towards the Internet payment systems, we integrate a generic signature module in SSL/TLS that generate a non repudiation proof over all exchanged data. This module is interoperable with SSL/TLS and TLS Extensions standards.

However, all these proposals suffer from the lack of interoperability with their previous versions. This will make it impossible to satisfy all the security needs through one existing protocol. Thus, we propose to design, validate and develop a new security protocol, which natively integrates the evolutions of the security protocols, in a powerful and elegant way. We called this protocol SEP for *Secure and Extensible Protocol*. SEP allows the establishment of a secure session made with the presence of an intermediate authority (IA), the implementation of a virtual private network (VPN) on the exchange level and the authorization.

KeyWords

Security protocols, security services, transport security, network security, key exchange protocol, authentication protocol

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

1 Introduction et contributions de cette thèse

1.1 Introduction et problématique

Le monde des télécommunications connaît une rapide évolution. Il existe actuellement une multitude de réseaux qui diffèrent entre eux par leurs topologies, les protocoles de communication qu'ils utilisent, la nature des données à transmettre, les services offerts, etc.
Actuellement, la sécurité est amenée à être l'une des priorités dans ces architectures et infrastructures de communication. Cette sécurité est d'autant plus importante qu'on observe une migration d'un nombre important de services dont la téléphonie sur des infrastructures IP.

Lorsqu'on parle de la sécurité des réseaux, on évoque généralement les protocoles (ou les solutions) de sécurité et les services qu'assurent ces protocoles.
Ces dernières années, beaucoup d'efforts ont été consentis pour aboutir à des solutions immédiates destinées à différents environnements. Ces solutions de sécurité qu'on appellera classiques, se veulent dans beaucoup de cas génériques et répondent ainsi insuffisamment aux besoins spécifiques des applications de communication. Ceci est dû au fait que la plupart de ces solutions offrent les mêmes services de sécurité (authentification, confidentialité et intégrité) à toutes les applications. Il existe actuellement deux grandes classes de solutions :

- La première classe où la ***sécurisation des réseaux*** s'applique au niveau trois du modèle de référence de l'OSI.

- La deuxième classe où la ***sécurisation des échanges*** s'applique au niveau sept du modèle de référence de l'OSI et qui est relativement plus simple à déployer.

De nos jours, différents protocoles sont mis en place. Dans le ***domaine de la sécurisation des échanges***, on trouve le protocole SSL/TLS (Secure Socket Layer/Transport Layer Security) [SSL3] [RFC2246] qui offre les mêmes services à toutes les applications de l'Internet. Cependant, il ne fournit pas les services de non répudiation et de contrôle d'accès. Dans le même domaine, on peut citer le protocole SSH (Secure Shell) [SSH-Arch] qui offre une session interactive sécurisée entre un client (un utilisateur ou une machine) et une machine distante (serveur). Ce protocole ne supporte pas actuellement la distribution des clés par un tiers de confiance.

Dans le domaine de la ***sécurisation des réseaux***, nous pouvons citer le protocole IPSec (IP Security) [RFC2401] qui est intégré aux systèmes d'exploitation et il est ainsi tributaire des fournisseurs de ces systèmes. La mise en œuvre de IPSec est basée sur une approche de *politique de sécurité* qui nécessite une maîtrise globale des flux d'informations. Cette approche s'adapte bien à une politique globale de sécurisation de réseaux. Cependant, elle est plus complexe à l'échelle des équipements. La valeur ajoutée significative de IPSec est sa

capacité à mettre en œuvre *un réseau privé virtuel* [Kols02] qui peut être implémenté d'une façon indépendante de l'application.

La liste des solutions de sécurité est encore longue mais nous nous limitons dans notre thèse à l'étude de ces trois protocoles (SSH, SSL/TLS et IPSec) ainsi que leur mécanismes d'authentification et d'échange des clés.

Certains objectifs de ces solutions sont tout à fait complémentaires. Par exemple, pour l'authentification au niveau réseau, ce sont les machines qui sont authentifiées, alors qu'au au niveau des données, ce sont les utilisateurs. Néanmoins, elles contiennent des fonctions redondantes. A titre d'exemple, l'intégrité et la confidentialité des données sont assurées en même temps au niveau trois et sept de la couche OSI. Ceci réduit les performances des logiciels de communication. D'autre part, nous soulignons le problème de gestion des clés réparties dans différents niveaux des couches OSI et qui doit interagir avec les protocoles de sécurité.
Au-delà de la redondance des fonctionnalités, il y a une absence de réponses à des besoins de services de sécurité nécessaires pour la mise en œuvre des réseaux et des applications particulières.

1.2 Besoins de nouveaux services et protocoles

A nos jours, Internet a franchi de nombreuses étapes avec le déploiement de plusieurs protocoles de sécurité. Actuellement et d'une manière plus précise, les applications essayent de choisir, suivant les particularités de chaque protocole de sécurité, celles qui correspondent le plus à leurs besoins. Toutefois, la nomadicité des utilisateurs et l'interconnexion des réseaux hétérogènes induit à la nécessité de nouveaux services tels que la connexion sécurisée à distance, l'autorisation et la protection d'identité.

L'interconnexion de ressources privées sur une infrastructure publique Internet et la nomadicité des utilisateurs ont poussé les entreprises à réorganiser leurs systèmes d'information afin d'adopter des techniques liées au *réseau privé virtuel*.
Même si ce service est déjà intégré dans quelques solutions de sécurité tels que le protocole IPSec, il restera difficile de limiter l'accès de certains utilisateurs à un certain nombre de ressources d'information. Un de nos objectifs dans cette thèse est d'intégrer ce service au niveau des données afin d'assurer des mécanismes plus flexibles pour *l'autorisation* et *le contrôle d'accès*. Nous utilisons pour cela, les certificats d'attributs que nous avons développés dans le projet RNRT ICARE [ICARE] qui permet d'offrir une méthode flexible pour la *délégation des rôles* entre les différents acteurs.

Par ailleurs, puisque le contrôle d'accès doit être basé sur des mécanismes d'authentification sûrs, il oblige les utilisateurs à fournir une identification et à la prouver [Samf96]. De ce fait, la filature de l'utilisateur devient aisée car un tiers malveillant peut faire une corrélation entre l'identité de l'utilisateur et un certain nombre d'informations telle que la localisation courante de l'utilisateur, son nom, son mail, etc. Un malveillant pourrait ensuite utiliser ces données de manière à porter préjudice à l'utilisateur. Nous estimons que l'accès illicite à ces informations privées, sans le consentement de l'individu, constitue une véritable atteinte à sa vie privée. De ce fait vient notre proposition d'offrir le service de *protection d'identité* avec les protocoles de sécurité.

Les solutions de sécurité doivent être plus adaptées aux besoins spécifiques des utilisateurs ainsi que leurs environnements. Dans ce contexte vient notre première contribution pour étendre le protocole SSL/TLS avec les nouveaux services cités précédemment. Ainsi, nous intégrons les services de protection d'identité et d'autorisation dans le protocole SSL/TLS. Ceci est obtenu à travers le protocole ISAKMP [RFC2408]. Nous étudions également la génération d'une preuve de non répudiation avec SSL/TLS.

Toutes ces propositions restent, par ailleurs, limitées par le problème d'interopérabilité avec les anciennes versions de chaque protocole. Ce qui rend la satisfaction de tous les besoins à travers un des protocoles existants irréalisables. Pour cela, nous proposons un nouveau protocole de sécurité que nous appelons SEP pour *Secure and Extensible Protocol*. SEP est un protocole de sécurité indépendant des infrastructures réseaux dont le but principal est de remplacer le protocole SSL/TLS en lui ajoutant de nouvelles fonctionnalités.
SEP partage avec les autres solutions certains services tels que l'authentification, l'intégrité et la confidentialité. Cependant, il possède ces propres avantages qui le diffèrent des solutions existantes. Ces principales fonctionnalités sont :

1. *La protection au niveau des données, indépendamment de l'infrastructure réseau utilisée.*
2. *La protection d'identité de l'usager.*
3. *La possibilité d'échanges par l'intermédiaire d'un tiers.*
4. *La gestion des habilitations entre acteurs.*
5. *La mise en œuvre dynamique des mécanismes de filtrage et de contrôle d'accès.*
6. *La flexibilité des méthodes d'authentification des usagers.*

1.3 Contributions de cette thèse

Motivé par la nécessité de la conception d'une solution *sécurisée et flexible*, notre travail vient répond aux nouveaux besoins des applications. Les principales contributions de cette thèse sont les suivantes :

1. Analyse des solutions existantes

Dans la première partie de cette thèse, nous faisons une analyse critique des trois principales solutions de sécurité à savoir : IPSEC, SSL/TLS et SSH. Nous commençons notre étude par la définition d'un ensemble de critères de comparaison et de besoins en sécurité que nous souhaitons étudier à travers ces protocoles.

2. Intégration du protocole ISAKMP dans SSL/TLS

La deuxième contribution porte sur l'intégration du protocole d'authentification et d'échange des clés ISAKMP dans le protocole de sécurité SSL/TLS. Une telle intégration a comme premier avantage, l'utilisation d'un mécanisme générique de gestion des clés pour un ensemble de protocoles de sécurité. Notre approche permet aussi à SSL/TLS de supporter le service d'anonymat et de contrôle d'accès à travers les certificats d'attribut.

3. Intégration d'un module générique de non répudiation dans le protocole SSL/TLS

Dans notre troisième contribution, nous proposons d'intégrer un module générique de non répudiation dans le protocole SSL/TLS. Par le caractère modulaire de ce protocole, il est possible d'introduire un nouveau module pour assurer le service de non répudiation et de

signature de données. Ceci est réalisé d'une manière totalement transparente et tout en préservant l'interopérabilité avec les implantations standard de SSL/TLS.

4. ***Proposition et validation d'un nouveau protocole de sécurité au niveau des échanges.***

Notre dernière contribution réside dans la conception d'une nouvelle architecture et d'un nouveau protocole de sécurité appelé SEP (Secure and Extensible Protocol). Ce dernier a comme objectif de fédérer un ensemble de services de base et de nouveaux besoins en sécurité dans une architecture flexible et sécurisée. Afin de valider cryptographiquement ce protocole, nous utilisons une technique formelle basée sur le Modèle de Millen-Rueß [MR00] et l'outil Hermes de EVA [Bozg03] [EVA].

1.4 Plan de ce mémoire de thèse

Cette thèse est organisée en trois parties principales : la première partie (chapitres 2 et 3) passe en revue les principales solutions de sécurité. Cette étude définit les besoins et les caractéristiques des solutions de sécurité en particulier, les trois principaux protocoles à savoir : IPSec, SSL/TLS et SSH. Cette étude nous permet de situer notre démarche dans le cadre des travaux menés jusqu'à présent et de dégager les limites et les faiblesses des mécanismes de sécurité existants.

La deuxième partie (chapitres 4 et 5) décrit nos deux propositions pour étendre les solutions existantes. Nous développons dans le chapitre 4 notre approche pour l'intégration d'ISAKMP dans le protocole SSL/TLS. Cette approche est suffisamment flexible pour prendre en compte les différents types d'authentification, de contrôle d'accès et de protection d'identité. Dans le chapitre 5, nous nous intéressons au problème de la génération d'une preuve de non répudiation avec le protocole SSL/TLS. Une solution intuitive à ce problème est l'intégration d'un module générique de non répudiation dans le protocole SSL/TLS. Cette méthode opère d'une manière transparente par rapport aux applications.

La troisième partie (chapitres 6, 7 et 8) présente la conception, la vérification et l'évaluation d'un nouveau protocole de sécurité que nous appelons SEP pour Secure and Extensible Protocol. Dans le chapitre 6 nous commençons par définir les exigences que nous souhaitons satisfaire dans SEP. Ensuite, nous développons les différentes fonctionnalités de notre protocole ainsi que ses services de base. Le chapitre 7 s'intéresse à la validation cryptographique du protocole SEP. Les résultats obtenus par l'utilisation d'un modèle de vérification automatique des protocoles cryptographiques (LEVA), montrent que notre protocole satisfait toutes les exigences cryptographiques définies dans les chapitres précédents. Enfin, dans le chapitre 8, nous concluons sur l'intérêt de notre travail et ses perspectives.

Le lecteur trouvera dans la première annexe la liste des acronymes utilisés dans cet ouvrage. La deuxième annexe donne la spécification en LEVA de quelques protocoles de sécurité. La troisième annexe décrit la spécification en XDR [RFC1832) et en DTD [W3Cdtd] des différents services SEP. Une liste des publications associées à cette thèse est présentée à la fin de cet ouvrage. Une Conclusion figure à la fin de chaque chapitre. Nous souhaitons qu'elle permette une relecture rapide du document.

Première Partie

Définitions des exigences pour les protocoles de sécurité et analyse des solutions existantes.

Chapitre 2

2 Définition des exigences pour les protocoles de sécurité

2.1 Résumé

Dans ce chapitre, nous définissons les exigences qu'un protocole de sécurité doit satisfaire. Une étude détaillée est présentée pour comparer les différentes solutions existantes afin d'identifier les besoins non couvertes par ces solutions.

2.2 Introduction

Dans le domaine de la sécurité dans les réseaux, il existe plusieurs solutions de sécurité telles que le protocole IPSec (IP Security), le protocole SSL/TLS (Socket Secure Layer/Transport Layer Security) et le protocole SSH (Secure Shell). Actuellement, il ne s'agit plus uniquement de chercher à faire de nouveaux développements afin que le réseau soit plus fiable ou d'une manière générale soit plus sécurisé dans son fonctionnement global. Il faudrait adapter ces solutions aux besoins spécifiques des utilisateurs ainsi qu'à leurs environnements. Dans ce contexte, on peut citer les travaux de l'IETF pour étendre le protocole SSL/TLS vers les environnements mobiles [RFC3546] et le travail d'amélioration et de simplification du protocole IKE [IKEv2].
Actuellement, les applications essayent de choisir suivant les particularités de chaque protocole de sécurité, celles qui correspondent le plus à leurs besoins.
Par conséquence, la conception ou le développement d'un nouveau protocole de sécurité exige une définition claire des exigences et des services que peut présenter ce protocole.
Pour cela, un nouveau protocole de sécurité peut partager avec les autres solutions certains services tels que l'authentification, l'intégrité et la confidentialité mais il doit aussi avoir ses propres avantages qui le diffèrent des solutions existantes.

Ce chapitre a pour but de définir les caractéristiques de base des différentes solutions de sécurité existantes ainsi que les exigences que nous souhaitons satisfaire à travers nos différentes contributions dans cette thèse.

2.3 Les exigences en terme de sécurité

Nous avons relevé certaines exigences nécessaires au bon fonctionnement d'un protocole de sécurité. Dans la suite de ce chapitre, nous présentons ces exigences.

1. Le choix des mécanismes cryptographiques

Dans la plupart des protocoles de sécurité, on peut diviser la cryptographie en deux grandes catégories qui dépendent de la phase d'initialisation du protocole et de la phase «de protection de données».

La première phase est généralement basée sur le chiffrement asymétrique (clé publique/privée) alors que la deuxième est basée sur un chiffrement symétrique (clé secrète).
Comme la technologie du chiffrement asymétrique est moins performante que celle du chiffrement symétrique (tableau 2-1), on utilise en général le chiffrement symétrique pour chiffrer les données ou l'information dont on doit protéger la confidentialité et l'intégrité. On utilise le chiffrement asymétrique pour assurer la distribution sûre des clés symétriques dont on doit protéger la confidentialité.
A travers la phase d'initialisation, une liste d'algorithmes de chiffrement, de hachage et de signature est négociée entre les deux communicateurs. Un protocole de sécurité doit assurer la négociation de cette liste mais en incluant seulement des algorithmes jugés sûrs.
Pour les algorithmes à chiffrement symétrique, une attaque exhaustive est hors de porté dès que l'espace des clés est suffisamment grand (minimum 64 bits) [Cant01] [Han99]. Pour les algorithmes à chiffrement asymétrique, leurs sécurités reposent principalement sur la factorisation du grand entier (n). Le plus grand nombre ordinaire factorisé à ce jour est un nombre de 576 bits établi en décembre 2003 [RSA03] [Rive78].

Algorithme	**Durée du traitement en seconde (s)**	
	Chiffrement	**Déchiffrement**
Algorithme symétrique DES (Data Encryption Standard), clé à 64 bits	3241 Koctets/s	3333 Koctets/s
Algorithme symétrique 3DES, clé à 112 bits	1596 Koctets/s	1620 Koctets/s
Algorithme asymétrique RSA (Rivest, Shamir and Adelman), clé à 1024 bits	4.23 Koctets/s	2.87 Koctets/s
Fonction de Hachage	**Création d'un condensât**	
MD5	36 250 Koctets/s	
SHA	36250 Koctets/s	

Tableau 2-1. Durée de traitement de quelques algorithmes cryptographiques[1]

2. L'authentification et l'identification

Le service d'authentification permet d'assurer qu'une communication est authentique. On peut distinguer deux types d'authentification: l'authentification d'un tiers et l'authentification de la source des données.
L'authentification d'un tiers consiste pour ce dernier à prouver son identité. L'authentification de la source des données sert à prouver que les données reçues viennent bien d'un tel émetteur déclaré. Les signatures numériques peuvent aussi servir à l'authentification. La signature numérique sera abordée dans la section Intégrité.
L'authentification nécessite de fournir une identification et de la prouver. Sur la plupart des réseaux, le mécanisme d'authentification utilise une paire *code d'identification /mot de passe*. Cependant, en raison de la vulnérabilité constamment associée à l'utilisation des mots de passe, il est souvent recommandé de recourir à des mécanismes plus robustes tels que l'authentification par des certificats [ISO-9594], des clés publiques [Rive78] ou à travers des centres de distribution des clés [RFC1510].

[1] Note : ce benchmarks est pris sur un Pentium II cadencé à 233 MHz, API BASE 4.0 de RSA [Sher92]

3. L'intégrité

L'intégrité se rapporte à la protection contre les changements et les altérations. Il y a une intégrité si les données émises sont identiques à celles reçues. Des différences peuvent apparaître si quelqu'un tente de modifier ces données ou tout simplement si un problème de transmission/réception intervient.
Les techniques utilisées pour faire face à cela sont, les bits de parité, les checksums ou encore les fonctions de hachage à sens unique [RFC2104]. Ces mécanismes ne peuvent cependant pas garantir absolument l'intégrité. Il est possible en effet, que les données altérées aient la même somme de contrôle. Il est aussi possible qu'un attaquant modifie les données et recalcule le résultat de la fonction de hachage (empreinte). Pour que seul l'expéditeur soit capable de modifier l'empreinte, on utilise des fonctions de hachage à clés secrètes ou privées. Dans ce cas, on garantit à la fois l'intégrité et l'authentification [Schn95]. Ces deux services de sécurité sont souvent fournis par les mêmes mécanismes pour la simple raison qu'ils n'ont de sens qu'accompagnés l'un de l'autre (dans le contexte d'un réseau peu sûr).

4. La confidentialité

La confidentialité est un service de sécurité qui consiste à assurer que seules les personnes autorisées peuvent prendre connaissance des données. Pour obtenir ce service, on utilise généralement le chiffrement des données concernées à l'aide d'un algorithme cryptographique.
Si seules les données sont chiffrées, une oreille espionne peut tout de même écouter les informations de l'en-tête. Elle peut ainsi, à partir des adresses source et destination, identifier les tiers communicants et analyser leur communication : fréquence des envois, quantité de données échangée, etc. On parle de protection contre l'analyse de trafic lorsqu'en plus de la confidentialité, on garantit l'impossibilité de connaître ces informations.

L'authentification, l'intégrité et la confidentialité vont souvent ensemble et offrent la base des services de sécurité.

5. La non répudiation

La non répudiation empêche tant l'expéditeur que le destinataire de nier avoir transmis un message. On dénombre deux types de service de non répudiation [Sher92] :

1. La non répudiation de l'origine qui protège un destinataire confronté à un expéditeur niant avoir envoyé le message.
2. La non répudiation de la réception qui joue le rôle inverse du précédent, à savoir démontrer que le destinataire a bien reçu le message que l'expéditeur lui a envoyé.

Dans le cadre de la cryptographie à clé publique, chaque utilisateur est le seul et unique détenteur de la clé privée. Ainsi, tout message accompagné par la signature électronique d'un utilisateur ne pourra pas être répudié par celui-ci, à moins que tout le système de sécurité n'ait été pénétré.
A l'opposé, la non répudiation n'est pas directement acquise dans les systèmes utilisant des clés secrètes. La clé de chiffrement étant distribuée par le serveur de distribution de clés aux deux parties, un utilisateur peut nier avoir envoyé le message en question en alléguant que la clé secrète partagée a été divulguée soit par une compromission du destinataire, soit par une attaque réussie contre le serveur de distribution de clés.
La non répudiation de la réception peut se faire en obligeant le destinataire à envoyer un accusé de réception signé et horodaté.

6. La protection contre les attaques actives et passives

La catégorie principale d'attaques sur les protocoles de sécurité est l'attaque *active*. Cette attaque implique certaines modifications du flot de données ou la création d'un flot frauduleux [Stal02]. On remarque principalement dans cette catégorie les attaques suivantes:

- L'*homme du milieu* : cette attaque a lieu lorsqu'une entité prétend être une autre entité. Dans la plupart du temps, l'attaquant utilise les techniques de détournement de flux pour rediriger les flux des deux bouts de la communication vers lui. Ceci, afin de surveiller tout leur trafic réseau et de le modifier à sa guise pour l'obtention de tout type d'information.
- Le *rejeu des paquets* : le rejeu implique la capture passive de données et leur retransmission ultérieure en vue de produire un effet non autorisé. Pour empêcher tel type d'attaques, on utilise souvent les nombres aléatoires dans les messages envoyés.
- La *modification des messages* : ceci signifie que certaines portions d'un message légitime sont altérées ou que les messages sont retardés ou réorganisés. Ce type d'attaques est souvent utilisé avec les protocoles qui sont basés sur le protocole de transport UDP (comme le cas du protocole ISAKMP).
- Le *déni de service* : cette attaque porte bien son nom puisque qu'elle aboutira à l'indisponibilité du service (saturation des ressources allouées à une application spécifique), de la machine visée ou même la saturation d'un réseau complet. Cette attaque vise surtout l'exploitation d'une mauvaise implémentation d'un protocole ou à des faiblesses de celui-ci. Pour les protocoles de sécurité, le déni de service peut prendre deux formes :
  - La première forme vise la vulnérabilité des protocoles de transport qu'ils utilisent tels que les attaques sur le protocole IP (IP Spoofing) [Wrig94], le protocole TCP (SYN Flooding) [CA-96.21] [Sprin02] et le protocole UDP (UDP Flooding, Packet Fragment) [Strei02].
  - La deuxième forme porte surtout sur la phase d'initialisation de ces protocoles (le protocole IKE, le Handshake de SSL/TLS, etc.).

Une deuxième catégorie d'attaques est l'attaque *passive*. Cette catégorie est souvent plus difficile à détecter car elle ne cause aucune altération des données. Le but de l'adversaire est de collecter les informations qui ont été transmises (adresse IP, port, services, environnement, etc.) afin d'analyser leur contenus et de mener par la suite des attaques *actives*.

7. La protection d'identité

La vérification efficace des événements liés à la sécurité se fonde aussi sur la capacité d'identifier chaque utilisateur.

Il est très important que chaque utilisateur de l'Internet ait une identité distincte. L'identité distincte de l'utilisateur est une combinaison qui donne le nom de l'utilisateur et possiblement celui de son ordinateur, de son organisation et de son pays.

Comme nous avons expliqué dans le chapitre 1, la connaissance de ces informations par un tiers malveillant peut être considérée comme une véritable atteinte à la vie privée des usagers. Pour cela, un nouveau défi présent actuellement dans le domaine de sécurité, est la protection de ces informations privées. Ceci nous permet de protéger le trafic réseau contre les malveillants qui comptent analyser tout type d'informations (algorithme, nom de l'utilisateur, environnement, etc.) afin d'intercepter les communications.

Dans la plupart des cas, l'identité du récepteur (généralement un serveur) est publique. Pour cela, un protocole de sécurité doit surtout protéger l'identité de l'initiateur (généralement les clients).

8. La protection d'échanges ou la protection du réseau

Par rapport aux couches OSI, les protocoles et services de sécurité sont insérés à divers emplacements de la pile de communication (tableau 2-2).
Le choix de l'emplacement dépend des exigences en matière de sécurité, c.-à-d. les menaces qui peuvent être rencontrées. Chaque emplacement offre des avantages et des inconvénients. Le chiffrement au milieu de la pile de communication, offre un service de chiffrement indépendant des applications et il est transparent pour celles-ci. Les données encapsulées provenant des couches supérieures sont chiffrées et la confidentialité des en-têtes des protocoles des couches supérieures (transport, présentation ou TCP) est protégée.
Toutefois, le chiffrement au niveau applicatif préserve la possibilité de transmettre des données par l'intermédiaire de relais multiples afin d'atteindre la destination. Cette méthode s'avère efficace pour le codage d'autres types de données propres à des applications qui résident sur un site ou qui sont transmises sur un réseau [CST96]. Comme le chiffrement et le déchiffrement sont réalisés à chaque extrémité d'un trajet de communication, on peut parler alors d'une protection de bout en bout.

Les couches selon l'architecture OSI	**Protocoles d'Internet**			
7 Application	SMTP, POP, IMAP, HTTP, FTP	SET	**PKI**	**ISAKMP, IKE, OAKLEY, …**
6 Présentation	MIME, HTML, XML, Base64, … SMIME, Signature XML	X509, PKCS8, PKCS10, PKCS12,		
	CMS (Cryptographic Message Syntax)/PKCS7			
5 Session	**TLS**	**SSH**		
4 Transport	TCP	UDP		
3 Réseau	IP, ICMP, RIP, OSPF, **IPSEC**			
2 Liaison	PAP, CHAP, RADIUS			
	Encapsulation IP ou PPP			
1 Physique	Pratiquement tout support de transmission, en particulier les réseaux locaux, les réseaux téléphoniques (commuté ou liaison spécialisée)			

Tableau 2-2. Emplacement des protocoles par rapport au modèle de référence OSI.

9. La confiance entre les communicateurs

La confiance entre les tiers est une exigence fondamentale de toute implantation à grande échelle de services de sécurité reposant sur la cryptographie à clé publique.
Toutefois, dans un réseau de grande taille, il est impossible et irréaliste de s'attendre à ce que chaque utilisateur établisse au préalable des relations avec tous les autres utilisateurs. En outre, comme la clé publique d'un utilisateur doit être accessible à l'ensemble des autres utilisateurs, le lien entre une clé publique et une personne donnée doit être garanti par une tierce partie de confiance, afin que nul ne puisse se faire passer pour un utilisateur légitime.
Une tierce partie de confiance, dont les mécanismes sont sûrs, permet aux utilisateurs d'avoir implicitement confiance dans toute clé publique certifiée par cette tierce partie que nous appelons Infrastructure à Clé Publique (ICP) ou même PKI pour *Public Key Infrastructure* [PKIX-WG].
L'agent ICP qui certifie les clés publiques des utilisateurs est appelé Autorité de Certification (AC). Les ACs s'occupent de la diffusion de ces clés publiques (ainsi que l'identificateur de l'utilisateur) sous forme de « certificats » sur des annuaires électroniques publics (par exemple

des annuaires X.500) [MG-15a]. Le format de certificat le plus utilisé dans le cadre d'une ICP est le standard normalisé X.509v3 [PKIX-WG].
Malheureusement, la prolifération des « extensions » dans ces certificats a induit d'autres problèmes dans les domaines d'interopérabilité, de juridiction et de révocation.
Une solution apparaît appropriée est de partager un certificat X.509 en deux : un certificat d'identité qui va porter des informations sur l'identité et un certificat d'attribut qui va porter des informations sur les rôles attribués à cette identité. Cette solution simplifie énormément le processus d'émission des certificats et peut dans certaines situations éliminer le problème de la révocation.
Le certificat d'attributs X.509 [ISO-9594] est une solution orientée vers les services d'authentification (par exemple : le contrôle d'accès). Son point faible est la complexité à déployer les certificats d'attribut. Une part de cette complexité est attribuable à l'encodage des certificats X.509 en format ASN.1 (Abstract Syntax Notation) [X.208] et à l'intégration difficile de nouveaux attributs.
La figure 2.1 représente les relations qui peuvent lier un certificat d'identité à un certificat d'attribut. Les deux certificats peuvent être liés à travers plusieurs champs tels que la clé publique, le numéro de série ou même le champ de signature de l'autorité de certification [Wadj02] [ISO-9594].

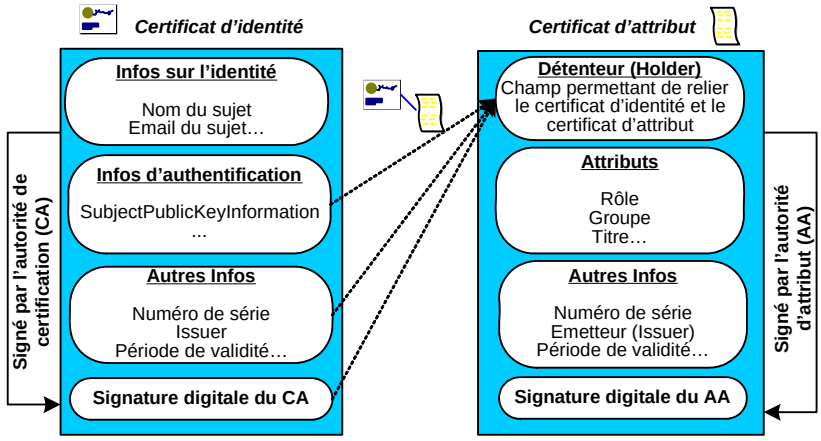


Figure 2-1. Relation entre certificat d'identité et certificat d'attribut

Dans le cadre du Projet RNRT-ICARE [ICARE], nous avons essayé de résoudre cette problématique en développant un nouveau type de certificat d'attribut encodé en XML [W3C-98a] (voir chapitre 4, figure 4-11). L'encodage du certificat en format XML amène à une grande souplesse au niveau du déploiement de ce type de certificat ainsi que son intégration dans les outils de signature, de contrôle d'accès et de délégation des droits [Deme04].

10. L'autorisation

Le service d'autorisation a comme rôle d'empêcher la divulgation non autorisée de l'information ou des données en contrôlant l'accès à celles-ci. Pour cela, il repose sur l'identification et l'authentification des utilisateurs, sans quoi elles sont inefficaces.
La plupart des protocoles de sécurité offrent une telle fonction, habituellement sous forme d'une liste de contrôle d'accès (LCA). Les listes LCA énumèrent les personnes, les groupes de personnes ou les processus qui sont autorisés à accéder à certains fichiers ou répertoires.
Pour cela et afin d'assurer la confidentialité de l'information, la mise en oeuvre des LCAs nécessite une gestion attentive et précise. Une méthode pour rendre ce service plus dynamique

serait d'utiliser des certificats de rôles ou ce qu'on appelle des certificats d'attribut. Ces deniers ne nécessitent pas une protection physique puisqu'ils sont générés pour une durée de vie très courte.

11.La gestion des clés

Pour être efficace, tout système de chiffrement doit reposer sur des méthodes fiables et robustes pour la gestion des clés. Une bonne gestion des clés comprend la production de clés de chiffrement qui ont des propriétés particulières (par exemple longueur, caractères aléatoires, etc.), la distribution des clés aux utilisateurs ou aux systèmes appropriés, la protection des clés contre leur divulgation, la modification et (ou) la substitution des clés, ainsi que leur distribution, ce qui peut comprendre l'archivage.
Lorsque l'on décide d'utiliser les technologies de chiffrement dans un réseau, il est crucial d'instaurer un système approprié de gestion des clés afin d'épauler le chiffrement. En effet, un système de chiffrement ne vaut rien si la sécurité des clés n'est pas assurée.
En plus, pour assurer la sécurité à long terme des transactions, les clés doivent être renégociées d'une manière totalement indépendante des anciennes clés utilisées. Ce service s'appelle le service de PFS (ou *Perfect Forward Secrecy*) [PFS] [Schn95]. Bien que ce service soit coûteux en terme d'opérations cryptographiques (la génération des nouveaux clés Diffie-Hellman [Diff76] prend 100 ms [Sher92]), il reste important pour assurer la sûreté cryptographique des clés. Ainsi, il peut être utilisé optionnellement par les entités communicantes.

12. La mise en œuvre d'un VPN

Le VPN [Kols02] est un réseau privé construit à l'intérieur d'un réseau public, comme Internet. Il permet donc de remplacer les traditionnels réseaux de lignes louées par de simples accès à l'Internet global avec le moindre coût.

Ces VPNs n'ont pas comme seul intérêt l'extension des WAN (Wide Area Network) à moindre coût mais aussi l'utilisation de services ou fonctions spécifiques assurant la qualité de service (QoS) et la sécurité des échanges. Les fonctionnalités de sécurité sont matures mais la réservation de bandes passantes pour les tunnels est encore un service en développement limité par le concept même d'Internet. La sécurité des échanges est assurée à plusieurs niveaux de la couche OSI et par différentes fonctions comme le cryptage des données, l'authentification des deux extrémités communicantes et le contrôle d'accès des utilisateurs aux ressources.

Les VPNs peuvent être créés suivant différentes formes, soit pour connecter deux réseaux locaux distants (connexion network-to-network), soit entre deux stations (hop-to-hop) soit entre une station et un réseau (hop-to-network). Ce dernier est généralement utilisé par des entreprises désireuses de créer des accès distants pour des télétravailleurs via Internet.

13. Le contrôle d'accès

La nomadicité des utilisateurs et la demande d'accès distant sécurisé vers les réseaux privés des entreprises ont poussé ces dernières à adapter des solutions de sécurité basées sur des points d'accès situés aux frontières des réseaux privés. Ces points d'accès sont également intéressants dans le sens où ils constituent un point unique où l'audit et la sécurité peuvent être imposés. Ils peuvent donner des résumés de trafic, des statistiques sur ce trafic, ou encore

toutes les connexions entre les deux réseaux. Dans ce domaine, nous citons les deux techniques suivantes :

a. Les Firewalls

Un Firewall (pare-feu) est un système ou un groupe de systèmes qui gère et représente une politique de contrôle d'accès définie par les administrateurs réseaux. Un Firewall se situe toujours sur le lien unique (ou en tous cas sur un lien de passage du trafic) qui relie un réseau privé (Intranet) au reste du monde (Internet) afin de pouvoir filtrer tout le trafic sortant et rentrant.

Généralement, les Firewalls sont configurés pour protéger contre les accès non authentifiés du réseau externe. Ils assurent, entre autres, une fonction de filtrage à différents niveaux de la couche OSI et empêchent les intrus de se logger sur des machines du réseau interne. Pour filtrer les paquets, on peut regarder les protocoles de niveau supérieur, ou bien regarder les données transportées ou vérifier qu'elles ne contiennent pas de virus, ou ne contiennent pas tel ou tel mot par exemple.
Voici un schéma illustrant une architecture classique de sécurité utilisant un Firewall :

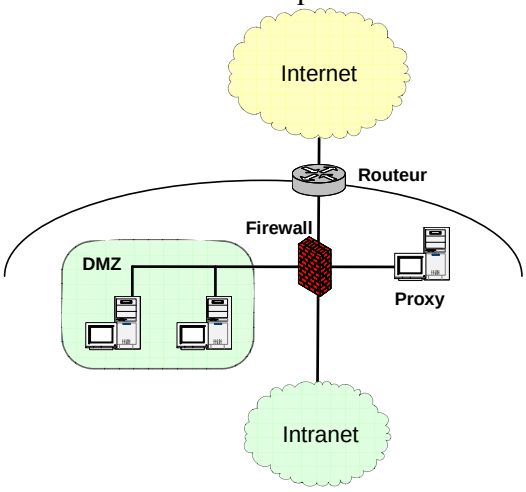


Figure 2-2. Architecture classique de sécurité utilisant un Firewall

Un Firewall peut être précédé d'un routeur particulier qui assure que les paquets entrants sont exclusivement à destination du Firewall. Un routeur peut également donner accès à un sous réseau dénommé DMZ (zone démilitarisée) dans laquelle réside les serveurs publics de l'entreprise. La DMZ est une zone protégée vis-à-vis de l'extérieure comme de l'intérieur.

Cependant, ce système de Firewall est insuffisant s'il n'est pas accompagné d'autres protections. En effet, il ne fournit pas les services de sécurité énoncés précédemment (authentification de la source des données, intégrité, confidentialité, etc.) [Alle00]. Plusieurs types d'attaques passives et actives ne sont pas protégeables (analyse de trafic, IP Spoofing, IP Flooding, etc.) ou encore les failles de configuration par des outils d'analyse automatique des vulnérabilités du système.

b. Les Proxy et les Reverse Proxy

Les Proxy permettent d'une part, de relayer le trafic entre Internet et un réseau protégé et d'autre part, ils effectuent un enregistrement intermédiaire de données à des points définis d'Internet. La meilleure utilisation de cette technique est apparue avec les serveurs Web. Avec

sa technique de cache, un *Proxy* permet de recevoir les pages Web plus rapidement car celles-ci sont directement activées depuis un *Proxy* proche de l'émetteur de la requête HTTP.

Le *Reverse Proxy* est un mécanisme de contrôle d'accès récemment apparu. Si le *Proxy* est un mécanisme de filtrage des connexions sortantes d'une entreprise, le *Reverse Proxy* est le mécanisme inverse qui permet de filtrer toutes les connexions des utilisateurs vers un ensemble d'applications. Un serveur *Reverse Proxy* permet la réalisation des VPNs au niveau applicatif tout en se basant sur un protocole de sécurité comme SSL/TLS ou SSH.

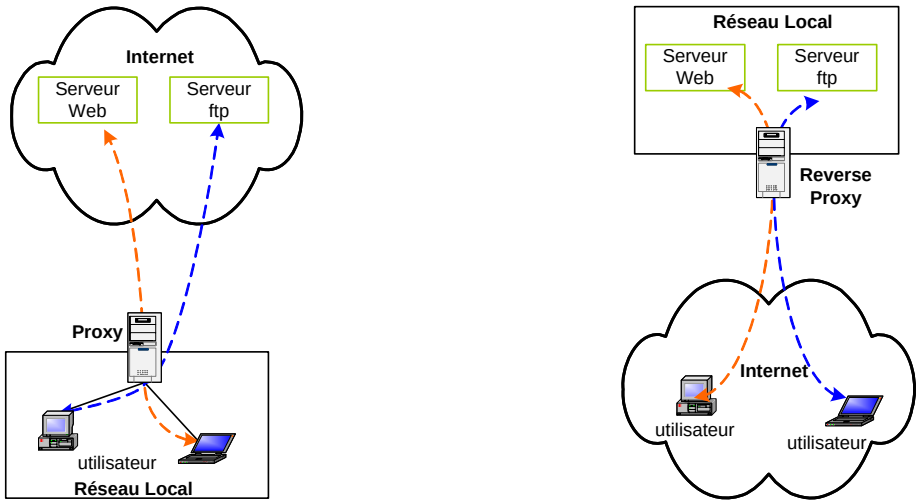


Figure 2-3. Architecture des Proxy et des Reverse Proxy

Actuellement les protocoles de sécurité sont plus adaptés à ces deux mécanismes largement déployés dans les réseaux des entreprises. Nous pouvons citer dans ce domaine l'utilisation conjointe du protocole IPSec avec les Firewalls et l'utilisation du protocole SSL/TLS avec les Reverse Proxy [Bert03]. Ils permettent ainsi de fournir des mécanismes plus robustes pour l'authentification et le contrôle d'accès des utilisateurs externes avant de pouvoir accéder au réseau interne de l'entreprise.

14. La gestion des «délégations» entre acteurs

Les points d'accès jouent le rôle d'un point d'authentification pour les utilisateurs qui se connectent au réseau interne des entreprises.

Les solutions de sécurité telles que SSL/TLS et SSH essaient de s'adapter à ce type de connexion en présentant les entités intermédiaires comme des entités de confiance et en faisant un double tunnel de sécurité pour assurer une sécurité de bout en bout des données.

Cependant, la problématique d'une telle approche vient du fait que dans des scénarios où une application finale A souhaite déléguer le rôle d'authentification à une autre entité B. Cette dernière doit présenter un certificat de délégation d'un rôle d'authentification émis par l'application A ou par une infrastructure de gestion des privilèges (ou PMI pour *Privilege Management Infrastructure*) [deme04].

Pour cela, nous proposons l'utilisation d'un mécanisme de délégation des droits basé sur les infrastructures de gestion des privilèges et les certificats d'attribut.

Par délégation, nous entendons dire l'autorisation donnée par une entité finale A (normalement le serveur destinataire) à une entité intermédiaire B pour pouvoir exercer un mécanisme d'authentification et de contrôle d'accès à sa place.

Un tel mécanisme permet d'offrir à l'utilisateur une preuve de l'identité et des rôles associés à l'entité intermédiaire B.
Notons ainsi que, dans le cas ou les passerelles sont éloignées des serveurs origines (cas de distribution de contenu dans Internet) [Dele04], la délégation du rôle d'authentification aux différentes passerelles peut jouer un rôle important pour la minimisation des bandes passantes vers le serveur Web d'application.

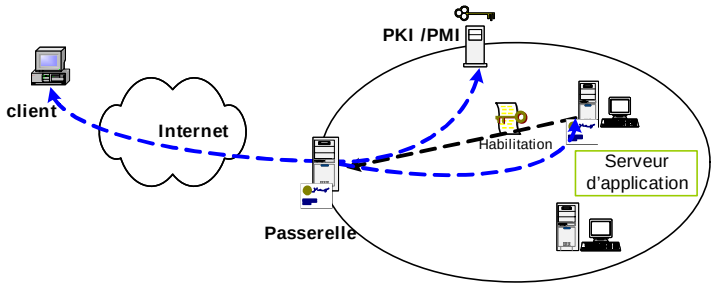


Figure 2-4. Exemple de délégation entre un serveur d'application et une passerelle

15. La vérification formelle des protocoles de sécurité

Au cours des dernières années, différents protocoles cryptographiques ont été mis en œuvre pour répondre aux exigences demandées, mais la difficulté de la conception de ces protocoles vient du fait que les messages échangés peuvent être écoutés par une tierce personne, interceptés ou modifiés.
Afin d'assurer que ces protocoles soient protégés contre des attaques pareilles, un nouveau domaine de sécurité est apparu. C'est la modélisation et la vérification des protocoles cryptographiques.
Depuis 1990, plusieurs techniques ont été présentées [Blan01] [Möd03] [Denk98]. Dans cette thèse, nous choisissons la méthode développée dans le projet RNTL EVA [EVA] afin de valider les différents protocoles et propositions étudiés dans cette thèse.

16. L'extensibilité du protocole

Un protocole est dit extensible s'il permet de fournir des nouveaux services sans avoir à changer sa structure de conception. Le grand intérêt de cette option a poussé les nouveaux successeurs des protocoles de sécurité (comme TLS Extensions) [RFC3546] et des protocoles d'échange des clés (comme IKEv2) [IKEv2] à ajouter des échanges ou des messages *génériques* dans leur phase d'initialisation. Ces messages permettent à ces protocoles de négocier durant leurs phases d'initialisation, des nouveaux services tels que l'URL des certificats et les HMAC tronqués.
Ces messages sont dans la plupart des cas, déclanchés du coté de l'émetteur qui propose la négociation d'un ou de plusieurs services optionnels. Ceci permet de garder une interopérabilité entre les entités qui supportent ces services et celles qui ne les supportent pas.

2.4 Conclusion

Dans ce chapitre, nous avons défini les principales exigences qu'un protocole de sécurité doit assurer. Ces exigences sont nécessaires pour de nombreuses applications telles que le commerce électronique, l'accès distant à une machine ou à certaines parties d'un site contenant des informations confidentielles.

En se basant sur ces exigences, nous allons comparer dans le chapitre suivant, les trois principales solutions de sécurité suivantes : IPSec, SSH et SSL/TLS. Nous proposons par la suite, d'étendre ces solutions avec de nouveaux services tels que la non répudiation et la protection d'identité. Notre analyse critique va nous permettre d'identifier des besoins réels non couverts par ces solutions existantes. Pour cela, nous proposons de couvrir ces exigences par un nouveau protocole nommé SEP (Secure and Extensible Protocol).

Chapitre 3

3 Analyse des solutions de sécurité existantes

3.1 Résumé

Dans ce chapitre, nous proposons une étude et une analyse critique des trois solutions de sécurités les plus déployées actuellement, à savoir le protocole SSH, le protocole SSL/TLS et le protocole IPsec. Nous détaillons également les protocoles d'authentification et de négociations des associations de sécurité dans IPSec ainsi que leurs successeurs.
Enfin, pour mieux situer ces protocoles, nous les comparons par rapport aux exigences définies dans le chapitre précédent. Ce travail a donné lieu à deux publications : [Hajj03b] et [Hajj03d].

3.2 Introduction

L'utilisation de l'IP (Internet Protocol) comme base des réseaux informatiques est devenue très importante, que ce soit par l'utilisation croissante d'Internet ou dans le cadre des réseaux d'entreprises de type Intranet.
Si la flexibilité d'IP et sa simplicité ont su répondre aux besoins, en matière de réseaux informatiques de ces dernières années, le but de ce protocole n'a jamais été d'assurer des communications sécurisées.
L'utilisation des réseaux IP pour des applications sensibles (commerce électronique, transactions bancaires, etc.) ont donc poussé au développement de diverses solutions de sécurité : Firewall, routeurs filtrants, protocoles et applications sécurisés se sont multipliés.

Devant les besoins grandissants, la solution était donc d'intégrer des couches de sécurité afin de pouvoir protéger les communications. Le réseau IPv4 [RFC791] est largement déployé et la migration complète vers IPv6 [RFC2460] (qui sera sécurisé par défaut) nécessite du temps. En effet, il est vite apparu intéressant de définir des mécanismes de sécurité applicables à IPv4. Les mécanismes sont couramment désignés par les termes SSH (*Socket Secure Shell*), SSL (*Secure Socket Layer*), IPSec (*IP Security Protocol*) ainsi que des mécanismes de gestion des clés tels que ISAKMP (*Internet Security Association and Key Mangement Protocol*) [RFC2408] et IKE (*Internet Key Exchange*) [RFC2409].

Ce chapitre s'attache à traiter les trois solutions de sécurité les plus répandues actuellement, tout en expliquant les avantages et les limites de chacune de ces solutions.

3.3 Le protocole SSH (Secure Shell)

3.3.1 Introduction

SSH, le Shell sécurisé, est développé en 1995 par *Tatun Ylönen*, un professeur finlandais. La première version de ce protocole, SSH-1 [Ylon95], avait pour but de sécuriser les communications distantes au serveur Unix, surtout les commandes à distances (rsh, rlogin, …).
À cause de plusieurs failles de sécurité dans la première version de ce protocole [Dbar02], l'IETF a mis en place un groupe de travail appelé SECSH (Secure SHell) pour normaliser le protocole et orienter son développement dans l'intérêt du public. Ce groupe a soumis un ensemble de Drafts détaillant une nouvelle version décisive du protocole SSH 2.0 (ou SSH-2) [SSH-Arch] [SSH-Auth] [SSH-Trans] [SSH-Conn]. Cette version contient de nouveaux algorithmes et services comme le transfert des fichiers (SFTP) et le « tunneling » ou « port forwarding» des protocoles (voir § 3.3.3.2). Dans la suite, nous décrivons seulement la version 2 de ce protocole.

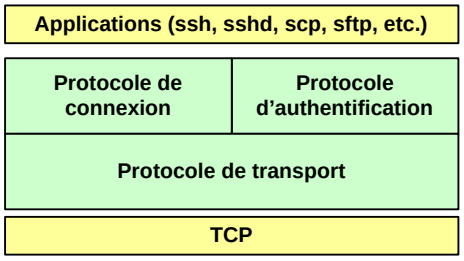


Figure 3-1. Architecture de SSH-2

3.3.2 Architecture

SSH utilise une architecture client serveur pour assurer l'authentification, le chiffrement et l'intégrité des données transmises dans un réseau. La version 2 de ce protocole spécifie une architecture séparée en trois sous protocoles (ou couches) travaillant ensemble (figure 3-1) :

- La couche transport SSH (SSH-TRANS)
  Elle fournit l'authentification du serveur, la confidentialité et l'intégrité des données. Cette couche doit être créée pour que le client sache qu'il communique bien avec le bon serveur. Ensuite, la communication est chiffrée entre le client et le serveur au moyen d'un chiffrement symétrique.

- La couche d'authentification SSH (SSH-AUTH)
  Elle permet de certifier l'identité du client auprès du serveur. Cette couche est sécurisée par la clé de chiffrement créée dans la partie précédente. Après l'authentification du client auprès du serveur, de nombreux services différents peuvent être utilisés de façon sécurisée au cours de la connexion, tels qu'une session Shell interactive, des applications X11 et des ports TCP/IP [RFC761] tunnellisés.

- La couche de connexion SSH (SSH-CONN)
  Cette couche s'appuie sur la couche d'authentification. Elle offre une variété de services riches aux clients, en se servant de l'unique tube fourni par SSH-TRANS. Ces services comprennent tout ce qu'il faut pour gérer plusieurs sessions interactives ou non :

multiplexage de plusieurs flux (ou canaux), gestion des transferts X, de port et d'agent, etc.

Même si SSH, dans sa version 2, est devenu plus modulaire, nous pouvons remarquer que les fonctionnalités de chaque sous protocole dépendent fortement des autres sous protocoles. Par exemple, le service d'authentification entre deux entités est divisé entre la couche d'authentification et la couche de transport.

Couche	Description
Transport	Authentification du serveur, négociation de l'algorithme, échange de la clé de session, intégrité des données, identification de session
Authentification	Authentification du client (clé publique, mot de passe, basée sur l'hôte), changement du mot de passe.
Connexion	Transfert de port TCP et transfert X, Transfert d'agent d'authentification Gestion des sessions interactives, Exécution de programmes distants Contrôle de flux, Gestion des terminaux (modes et tailles des fenêtres) Compression des données

Tableau 3-1. Familles des couches SSH-2

Dans ce qui suit, nous détaillons la phase d'initialisation ainsi que les trois composantes essentielles de SSH.

3.3.2.1 La phase d'initialisation

Dans cette section, nous présentons sous forme d'étapes, la phase d'initialisation du protocole SSH (figure 3-2).

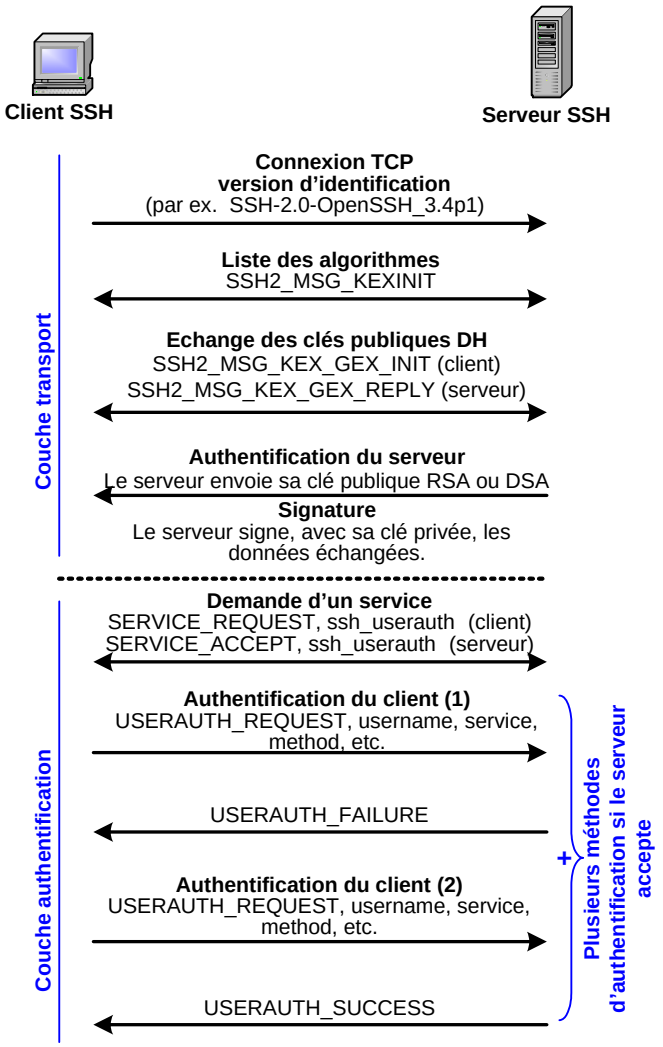


Figure 3-2. Le Handshake de SSH

Dans la phase d'initialisation du protocole SSH, de nombreux éléments importants sont négociés. Dans le cas d'utilisation du protocole TCP/IP sur le réseau, la procédure suivante se déroule entre la machine cliente et la machine serveur :

1. Dès que la connexion est établie (protocole TCP sur le port 21 du serveur), le client et le serveur échangent en clair leurs numéros de version du protocole SSH. Sitôt ce premier échange est effectué, le client et le serveur utilisent un protocole binaire par paquet.
2. Le serveur commence par envoyer au client la liste des méthodes de cryptage supportées, la liste des méthodes d'authentification supportées, des indicateurs d'extensions de protocole (par exemple, la méthode de compression, etc.) et un cookie sur 64 bit que le client devra renvoyer. Ce cookie a comme but de protéger le serveur contre une attaque par déni de service.
3. Le client envoie à son tour la liste des méthodes de cryptage supportées, la liste des méthodes d'authentification supportées et une copie du cookie du serveur.
4. Le client et le serveur choisissent les meilleurs algorithmes supportés par les deux.
5. Le client et le serveur calculent séparément un identifiant de session à partir des valeurs Diffie-Hellman échangées entre les deux entités. SSHv2 utilise aussi une

méthode d'échange des groupes DH (par exemple, *diffie-hellman-group1-sha1*) pour simplifier l'échange DH.

6. Le serveur envoie sa clé publique au client et signe avec sa clé privée les valeurs échangées précédemment.
7. Le client vérifie la signature du client et passe ensuite en mode crypté.
8. Le serveur répond au client par un message de confirmation crypté.
9. Les deux parties, le client et le serveur, sont maintenant en mode crypté avec utilisation de l'algorithme et de la clé sélectionnés.
10. Le client envoie maintenant la demande d'un service (par exemple, ssh-userauth ou ssh-connection).
11. Le serveur précise les méthodes d'authentification qu'il peut accepter (publickey, password, etc.).
12. Finalement, le client envoie sa méthode d'authentification que le serveur accepte ou rejette. Dans la plupart des implémentations SSH et suivant la politique du serveur, le client a le droit à trois essais pour s'authentifier.

3.3.2.2 Couche transport (SSH-TRANS)

Le rôle principal de la couche transport est d'établir une communication sécurisée entre deux ordinateurs hôtes au moment de l'authentification et par la suite également.
Elle utilise généralement le protocole TCP/IP et accomplit sa tâche en s'occupant du chiffrement et du déchiffrement des données et en offrant la protection nécessaire aux paquets de données lors de leur envoi et de leur réception. En outre, la couche transport peut également faire la compression des données pour accélérer la vitesse de transfert de l'information.
La couche transport est une partie de la phase d'initialisation du protocole SSH décrit précédemment. En effet, lorsqu'un client communique avec un serveur au moyen d'un protocole SSH, de nombreux éléments importants sont négociés afin que les deux systèmes puissent créer correctement une session sécurisée.

Durant l'échange des clés, le serveur s'identifie au client au moyen d'une clé publique RSA ou DSA. Puisque dans la spécification du protocole SSH, ce dernier ne supporte pas la distribution des clés par un tiers de confiance (cas des certificats X.509), la clé du serveur est inconnue pour le client durant sa première communication. SSH propose de contourner ce problème en permettant au client d'accepter la clé du serveur lors de leur première connexion SSH. Ensuite, lors des connexions suivantes, la clé du serveur peut être vérifiée au moyen d'une version enregistrée au niveau du client, ce qui permet au client de s'assurer qu'il communique bien avec le serveur désiré. Ce mécanisme est un des inconvénients majeurs de ce protocole qui le rend vulnérable à des attaques de type homme du milieu.
A partir des valeurs publiques DH échangées, les deux entités génèrent automatiquement une clé session utilisée avec des clés dérivées pour le chiffrement des données.
Une fois une certaine quantité de données est transmise au moyen d'une clé et d'un algorithme précis (la quantité exacte dépend de la mise en application du protocole SSH), il y un nouveau échange de clés. Ce qui produit un autre ensemble de valeurs repères et une autre valeur secrète partagée. Ainsi, le protocole SSH permet d'assurer le service de PFS [Schn95].

3.3.2.3 Couche authentification (SSH-AUTH)

Une fois que la couche transport a créé un tunnel sécurisé pour envoyer les informations entre les deux systèmes, le serveur indique au client les différentes méthodes d'authentification

prises en charge, telles que l'utilisation d'une signature chiffrée privée ou l'entrée d'un mot de passe. Le client doit ensuite essayer de s'authentifier au serveur au moyen d'une des méthodes spécifiées.
Etant donné que les serveurs peuvent être configurés de façon à permettre différents types d'authentification, cette méthode donne aux deux parties un niveau de contrôle optimal. Le serveur peut décider quelles méthodes d'authentification prendre en charge en fonction de son modèle de sécurité et le client peut choisir l'ordre des méthodes d'authentification à utiliser parmi celles qui sont disponibles. Grâce à la nature sécurisée de la couche authentification SSH, même les méthodes d'authentification qui, de prime abord, semblent non sécurisées telles que l'authentification d'ordinateur hôte, peuvent être utilisées en toute sécurité.
La plupart des utilisateurs exigeant un Shell sécurisé procèdent à l'authentification au moyen d'un mot de passe. Comme ce mot de passe est complètement chiffré, il peut être envoyé sans problème sur n'importe quel réseau.

3.3.2.4 Couche connexion (SSH-CONN)

Après avoir effectué avec succès l'authentification au moyen de la couche transport de SSH et de la couche d'authentification, des canaux multiples sont ouverts en transformant la connexion simple entre les deux systèmes en connexion multiplex[2]. Chaque canal peut ainsi s'occuper de la communication d'une session de terminal différent, du transfert d'informations X11 ou de tout autre service séparé essayant d'utiliser la connexion SSH.
Le client et le serveur peuvent tous deux créer un nouveau canal et chaque canal reçoit un numéro différent aux deux extrémités de la connexion. Lorsque le client essaye d'ouvrir un nouveau canal, les clients envoient le numéro du canal accompagné de la requête. Cette information est stockée par le serveur et utilisée pour adresser la communication à ce canal.

Ainsi, les types différents de session ne peuvent pas se nuire et lorsqu'une session se termine, son canal peut être fermé sans que la connexion SSH primaire ne s'interrompe.
Les canaux prennent aussi en charge le contrôle du flux de données, ce qui leur permet d'envoyer et de recevoir des données de façon ordonnée. Ce faisant, aucune donnée n'est envoyée par le canal tant que l'hôte n'a pas reçu un message lui indiquant que le canal est ouvert.
Le client et le serveur négocient automatiquement la configuration de chaque canal, selon le type de service demandé par le client et le mode de connexion de l'utilisateur au réseau. Ceci permet de gérer facilement les différents types de connexions distantes sans devoir changer l'infrastructure de base du protocole.

[2] Une connexion multiplex envoie plusieurs signaux sur un support commun et partagé. Avec le protocole SSH, divers canaux sont envoyés sur une connexion sécurisée commune.

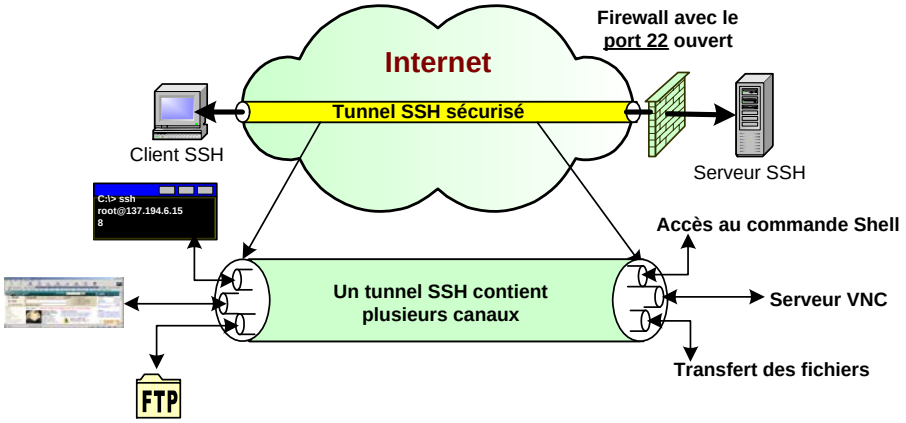


Figure 3-3. Tunnels et canaux SSH

3.3.3 Les avantages

SSH constitue une approche puissante et pratique pour protéger les communications sur un réseau d'ordinateurs. A travers son mécanisme d'authentification, SSH permet d'effectuer sous un tunnel sécurisé des connexions à distance, des transferts de fichiers, le transfert de ports TCP/IP et d'autres fonctionnalités importantes. Ainsi, il présente comme avantages :

3.3.3.1 La création d'un VPN au niveau d'échange

SSH est un simple programme applicatif qui peut établir des canaux sécurisés dans un réseau ouvert et assure entre autres des VPNs beaucoup plus faciles à déployer que IPSec. Ce dernier nécessite des ajouts au système d'exploitation des machines situées aux deux extrémités, si elle ne dispose pas déjà et, aux équipements réseau comme les routeurs. Par ailleurs, SSH utilise la technique de SOCKS [RFC1928] intégrée dans la plupart des équipements réseaux.

3.3.3.2 La notion du Transfert (tunneling)

Le transfert, ou tunneling, consiste à encapsuler un autre service TCP/IP comme Telnet ou IMAP, dans une session SSH afin de lui apporter les bénéfices de la sécurité de SSH (confidentialité, intégrité, authentification, autorisation). En transférant Telnet par exemple, via SSH, toutes les données seront chiffrées et leur intégrité sera contrôlée. En plus l'authentification des clients sera assurée d'une manière sécurisée par SSH. SSH reconnaît trois types de transfert : le transfert de port TCP, le transfert des sessions interactives de type X-Window et le transfert des agents SSH qui permet aussi d'utiliser des clés privées SSH sur des machines distantes.

3.3.3.3 Le service de Single Sign On (SSO)

La méthode de Single Sign-On [Rizc00] [Dbar02] des clients SSH est basée sur les agents SSH. Un agent SSH est un programme qui garde les clés privées en mémoire et qui fournit les services d'authentification aux clients SSH. Cette méthode permet au client d'introduire son mot de passe lors de la première connexion à un serveur SSH. L'ouverture d'une session sécurisée avec un nouveau serveur SSH passe de manière transparente à l'utilisateur. Ainsi le client ne retape pas son mot de passe une deuxième fois puisque les serveurs vont communiquer directement avec l'agent du client.

Avec les passerelles, on utilise la notion des agents mobiles pour se connecter au serveur interne. La passerelle (un serveur SSH) se déguise comme un deuxième agent SSH qui communique avec l'agent du client à travers SSH. Une fois le client active l'option de

transfert d'agent et veut accéder à un service interne dans le réseau de passerelle, le serveur SSH contacte le service SSH en utilisant l'agent du client stocké chez lui. En coulisse, le serveur SSH communique avec l'agent du client SSH qui construit un authentificateur prouvant l'identité du client et qui le repasse au serveur. Le serveur SSH vérifie l'identité du client et l'authentification réussit avec le service SSH [Dbar02] (figure 3-4).

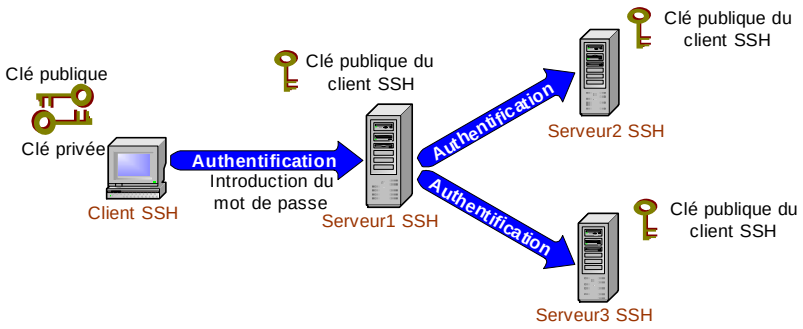


Figure 3-4. Authentification avec un agent SSH

3.3.4 Les inconvénients

SSH est capable de contourner de nombreuses menaces de sécurité liées au réseau. Cependant, il est vulnérable aux attaques par déni de service, héritant ainsi les faiblesses de TCP/IP sur lequel il repose. En outre et suivant l'environnement, SSH est sensible à certaines méthodes d'attaques, comme l'analyse et le détournement de trafic. Dans ce qui suit, nous détaillons les principaux inconvénients de SSH.

3.3.4.1 La première authentification non sécurisée du client avec le serveur SSH

Puisque SSH n'utilise pas un tiers de confiance pour distribuer les clés publiques, la première fois qu'un client SSH se connecte à une nouvelle machine distante, il effectue un travail supplémentaire en vérifiant lui-même la clé publique du serveur, soit en comparant le hachage, soit en contactant l'administrateur du serveur distant.
Une fausse acceptation de la clé publique envoyée par le serveur SSH peut laisser un malveillant détourner le trafic SSH en envoyant une fausse clé lors de la première connexion au serveur SSH.

3.3.4.2 Attaques sur le mot de passe

Comme dans Telnet [RFC854], les caractères formant le mot de passe des clients SSH sont envoyés directement dans des paquets IP séparés. Un malveillant qui observe ces paquets peut conclure non seulement le nombre de caractères formant le mot de passe mais il peut aussi, à travers des méthodes markoviennes basées sur le temps et le délai nécessaire par le client pour introduire son mot de passe, d'estimer les différents caractères du mot de passe [Xiao01].

3.3.4.3 Authentification non sûre par « hôte de confiance »

Par définition, l'authentification par hôte de confiance renvoie aux méthodes d'authentification Rhosts, RhostsRSA de SSH-1 et hôte de SSH-2 [Dbar02]. Plutôt que de demander au client de prouver son identité à tous les hôtes qu'il visite, l'authentification par hôte de confiance établit des relations de confiance entre les machines. Même si l'authentification par hôte de confiance est pratique pour les utilisateurs et les administrateurs (car elle permet de mettre en place une authentification automatique entre des hôtes), cette

technique dépend lourdement d'une administration correcte et de la sécurité des hôtes concernés. En effet, si un seul de ces hôtes est compromis, un attaquant pourra accéder automatiquement à tous les comptes des autres hôtes.

3.3.4.4 SSH et la traduction des adresses réseau (NAT)

Un autre problème de SSH est lié à des applications interactives qui traversent des réseaux masqués par une passerelle NAT [RFC2663].
Nous rappelons que, la traduction des adresses réseau ou NAT (Network adresse translation), consiste à connecter deux réseaux par une passerelle réécrivant les adresses source et destination des paquets lorsqu'ils la traversent. L'avantage de NAT est qu'il permet de partager un nombre limité d'adresses Internet routables entre un grand nombre de machines utilisant des adresses privées.
Avec SSH, il est impossible de se connecter sous un tunnel SSH sécurisé vers des serveurs FTP [RFC959] situés derrière une passerelle NAT. En effet, le mode passif de FTP ne fonctionne pas car le serveur FTP fournit son adresse interne au client et cette adresse ne peut être atteinte par le client. En plus, l'utilisation du mode actif n'est pas simple puisque derrière le NAT, on trouve des machines clients où la configuration des NAT et Firewall laisse les connexions en mode passif seulement. Une solution à ce problème est que la passerelle NAT réécrit automatiquement le trafic du contrôle FTP en transit afin de prendre en compte la traduction d'adresses. Cette solution est impossible avec SSH qui protège en intégrité et en confidentialité les données et empêche par la suite toute modification intermédiaire (par la passerelle NAT ou autre).

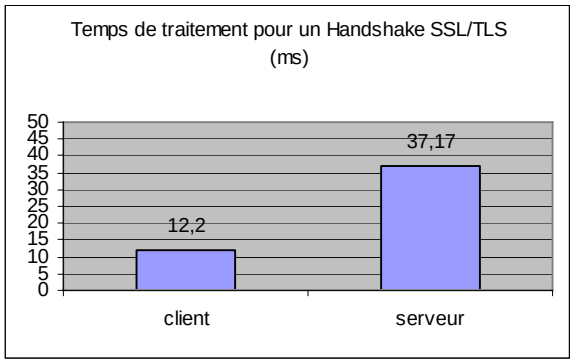
3.4 Le protocole SSL/TLS

3.4.1 Introduction

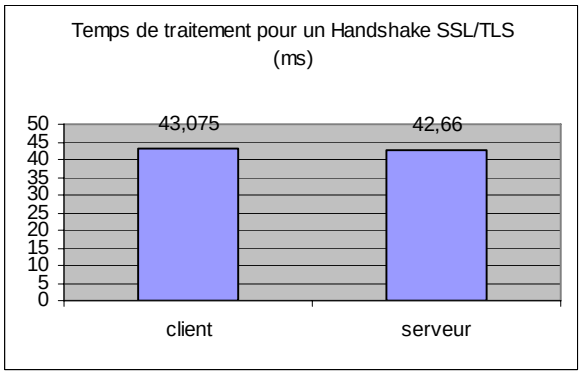
Le protocole SSL (Secure Socket Layer) est développé à l'origine en 1996 par Netscape [SSL3]. En 1999, la première version standard de ce protocole est apparue à l'IETF sous le nom de TLS (Transport Layer Security). TLS[3] correspond à la version 3.1 [RFC2246] de SSL.
SSL/TLS est un protocole modulaire dont le but est de sécuriser les transactions Internet entre le client et le serveur indépendamment de tout type d'application. En effet, SSL/TLS agit comme une couche supplémentaire au-dessus de TCP, permettant ainsi d'assurer les services d'authentification, de confidentialité et d'intégrité.
L'implémentation native de la dernière version de SSL/TLS (version 3) par les navigateurs et serveurs Web actuels du marché, fait de HTTPS (HTTP [RFC1945] sur SSL/TLS) [RFC2818] [RFC2817] le standard de facto de sécurisation des applications Web. C'est principalement pour cette raison que SSL/TLS est aujourd'hui massivement utilisé dans le commerce électronique afin de chiffrer les échanges et authentifier les serveurs via l'utilisation de certificats X.509v3. En revanche, l'authentification des clients reste beaucoup moins répandue. Ceci est dû au problème du déploiement à grande échelle des PKI, les lourdes opérations cryptographiques pour la vérification des certificats ainsi que la chaîne de certification. La figure suivante (figure 3-5) montre le temps de traitement cryptographique nécessaire pour l'établissement d'une session SSL/TLS avec une longueur de chaîne de certification égale à 1 [Resc02] [Shc02]. Nous ajoutons à cela le fait que SSL/TLS est utilisé

[3] Dans la suite de cette thèse, nous dénotons par SSL/TLS, le protocole TLSv1.0

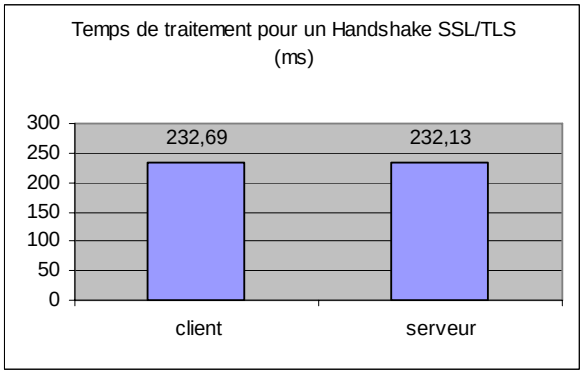
surtout dans des scénarios qui ne demandent pas une lourde vérification de l'identité des clients (avec SSL/TLS, les clients sont authentifiés au niveau applicatif).



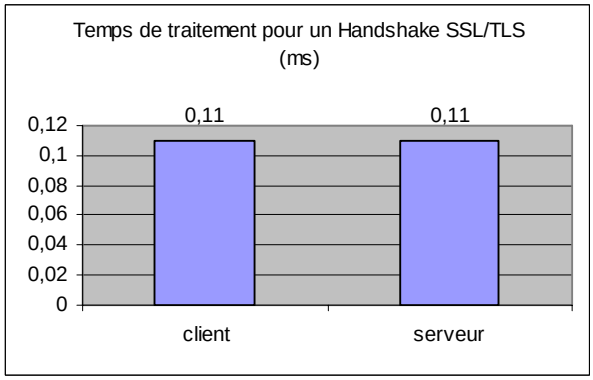
a) Mode RSA sans authentification du client



b) Mode RSA avec authentification du client



c) Mode DH éphémère avec authentification du client



d) Echange SSL /TLS abrégé

Figure 3-5. Temps de traitement pour un Handshake SSL/TLS[4]

3.4.2 Architecture

SSL/TLS est conçu pour se servir de TCP afin de fournir un service sécurisé de bout en bout fiable. SSL/TLS n'est pas un simple protocole, mais plutôt constitué de deux couches de protocole, comme illustré dans la figure 3-6.

- La couche d'enregistrement (*record protocol*) : la couche (ou le protocole) *Record* fournit des services de sécurité de base à divers protocoles de couche supérieure. Ce protocole sera décrit dans la section 3.3.5.

La deuxième couche est composée de deux sous protocoles :

- Le protocole de poignée de main (*Handshake Protocol*) : ce protocole permet d'authentifier les parties en jeu et d'assurer un mécanisme sécurisé pour la gestion des clés.

- Le protocole de changement des spécifications de chiffrement (*Change Cipher Spec protocol, CCS*) : ce protocole comprend un seul et unique message. Son but est d'activer pour la session SSL courante les algorithmes, les clés et les nombres aléatoires négociés

[4] Les temps de traitement cryptographique sont pris sous une machine Pentium avec un processeur 300MHZ, possédant comme système d'éxploitations FreeBSD et utilisant l'API de OpenSSL, version 0.9.4.

durant la phase d'initialisation (la phase *Handshake*). Ceci en passant ces informations au protocole *Record*.

- Le protocole d'alerte (*Alert Protocol*) : ce protocole spécifie les messages d'erreur envoyés entre les clients et les serveurs. Les messages sont de deux types : le premier contient les messages d'erreurs fatales (*Fatal Error*) et le deuxième, les alertes ou les messages d'erreur non fatale (*Warning Error*). Si le niveau est fatal, la connexion est abandonnée. Les autres connexions sur la même session ne sont pas coupées mais on ne peut pas en établir de nouvelles.

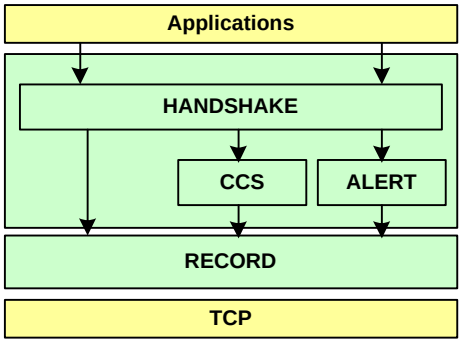


Figure 3-6. Architecture de SSL/TLS

Dans la suite, nous décrivons essentiellement la phase d'initialisation du protocole SSL/TLS (le protocole *Handshake)* et le protocole *Record*. Nous décrivons également le Handshake du RFC 3546 qui propose une extension générique dans le premier échange de SSL/TLS.

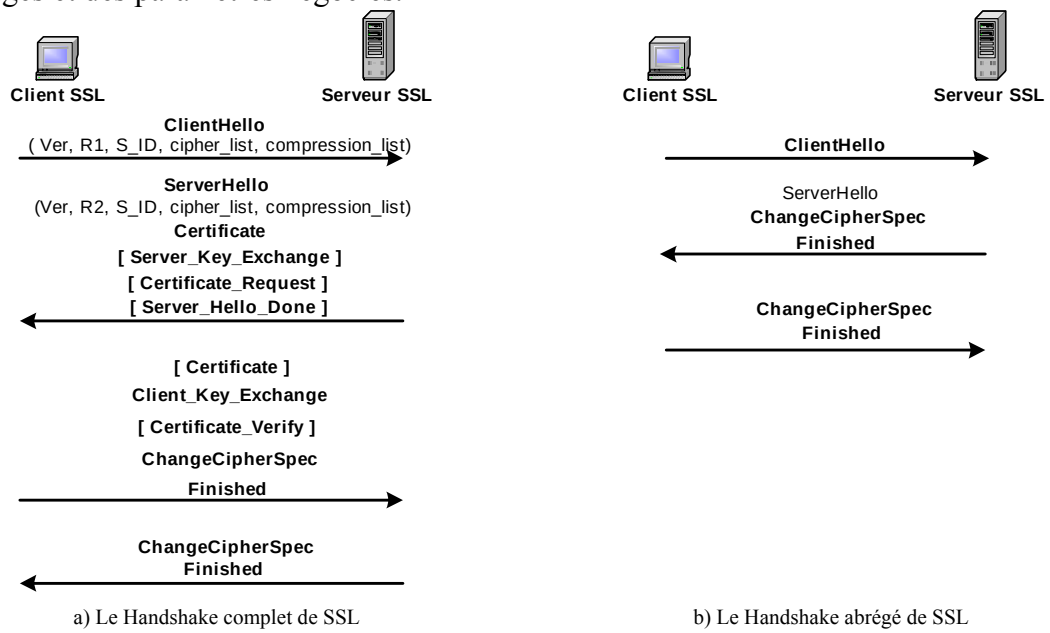
3.4.3 Le protocole Handshake

Le protocole *Handshake* est la partie la plus complexe de SSL/TLS. Il permet l'échange des paramètres de sécurité (nombres aléatoires, liste des algorithmes de chiffrement, de hachage, etc.) entre le client et le serveur avant que les données applicatives ne soient transmises. Il permet aussi l'authentification des deux communicateurs. Cependant, dans la plupart des cas, le serveur est uniquement authentifié.
Ce protocole peut opérer en deux formes : soit il établit un échange complet avec la négociation des paramètres de sécurité (le Handshake complet), soit il essaye d'utiliser une ancienne session SSL/TLS déjà négociée (Handshake abrégé).

3.4.3.1 Le Handshake Complet

La figure (3-7 a) illustre l'échange initial nécessaire pour établir une connexion complète entre le client et le serveur.
Le client commence l'échange SSL/TLS en envoyant le message *ClientHello* qui contient un nombre aléatoire (*R1*), un identificateur de session (*S_ID*), une liste des algorithmes de compression (*compression_list*) et une suite de chiffrement (*cipher_list*). Notons que le nombre aléatoire est une concaténation entre le temps système en format Unix et un nombre aléatoire.
Le serveur répond en envoyant le message *ServerHello* contenant un nombre aléatoire (*R2*), un identificateur non nul de session et une suite choisie des algorithmes de chiffrement et de hachage. Le serveur envoie également un certificat X.509 contenant sa clé publique pour s'authentifier. Ensuite, le client vérifie la clé publique du serveur et génère un nombre

aléatoire sur 48 octets connu sous le nom du *pre_master_secret*. Le client chiffre le *pre_master_secret* avec la clé publique du serveur et l'envoie dans le message *ClientKeyExchange.*
A la réception de ces messages, le serveur déchiffre le *pre_master_secret* en utilisant sa clé privée. C'est à ce moment là que le client et le serveur peuvent calculer un secret partagé, le *master_secret,* à partir des nombres aléatoires R1 et R2 et du *pre_master_secret*. Ce secret servira par la suite à la dérivation des clés de chiffrement et de déchiffrement dans les connexions SSL/TLS. Le dernier échange achève l'instauration d'une connexion sûre. Les deux entités échangent les messages *finished* contenant le hachage de l'ensemble des messages et des paramètres négociés.



a) Le Handshake complet de SSL b) Le Handshake abrégé de SSL

Figure 3-7 Le Handshake de SSL/TLS[5]

Si le certificat client est exigé par le serveur (le serveur envoie le message *CertificateRequest*), alors le client répond en envoyant le message *CertificateVerify* contenant sa signature sur toutes les informations échangées avec le serveur.
Même si l'authentification par certificat X.509 et par clés RSA reste la plus utilisée pour l'authentification des serveurs SSL/TLS, ces derniers peuvent avoir d'autres méthodes d'authentification telles que l'authentification par des valeurs *DH Anonymes* (un nombre premier, un nombre qui lui est relativement premier et la clé publique DH du serveur), des valeurs *DH temporaires* (les paramètres DH et une signature) [Diff76], un *échange RSA* (où le serveur n'a qu'une clé RSA pour signer et établir une clé RSA temporaire) ou finalement par *Fortezza* (norme de sécurité du gouvernement américain) [FRZ02]. Parmi toutes ces méthodes, l'échange du DH temporaire reste le plus sûr. De plus, il assure le service de PFS sur les données échangées. Toutes ces valeurs sont envoyées à travers les deux messages *Client_Key_Exhange* et *Server_Key_Exchange* envoyés respectivement avant les messages *ChangeCipherSpec* et *Server_Hello_Done*.

3.4.3.2 Le Handshake Abrégé

Puisque le Handshake de SSL/TLS est cryptographiquement coûteux en terme d'opérations de chiffrement asymétrique, SSL/TLS introduit un mécanisme plus rapide (figure 3-5 d) pour

[5] [M] signifie que le message M est optionnel

la négociation d'une nouvelle session basée sur un secret partagé entre les communicateurs (figure 3-7 b). Ce mécanisme est appelé « *Resumed Handshake* ».
Le client et le serveur font une négociation rapide puis calculent les nouvelles clés de sessions à partir de l'ancienne clé partagée. Le « Resumed Handshake » permet de générer des nouvelles clés symétriques et des vecteurs d'initialisation à partir du *master_secret* généré dans une ancienne session SSL/TLS.
Dans ce but, le client envoie dans le message *client_hello* le champ *S_ID* d'une ancienne session. Le serveur peut accepter la nouvelle connexion en plaçant le champ *S_ID* dans son message *server_hello* avant de l'envoyer au client. A ce point, le reste de l'échange SSL/TLS est sauté et les deux communicants calculent les nouvelles clés à partir de l'ancien *master_secret* créé dans l'ancienne session négociée.
Avec le Handshake actuel de SSL/TLS, le serveur ne peut pas savoir le service demandé par le client jusqu'à ce que la phase d'initialisation soit complètement établie. L'objectif de TLS Extensions est de permettre aux deux entités, client et serveur, de négocier de nouveaux services pendant le premier échange SSL/TLS et avant l'établissement d'une session sécurisée.

3.4.4 Le protocole Handshake de TLS Extensions

Le standard *TLS Extensions* [RFC3546] propose un cadre générique permettant d'étendre le protocole TLS avec des fonctionnalités et des services nouveaux. Le RFC 3546 définit deux messages génériques appelés *ExtendedClientHello* et *ExtendedServerHello* (tableau 3-2). Ces deux messages permettent au client et au serveur SSL/TLS de négocier de nouveaux services proposés dans le premier échange SSL/TLS et avant l'établissement d'une session sécurisée. Ceci permet de résoudre tous les problèmes de comptabilité entre les clients SSL/TLS qui veulent négocier des nouveaux services (tels que le HMAC tronqué, l'URL des certificats, la non répudiation, etc.) et les serveurs qui ne supportent pas ces services et vice-versa.
Avec le Handshake de TLS Extensions (figure 3-8), le client envoie le message *ExtendedClientHello* contenant le message standard *ClientHello* et une liste proposée de nouvelles fonctionnalités (*Extension_list*). Si le serveur accepte les nouvelles fonctionnalités proposées, il répond avec le message *ExtendedServerHello* contenant le même champ envoyé par le client.
Pour garder toute interopérabilité entre le standard TLSv1.0 et TLS Extensions, le message *ExtendedServerHello* devrait être envoyé en réponse à un message de type *ExtendedClientHello* envoyé par le client. Notons que toute erreur dans le format de message produira une erreur SSL/TLS fatale [RFC3546]. Le reste de cet échange est semblable au Handshake du protocole SSL/TLS.

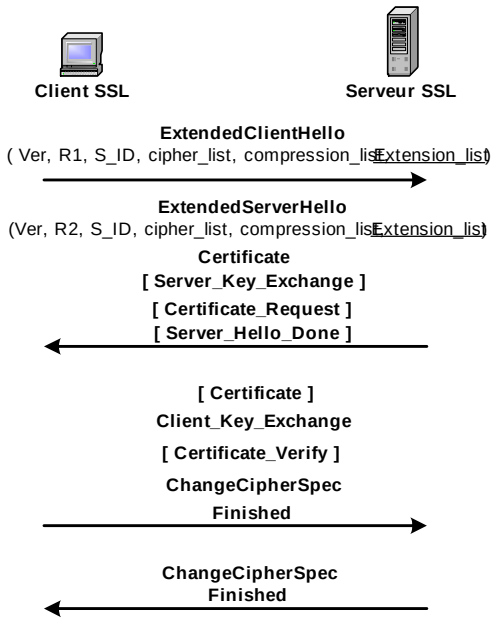


Figure 3-8. Le Handshake de TLS Extensions

Les deux champs (*ExtendedClientHello* et *ExtendedServerHello)* ci-dessous sont décrits par la syntaxe de SSL/TLS définie dans le RFC 2246. La syntaxe utilisée ressemble au langage de programmation C et à la syntaxe de XDR [RFC1832].

Le champ *ExtendedClientHello*	Le champ *ExtendedServerHello*
struct { ProtocolVersion client_version; Random random; SessionID session_id; CipherSuite cipher_suites<2..2^16-1>; CompressionMethod compression_methods<1..2^8-1>; Extension client_hello_extension_list<0..2^16-1>; }ClientHello;	struct { ProtocolVersion server_version; Random random; SessionID session_id; CipherSuite cipher_suite; CompressionMethod compression_method; Extension server_hello_extension_list<0..2^16-1>; }ServerHello;

Tableau 3-2. Table représentative des deux messages *Extended- ClientHello* et *ServerHello*

3.4.5 Le protocole Record

Le protocole Record n'intervient qu'à la suite de l'émission du message ChangeCipherSpec. Pendant la phase d'établissement de la session. Le rôle du protocole Record est d'encapsuler les données du Handshake et de les transmettre sans aucune modification vers la couche du protocole TCP.

Pendant la phase de chiffrement des données, le protocole Record fragmente les données reçues des divers protocoles de niveau élevé en des fragments de taille maximale égale à 2^{14} octets. Ensuite, il calcule un condensât, ajoute le bourrage si nécessaire [RFC2104], effectue le chiffrement, ajoute un en-tête et transmet l'unité résultante dans un segment TCP.

Le protocole Record rajoute un en-tête de 5 octets à chaque message reçu des couches supérieures (figure 3-9). Cet en-tête indique le type du message, selon qu'il provient des sous protocoles Handshake, Alert, CCS ou s'il s'agit des données d'application, par exemple, HTTP, FTP. Il signale aussi la version du protocole SSL utilisée ainsi que la longueur du bloc de données encapsulées.

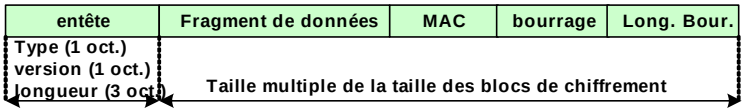


Figure 3-9. Le format d'un paquet SSL/TLS

Le protocole Record a quatre états (ou states) qui lui sont liés : *Current_read_state, current_write_state, pending_read_state et pending_write_state*.
Chaque état possède ses propres algorithmes de chiffrement, de hachage, de compression et des divers paramètres cryptographiques. Tout changement d'état par le protocole Handshake est exécuté sur l'état d'attente (*pending state*). La configuration employée par le protocole Record est liée à l'état actuel (*current state*). Quand un message est reçu de CCS, les états en attente deviennent les nouveaux états actuels.

3.4.6 Les opérations cryptographiques de SSL/TLS

Les deux tableaux suivant récapitulent les opérations cryptographiques (asymétriques) effectuées par un client et un serveur SSL/TLS dans les différents modes de Handshake de SSL/TLS.

	RSA	**DH-DSA**
Client	$RSA_{verify} + RSA_{encrypt}$	$DSA_{verify} + DH_{op}$
Serveur	$RSA_{decrypt}$	DH_{op}

Tableau 3-3. Opérations cryptographiques avec authentification serveur SSL/TLS

Dans le tableau 3-3, c'est seulement le serveur qui s'authentifie. Le client et le serveur ont le choix entre deux modes.

- Sans la propriété de PFS : ici, le client effectue deux opérations RSA. La première est la vérification du certificat serveur et la deuxième est le chiffrement su pre_master_secret. Le serveur effectue une seule opération avec sa clé privée pour déchiffrer le message ClientKeyExchange afin de récupérer le secret envoyé par le client.
- Si la propriété de PFS est exigée par le serveur, le client exécute une vérification DSA sur le message ServerKeyExchange pour vérifier le certificat serveur puis une opération DH (génération d'une valeur publique éphémère). Le serveur effectue seulement une opération DH pour arriver au même secret calculé par le client.

	RSA	**DH-DSA**
Client	$RSA_{verify} + RSA_{encrypt} + RSA_{sign}$	$DSA_{verify} + DH_{op}$ or
		$DSA_{verify} + DSA_{sign} + DH_{op}$
Server	$2*RSA_{verify} + RSA_{decrypt}$	$DSA_{verify} + DH_{op}$ or
		$2*DSA_{verify} + DH_{op}$

Tableau 3-4. Opérations cryptographiques avec authentification SSL/TLS

Dans le tableau 3-4, l'authentification et mutuelle entre le client et le serveur SSL/TLS. Les deux entités ont le choix entre deux modes.

- Sans la propriété de PFS. En plus des deux opérations cryptographiques expliquées précédemment dans le tableau 3 - 3, le client effectue une opération RSA avec sa clé privée pour la génération du message ClientVerify. Le serveur effectue deux opérations de vérification avec les clés publiques RSA (une pour la vérification du certificat client et l'autre pour la vérification de la signature du client sur le message CertificateVerify) et une opération avec sa clé privée pour déchiffrer la clé secrète.

Si la propriété de PFS est exigée par le serveur, quand le client emploie le certificat DH, les deux côtés effectuent une opération de vérification DSA sur le certificat de l'autre suivi d'une opération DH pour calculer le secret partagé. Quand le client emploie un certificat DSA, les vérifications exigées des deux côtés sont asymétriques. Le client exécute une vérification DSA sur le certificat serveur, une opération DH pour calculer le secret partagé et une signature DSA pour produire le message CertificateVerify. La première serveur exécute une opération DH pour calculer le secret partagé et deux vérifications DSA. Le premier pour vérifier le certificat client et la deuxième pour la vérification du message CertificateVerify.

3.4.7 Les avantages

SSL/TLS est un protocole de sécurité qui permet de sécuriser les échanges entre un client et un serveur SSL/TLS d'une manière transparente, en se plaçant entre les couches d'application et de transport. Les principaux avantages de ce protocole sont :

3.4.7.1 Authentification et intégrité fortes des messages.

Le dispositif primaire de SSL/TLS est la capacité de sécuriser les flux de données transitant sur des réseaux publics, tout en utilisant des algorithmes de chiffrement symétrique. SSL/TLS offre également l'authentification de serveur et sur option, l'authentification du client pour prouver les identités des deux parties. Il fournit également l'intégrité de données par une valeur de contrôle d'intégrité. En plus de la protection contre la révélation de données par le chiffrage, le protocole de sécurité SSL/TLS peut être employé pour se protéger contre les attaques de type man-in-the-middle, rejeu des paquets et attaques de changement ou baisse des versions SSL/TLS.

3.4.7.2 Interopérabilité et facilité de déploiement

SSL/TLS fonctionne avec la plupart des navigateurs Web, y compris le navigateur de Microsoft (Internet Explorer) et de Netscape (Netscape Navigator) et sur la plupart des serveurs Web inclus dans les différents systèmes d'exploitation. En outre, il est souvent intégré avec les serveurs LDAP [RFC2252] [RFC2253] et une variété d'autres applications.

3.4.7.3 Facilité d'utilisation

Puisque SSL/TLS est mis en application sous la couche application, la plupart de ses opérations sont complètement invisibles au client. Ceci permet au client d'avoir peu de connaissances sur ses communications sécurisées et d'être toujours protégé contre les attaquants.

3.4.7.4 SSL VPN

Le terme SSL VPN désigne simplement une encapsulation du trafic d'une application particulière (HTTP ou IMAP par exemple) par SSL/TLS afin d'atteindre les ressources informatiques de l'entreprise. Le SSL VPN ne concurrence pas directement les VPNs IPSec. Les constructeurs fournissant des SSL VPN les positionnent comme une solution complémentaire à IPSec dans le but de répondre aux nouveaux besoins des entreprises. Le SSL VPN est une solution qui palie aux difficultés rencontrées avec IPSec et les accès distants (notamment dus à la translation d'adresse et au filtrage d'application dans le réseau distants). Le principal avantage est de permettre aux utilisateurs nomades (télétravailleurs par exemple) de pouvoir se connecter au réseau de leur entreprise depuis n'importe quel ordinateur (PDA, cybercafé, réseau protégé par un Firewall laissant passé les flux HTTP …) sans modification à apporter sur le poste distant.

Ainsi, on parle souvent de VPN Clientless [Bert03]: il n'y a pas besoin d'installer de logiciel spécifique sur le client pour atteindre le réseau de l'entreprise, contrairement au VPN IPSec. Ceci est possible avec des sessions HTTPS car SSL/TLS est aujourd'hui répandu sur tous les systèmes d'exploitation et supporté par n'importe quel navigateur Web.
Une autre méthode pour réaliser un VPN avec SSL/TLS est inspirée du protocole SSH. En effet, le logiciel libre *stunnel* [Stunnel] fourni un tunnel SSL/TLS entre deux points (ou deux sites) à l'aide d'un mécanisme de changement de port (port forwarding) proposé par le protocole SSH.
Par ailleurs, ces propositions restent loin d'aboutir à un vrai standard VPN qui permet de multiplexer les données de plusieurs clients ainsi que celles des applications dans une seule session SSL/TLS.

3.4.8 Les inconvénients

Le protocole SSL/TLS, comme tout autre technologie novatrice dans le domaine de sécurité, possède son lot d'inconvénients :

3.4.8.1 Problèmes liés aux navigateurs

Le problème des navigateurs réside surtout dans la vérification automatique de la validité des certificats, des listes des révocations et des restrictions d'exportation et la conservation des clés privées dans des endroits sécurisés.
Vu l'absence de consultation automatique des signatures par les navigateurs, l'autorité de certification, qui a émis les certificats pour les clients ou même les sites, peut même ensuite révoquer ces certificats sans que les utilisateurs cessent d'en vérifier la validité.
Ce problème est bien relié aussi à l'usage de ces navigateurs par des utilisateurs non experts dans le domaine de la sécurité et des PKI dont SSL/TLS est bien rattaché (quand un certificat expire, l'utilisateur reçoit un message et doit obtenir manuellement un nouveau certificat, ce qui n'est pas forcément trivial pour un utilisateur quelconque).
Dans ce cadre, puisque le but était de rendre l'utilisation de SSL/TLS la plus souple possible, un problème de confiance est apparu entre les utilisateurs et la liste des certificats des autorités de certification imposée et codée en dure dans les navigateurs.

3.4.8.2 Problème cryptographique (lié à l'implémentation)

Même si SSL/TLS a été standardisé à l'IETF et approuvé cryptographiquement comme un protocole sécurisé, de nouvelles attaques sont apparues sur ce protocole ciblant la partie Handshake en essayant de recouvrir le *master_secret* dont tout le chiffrement du canal SSL/TLS est fondé.
Ces attaques sont liées surtout à l'implémentation de ce protocole dans l'API OpenSSL [OpenSSL] la plus répondue actuellement dans le domaine des « *Open Source* ».
Des chercheurs de l'EPFL (Ecole Polytechnique de Lausanne) [Canv03], de l'université de Stanford et de l'université de Prague ont trouvé des failles dans les implémentations OpenSSL version 0.9.6 liées à l'interception des mots de passe, le temps d'encryptage et de décryptage des blocs SSL/TLS et à la création des sessions basées sur RSA. Le dernier problème est le plus persistant puisque le recouvrement du *master_secret* dérive la capturation et le déchiffrement de tous les messages SSL/TLS.

3.4.8.3 Attaques liées au réseau

Une autre attaque qui peut arriver sur SSL/TLS est la construction des sites très similaires à des sites que l'on souhaite abuser. Ceci est dû, en grand sort, en trompant des serveurs DNS [RFC1034] [RFC1035] non sécurisés.

3.4.8.4 HTTPS et les Proxy

La sémantique générale de HTTPS est d'ouvrir un canal sécurisé entre deux entités (client et serveur SSL/TLS), sans permettre à aucun tiers intermédiaire d'accéder ou de partager la communication avec eux. Pour cela, les Proxy, qui joue le premier rôle dans l'optimisation de la bande passante sur les liens Internet, sont totalement inutiles avec SSL/TLS.
D'ailleurs, les Proxy exigent un support spécial de la méthode CONNECT pour pouvoir passer le HTTPS. Quand la connexion SSL/TLS est établie, le cache des Proxy est désactivé puisque les serveurs Proxy ne peuvent faire aucun contrôle sur les messages SSL/TLS.

3.4.8.5 SSL et les hôtes virtuels

HTTPS interagit faiblement avec les serveurs virtuels. Le problème résiste encore dans la faite que dans SSL/TLS, toutes les données sont envoyées après l'établissement du Handshake SSL/TLS. Normalement dans les connexions HTTP, le serveur utilise l'entête «HOST» du HTTP pour déterminer quels serveurs virtuels le client veut accéder. Puisque dans HTTPS, l'entête HTTP est envoyé après l'établissement du Handshake SSL/TLS, le serveur ne peut savoir à quels serveurs virtuels le client veut accéder (surtout si les serveurs virtuels sont sur plusieurs machines).
Une méthode pour détourner ce problème est de configurer une machine avec plusieurs adresses IP de sorte que pour chaque serveur virtuel on lui assigne une adresse IP. Une méthode plus opérante serait d'envoyer dans le message *client_hello* du Handshake de SSL/TLS la DNS du serveur que le client essaye de se connecter.

3.4.8.6 Utilisation des faibles clés de chiffrement

Théoriquement, SSL/TLS est vulnérable aux attaques par force brute (tentatives de déchiffrement direct d'un message crypté) si les clés utilisées sont de 40 bits. C'est pourquoi, il est plus sûr d'utiliser des clés de 128 bits pour se prémunir des attaques. Les caractéristiques temporelles et financières de ce type d'attaque sont facilement calculables.
Des renseignements sur le type de SSL/TLS utilisé indiquent le matériel et le temps nécessaire à l'attaque. Le chiffrement à 40 bits est de moins en moins cher et long à attaquer, même pour un pirate seul. Tant qu'il n'existe pas d'algorithmes cryptographiques capables d'attaquer directement le chiffrement 128 bits, ce dernier pourra être considéré comme sûr.

3.4.8.7 SSL/TLS et les applications basées sur UDP

L'avantage majeur de SSL/TLS est qu'il fournit un canal sécurisé transparent aux applications. Cependant, SSL/TLS exige un canal fiable comme TCP et ne peut pas être utilisé pour sécuriser des applications au-dessus de UDP [RFC768].
Quand TLS a été développé, cette limitation n'a pas été considérée particulièrement sérieuse parce que la grande majorité d'applications utilise le protocole TCP. Actuellement, cette situation a changé. Au cours de ces dernières années, un nombre croissant de protocoles de couche application, comme les protocoles (SIP, Session Initiation Protocol) [RFC3261], (RTP, Real Time Protocol) [RFC3550], (MGCP, Media Gateway Control Protocol) [RFC3435] et une variété de protocoles de jeu ont été conçus pour employer le protocole UDP. SSL/TLS est incapable d'offrir ces services de sécurité à ces types d'applications.

3.5 Le protocole IPSec

3.5.1 Introduction

IPSec est un ensemble de protocoles conçu par l'IETF afin de sécuriser le trafic IP. Initialement conçu dans la philosophie de IPv6, IPSec est un système d'encapsulation offrant les services de sécurité requis au niveau IP. Néanmoins, étant donné que les besoins en matière de sécurité sur Internet et sur les Intranets ne peuvent attendre que la totalité (ou d'au moins une grande majorité) du parc informatique mondial ait migré vers IPv6, il est nécessaire que IPSec soit également utilisable avec IPv4. Une manière commode de procéder, qui a l'avantage de garantir la compatibilité avec toutes les implémentations existantes du protocole IP sans avoir à les modifier, est de considérer le protocole IPSec comme un protocole indépendant, implémentable comme un module additionnel sous forme d'un logiciel ou d'un équipement électronique dédié.

3.5.2 Les services offerts par IPSec

Une communication entre deux hôtes, protégée par IPSec, est susceptible de fonctionner suivant deux modes différents : le mode transport et le mode tunnel.
Le premier mode offre essentiellement une protection aux protocoles de niveau supérieur, le second permet quant à lui, d'encapsuler des datagrammes IP dans d'autres datagrammes IP dont le contenu est protégé. L'intérêt majeur de ce second mode est qu'il traite toute la partie IPSec d'une communication et transmet les datagrammes épurés de leur partie IPSec à leur destinataire réel réalisable. Il est également possible d'encapsuler une communication IPSec en mode tunnel, elle-même traitée par une passerelle de sécurité, qui transmet les datagrammes après suppression de leur première enveloppe à un hôte traitant à son tour les protections restantes ou à une seconde passerelle de sécurité [Misc02].

3.5.2.1 AH (Authentication Header)

AH est le premier et le plus simple des protocoles de protection des données qui font partie de la spécification IPSec. Il est détaillé dans le RFC 2402 [RFC 2402]. Il a pour vocation de garantir :

- L'authentification : les datagrammes IP reçus ont effectivement été émis par l'hôte dont l'adresse IP est indiquée comme adresse source dans les en-têtes.
- L'unicité (optionnelle, à la discrétion du récepteur) : un datagramme ayant été émis légitimement et enregistré par un attaquant ne peut être réutilisé par ce dernier, les attaques par rejeu sont ainsi évitées.
- L'intégrité : les champs suivants du datagramme IP n'ont pas été modifiés depuis leur émission : Données (en mode tunnel, ceci comprend la totalité des champs, y compris en-têtes du datagramme IP encapsulé dans le datagramme protégé par AH), version (4 en IPv4, 6 en IPv6), longueur de l'en-tête totale du datagramme (en IPv4), longueur des données (en IPv6), identification, protocole ou en-tête suivant (ce champ vaut 51 pour indiquer qu'il s'agit du protocole AH), adresse IP de l'émetteur, adresse IP du destinataire (sans *Source Routing*).
  En outre, au cas où le *Source Routing* [RFC791] [Misc02] serait présent, le champ adresse IP du destinataire a la valeur que l'émetteur a prévu qu'il aurait lors de sa réception par le destinataire. Cependant, la valeur que prendront les champs type de service (IPv4), somme de contrôle d'en-tête (IPv4), classe (IPv6), flow label (IPv6) et hop limit (IPv6) lors de leur réception n'étant pas prédictible au moment de l'émission, leur intégrité n'est pas garantie par AH.

En plus, AH n'assure pas la confidentialité des données : les données sont signées mais pas chiffrées.

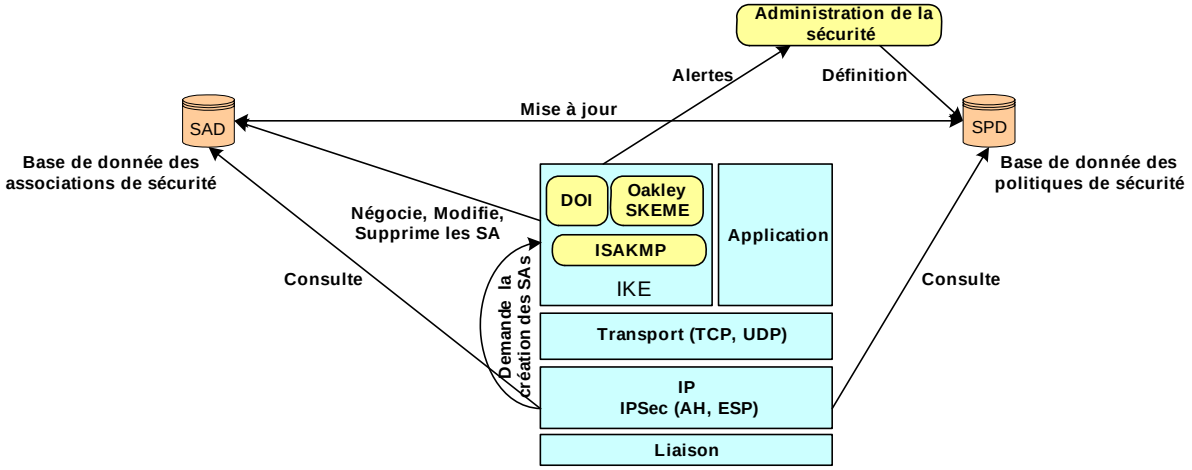


Figure 3-10. Architecture d'IPSec

3.5.2.2 ESP (Encapsulating Security Payload)

ESP est le second protocole de protection des données qui fait partie de la spécification IPSec. Il est détaillé dans la RFC 2406 [RFC2406]. Contrairement à AH, ESP ne protège pas les en-têtes des datagrammes IP utilisés pour transmettre les communications. Seules les données sont protégées. En mode transport, il assure :

- La confidentialité des données (optionnelle) : la partie données des datagrammes IP transmis est chiffrée.
- L'authentification (optionnelle, mais obligatoire en l'absence de confidentialité) : la partie données des datagrammes IP reçus ne peut avoir été émise que par l'hôte avec lequel a lieu l'échange IPSec, qui ne peut s'authentifier avec sucées que s'il connaît la clé associée à la communication ESP. Il est également important de savoir que l'absence d'authentification nuit à la confidentialité en la rendant plus vulnérable à certaines attaques actives.
- L'unicité (optionnelle, à la discrétion du récepteur).
- L'intégrité : les données n'ont pas été modifiées depuis leur émission.

En mode tunnel, ces garanties s'appliquent aux données du datagramme dans lequel est encapsulé le trafic utile, donc à la totalité (en-têtes et options inclus) du datagramme encapsulé. Dans ce mode, deux avantages supplémentaires apparaissent :

- Une confidentialité limitée des flux de données (en mode tunnel uniquement, lorsque la confidentialité est assurée) : un attaquant capable d'observer les données transitant par un lien n'est pas à même de déterminer quel volume de données est transféré entre deux hôtes particuliers. Par exemple, si la communication entre deux sous réseaux est chiffrée à l'aide d'un tunnel ESP, le volume total de données échangées entre ces deux sous réseaux est calculable par cet attaquant, mais pas la répartition de ce volume entre les différents systèmes de ces sous réseaux.
- La confidentialité des données, si elle est demandée, s'entend à l'ensemble des champs, y compris les en-têtes du datagramme IP encapsulé dans le datagramme protégé par ESP.

Enfin, ESP et AH ne spécifient pas d'algorithmes de signature et/ou de chiffrement particuliers. Ceux-ci sont décrits séparément.
Une implémentation conforme au RFC 2402 (AH) est tenue de supporter les algorithmes MD5 [RFC1321] et SHA-1[SHA] tandis qu'une implémentation conforme au RFC 2406 (ESP) est tenue de supporter l'algorithme de chiffrement DES [DES] en mode CBC et les signatures à l'aide des fonctions de hachage MD5 et SHA-1.

	Mode transport	**Mode tunnel**
AH	Authentifie l'information utile IP et certaines portions de l'en-tête IP et les en-têtes d'extension IPv6.	Authentifie le paquet IP tout entier plus certaines portions de l'en-tête externe et des en-têtes d'extension IPv6 externes
ESP	Chiffre l'information utile IP et tout en-tête d'extension IPv6 suivant l'en-tête ESP.	Chiffre le paquet IP tout entier.
ESP avec authentification	Chiffre l'information utile IP et tout en-tête d'extension IPv6 suivant l'en-tête ESP. Authentifie l'information et utilise IP mais pas l'en-tête IP.	Chiffre le paquet IP tout entier. Authentifie le paquet IP.

Tableau 3-5. Fonctionnalité des modes tunnel et transport

3.5.3 Architecture

La RFC 2401 décrit le protocole IPSec au niveau le plus élevé. En particulier, elle indique ce qu'une implémentation est censée permettre de configurer en termes de politique de sécurité (c.-à-d. quels échanges IP doivent être protégés par IPSec et le cas échéant, quel(s) protocole(s) utiliser). Sur chaque système capable d'utiliser IPSec, doit être présente une SPD (*Security Policy Database),* dont la forme précise est laissée au choix de l'implémentation, et qui permet de préciser la politique de sécurité à appliquer au système. Chaque entrée de cette base de données est identifiée par un SPI (*Security Parameter Index*) unique (32 bits).
Une communication protégée à l'aide d'IPSec est appelée une SA (*Security Association*). Une SA repose sur une unique application de AH ou sur une unique application de ESP. Ceci n'exclut pas l'usage simultané de AH et de ESP entre deux systèmes, ou par exemple l'encapsulation des datagrammes AH dans d'autres datagrammes AH, mais plusieurs SA devront alors être activées entre les deux systèmes. En outre, une SA est unidirectionnelle. La protection d'une communication ayant lieu dans les deux sens nécessitera donc l'activation de deux SA. Chaque SA est identifiée de manière unique par un SPI, une adresse IP de destination (éventuellement adresse des broadcast ou de multicast) et un protocole (AH ou ESP). Les SA actives sont regroupées dans une SAD (*Security Association Database*).
La SPD est consultée pendant le traitement de tout datagramme IP, entrant ou sortant, y compris les datagrammes non IPSec. Pour chaque datagramme, trois comportements sont envisageables : le rejeter, l'accepter sans traitement IPSec, ou appliquer IPSec. Dans le troisième cas, la SPD précise en outre quels traitements IPSec appliquer (ESP, AH, mode tunnel ou transport, quel(s) algorithme(s) de signature et/ou de chiffrement utiliser).
Les plus importants de ces termes sont les suivantes :

SPD Base de données définissant la politique de sécurité
SA Une entrée de la SPD
SAD Liste des SA en cours d'utilisation

Les règles de la SPD doivent pouvoir, si l'administrateur du système le souhaite, dépendre des paramètres suivants :

- Adresse ou groupe d'adresses IP de destination ;
- Adresse ou groupe d'adresses IP source ;

- Nom du système (DNS complète, nom X.500 distingué ou général) ;
- Protocole de transport utilisé (typiquement, TCP ou UDP) ;
- Nom d'utilisateur complet, comme hajjeh@enst.fr (ce paramètre n'est toutefois pas obligatoire sur certains types d'implémentation) ;
- Importance des données (ce paramètre n'est toutefois pas obligatoire sur certains types d'implémentation) ;
- Ports source et destination (UDP et TCP seulement, le support de ce paramètre est facultatif).

Certains paramètres peuvent être illisibles à cause d'un éventuel chiffrement ESP au moment où ils sont traités, auquel cas la valeur de la SPD les concernant peut le préciser. Il s'agit de :

- L'identité de l'utilisateur
- Le protocole de transport
- Les ports source et destination.

Il est donc possible de spécifier une politique de sécurité relativement fine et détaillée. Cependant, une politique commune pour le trafic vers un ensemble des systèmes, permet une meilleure protection contre l'analyse de trafic qu'une politique extrêmement détaillée, comprenant de nombreux paramètres propres à chaque système particulier.
Enfin, si l'implémentation permet aux applications de préciser elles-mêmes à quelle partie du trafic appliquer IPSec et comment, l'administrateur doit pouvoir les empêcher de contourner la politique par défaut.

3.5.4 La gestion des clés dans IPSec

La présence de mécanismes de chiffrement asymétriques implique la prise en compte des problématiques de gestion des clés et de leur distribution à l'ensemble des systèmes destinés à être source et/ou destination d'une communication IPSec.
Dans le cas de petites infrastructures, il est possible, voire préférable, d'opter pour une gestion et une distribution manuelle des clés. Cette simplicité technique doit cependant être couplée à une bonne organisation et à une certaine rigueur afin d'une part, d'en assurer un fonctionnement correct et d'autre part de respecter les critères de renouvellement des clés.
Cependant, une telle méthode n'est pas applicable sur des infrastructures comprenant de nombreux systèmes, ceci pour d'évidentes raisons de volumes de données à traiter et de délais de mise en œuvre. Le protocole IKE (Internet Key Exchange) [RFC2409] a par conséquent été défini comme protocole par défaut pour la gestion et la distribution des clés. La gestion des clés est assurée par le protocole ISAKMP [RFC2408], le principe d'échange assuré par un mécanisme dérivé d'Oakley [RFC2412].
Le rôle d'ISAKMP est essentiel. Il fourni un mécanisme « générique » pour l'échange des messages et la gestion des clés. Il définit en particulier, le support des opérations de négociation de ces dernières.
En raison d'un grand intérêt que porte ce protocole dans la première contribution de cette thèse, nous préférons détailler les protocoles ISAKMP, IKEv1 et les trois successeurs de ce dernier (JFK [JFK], SIGMA [Kraw03] et IKEv2 [IKEv2]) dans la section 3.6.

3.5.5 Les avantages

Le fait que IPSec soit un protocole de sécurité au niveau réseau, il permet d'avoir un niveau de sécurité très élevé par rapport aux autres solutions de couches applicatives. [Mark97] et [Stal02] énumèrent les avantages d'IPSec et les présentes comme suit :

3.5.5.1 Protection contre le contournement et l'analyse du trafic

IPSec est le seul protocole qui répond à la nécessité de protéger l'infrastructure de réseau contre un contrôle en une surveillance non autorisés du trafic et de sécuriser le trafic entre utilisateurs, en utilisant des mécanismes d'authentification et de chiffrement. Il protège alors contre l'usurpation d'adresse IP (*IP Spoofing*) [Wrig94] et les diverses formes d'écoute et de changement de paquets.

3.5.5.2 Protocole transparent aux applications

IPSec est au-dessous de la couche transport (TCP, UDP), il est donc transparent aux applications. IPSec ne nécessite aucun changement de logiciel sur un système utilisateur ou serveur quand IPSec est installé sur un pare-feu ou un routeur. Même si IPSec est mis en œuvre dans des systèmes finaux, le logiciel de la couche supérieure, y compris les applications, ne sera pas affecté.

3.5.5.3 Mise en œuvre d'un VPN

Le grand intérêt de IPSec par rapport à d'autres techniques (par exemple les tunnels SSH), est qu'il s'agit d'une méthode standard (facultative en IPv4 et obligatoire en IPv6), mise au point dans ce but précis décrite par différentes RFCs.

D'abord et avant toute autre considération, un VPN IPSec apporte de multiples éléments assurant la sécurité des réseaux. Il réduit considérablement les risques d'intrusions depuis l'extérieur sous forme d'usurpation d'adresse IP et d'écoute passive. De plus, un VPN IPSec protège un intranet des virus tels que les worms, spécifiquement conçus pour répandre à grande vitesse sur Internet. Il reste la solution la plus déployée pour relier les réseaux des entreprises à travers un réseau non sécurisé comme Internet.

Il est aussi utile pour les télétraveilleurs qui souhaitent communiquer avec le serveur de leur entreprise et pour installer un sous réseau virtuel sécurisé dans une organisation pour des applications sensibles.

Toutefois, un VPN IPSec ne garantit pas des indiscrétions ou des attaques venant de l'intérieur. Différentes études ont montré qu'en général, le piratage interne est plus dommageable, plus coûteux et aussi plus fréquent.

3.5.6 Les inconvénients

Les problèmes rencontrés avec le protocole IPSec sont relatifs à l'ambiguïté documentaire, l'implémentation et à la redondance des fonctionnalités [Ferg00].

3.5.6.1 Interopérabilité entre les implémentations

A cause de l'ambiguïté et la complexité dans la documentation des différentes parties de ce protocole [Ferg00], nous sommes arrivés actuellement à des implémentations incompatibles. De plus, puisque IPSec est intégré dans le noyau des systèmes, l'interopérabilité d'IPSec dépendra de la volonté qu'auront les constructeurs à faire coopérer des équipements hétérogènes entre eux (cela pose notamment le problème des algorithmes cryptographiques communs).

3.5.6.2 Redondances des fonctionnalités

Les fonctionnalités des protocoles IPSec sont redondantes. Par exemple, on peut utiliser l'un ou l'autre des protocoles AH ou ESP à des fins d'authentification. Cette situation peut entraîner des faiblesses dans les différentes versions du produit IPSec.

Compte tenu de la souplesse des protocoles, certaines combinaisons peuvent s'avérer inacceptables au plan de la sécurité. Par exemple, le protocole ESP offre des services d'authentification et de chiffrement mais n'empêche pas l'utilisateur de transmettre un message chiffré sans authentification.

3.5.6.3 Broadcast et multicast

L'utilisation d'IPSec pour l'envoi et la réception de datagrammes multicast et broadcast pose quelques problèmes liés à des difficultés qu'une simple implémentation de la puissance de calcul ne saurait résoudre, comme la vérification des numéros de séquence. Le service de protection contre la duplication des données n'est donc pas utilisable en environnement multicast et broadcast à l'heure actuelle.

3.5.6.4 NATs

Théoriquement, aucune translation d'adresse ne devrait affecter un datagramme IPSec, car ce type d'opération modifie le contenu des datagrammes, ce qui est incompatible avec les mécanismes de protection de l'intégrité des données d'IPSec. Par exemple, en mode AH (Authentication Header), l'encapsulation IPSec ajoute une somme de contrôle (*checksum*), calculée notamment sur les adresses source et destination. Le protocole IKE, lui aussi se base sur les adresses IP sources et destinations pour la génération des clés et des cookies.

Il existe une solution proposée actuellement comme un draft à l'IETF : l'*IPSec NAT-Traversal* [Kivi04]. Il s'agit d'encapsuler les données dans un tunnel UDP. Ainsi, de la même façon qu'en mode tunnel et ESP, l'en-tête modifié par la NAT ne sera pas utilisé pour le test d'authentification. Certains équipements permettent déjà de traverser des NAT, en utilisant des protocoles plus ou moins normalisés.

3.6 Les protocoles ISAKMP et IKEv1

3.6.1 Introduction

Le protocole ISAKMP (Internet Security Association and Key Management Protocole), défini dans le RFC 2408 de novembre 1998 [RFC2408], est proposé pour la gestion des associations de sécurité. ISAKMP fournit un cadre générique pour la négociation, la modification et la destruction des SA ainsi que pour le format de leurs attributs.

Dans le cadre de la standardisation de IPSec, ISAKMP est associé à une partie des protocoles SKEME [Kraw96] et Oakley [RFC2412] pour donner un protocole final du nom de Internet Key Exchange IKE [Hajj03d].
ISAKMP n'impose pas un algorithme spécifique d'échanges de clés. Il se compose plutôt d'un ensemble de types de messages qui permettent l'utilisation de divers algorithmes d'échanges de clés. Cependant, Oakley est l'algorithme spécifique d'échange de clés exigé pour la version initiale d'ISAKMP.

Pour IPSec, IKE est complété par un « domaine d'interprétation » ou (Domain of Interpretation, DOI) [RFC2407]. C'est dans ce document qu'on défini les algorithmes d'échanges des clés et les différents blocs ISAKMP.

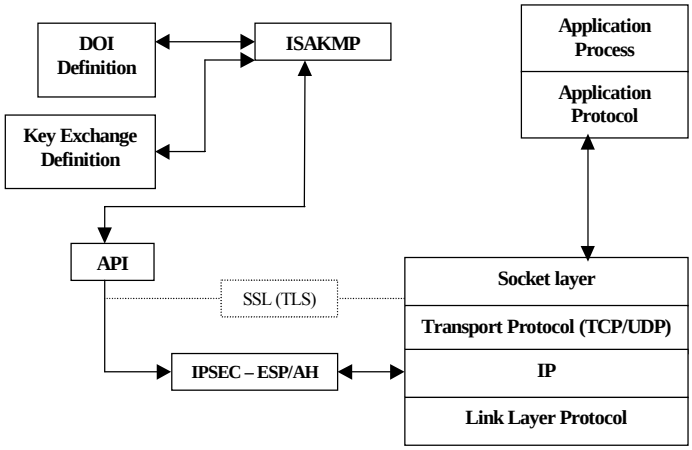


Figure 3-11. Architecture d'ISAKMP

3.6.2 Architecture

Si le DOI IPSec précise le contenu des blocs ISAKMP dans le cadre d'IPSec, IKE en revanche, utilise ISAKMP pour construire un protocole pratique. Il utilise certains des modes définis par Oakley et emprunte à SKEME son utilisation du chiffrement à clé publique pour l'authentification et sa méthode de changement de clef rapide par échanges d'aléas.
Bien que ce nom insiste sur le rôle d'échange de clés, IKE se charge en réalité de la gestion (négociation, mise à jour, suppression) de tous les paramètres relatifs à la sécurisation des échanges. Contrairement aux mécanismes AH (Authentification Header) et ESP (Encapsulating Security Payload) qui agissent directement sur les données à sécuriser, IKE est un protocole de plus haut niveau, dont le rôle est l'ouverture et la gestion d'une pseudo connexion au-dessus de tout protocole de transport ou sur IP directement (les réalisations doivent cependant supporter au moins le protocole UDP, l'IANA a attribué le port 500 à ISAKMP).
IKE inclut, au début de la négociation, une authentification mutuelle des tiers communicants qui peut se baser soit sur un secret partagé préalablement, soit sur des clés publiques. L'échange des clés publiques utilisées par IKE peut se faire soit manuellement, soit directement dans le cadre d'IKE par un échange de certificats en ligne, ou encore par le biais d'une infrastructure à clés publiques (public key infrastructure, PKI) extérieure.

DOI définit la syntaxe des paramètres, les types d'échanges et les conventions de nommage relatifs à l'emploi d'ISAKMP dans un cadre particulier. Un identifiant (DOI identifier) est par conséquent utilisé pour interpréter et synthétiser la charge utile des messages ISAKMP dans le contexte d'un DOI donné (figure 3-9). Par exemple, le protocole IPSec a le numéro 1 comme identifiant pour son DOI. L'utilisation d'ISAKMP avec un nouveau protocole (tel que le cas que nous proposons pour SSL/TLS) exige la définition d'un nouveau DOI propre à ce protocole [Hajj03a].

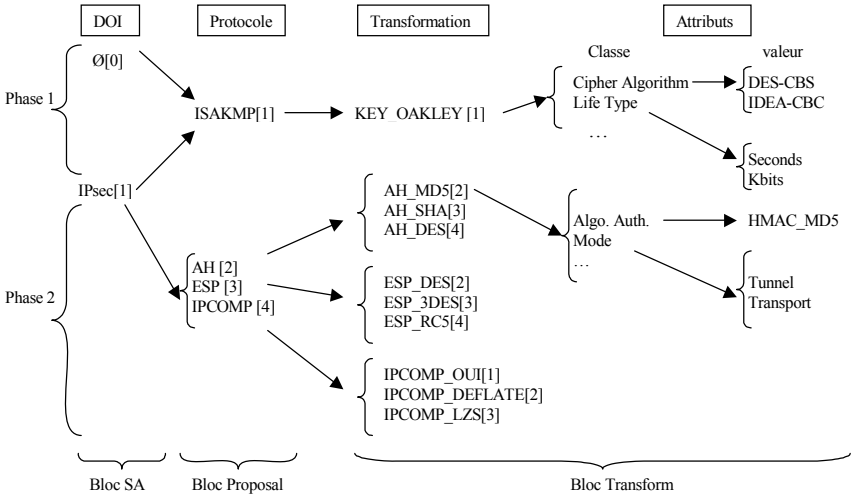


Figure 3-12. Présentation symbolique du domaine d'interprétation d'ISAKMP IPSec

3.6.3 Les échanges ISAKMP/IKEv1

ISAKMP comprend deux phases de communication qui permettent une séparation claire de la négociation des SA pour un protocole spécifique et la protection du trafic ISAKMP.

La première phase entame la négociation d'une association de sécurité relative au protocole ISAKMP. Une fois les paramètres partagés et l'échange authentifié, la SA ISAKMP est établie. Celle-ci définit la manière dont deux entités ISAKMP vont sécuriser leurs échanges ultérieurs. Ceci constitue une première « association de sécurité » connue sous le nom de SA ISAKMP.

La deuxième phase permet la négociation des nouveaux paramètres de sécurité liés à un autre protocole, comme AH (Authentication Header) [RFC2402] et ESP (Encapsulating Security Payload) [RFC2406]. Les échanges de cette phase sont protégés par l'association de sécurité établie dans la phase 1 (SA ISAKMP)

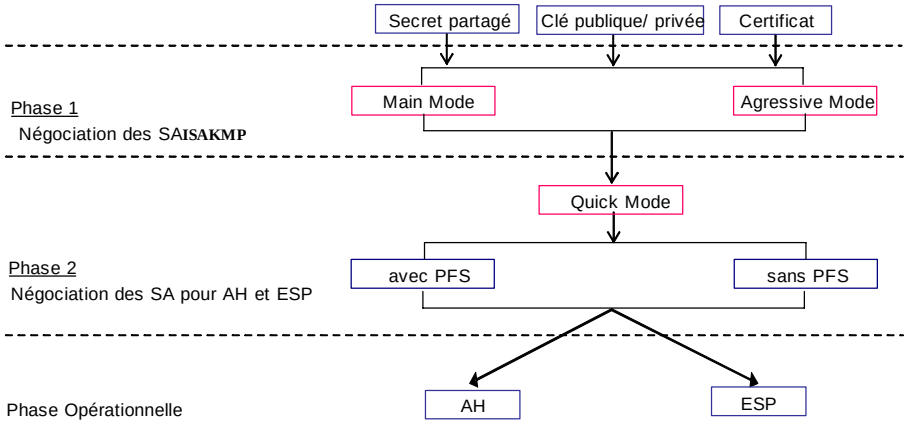


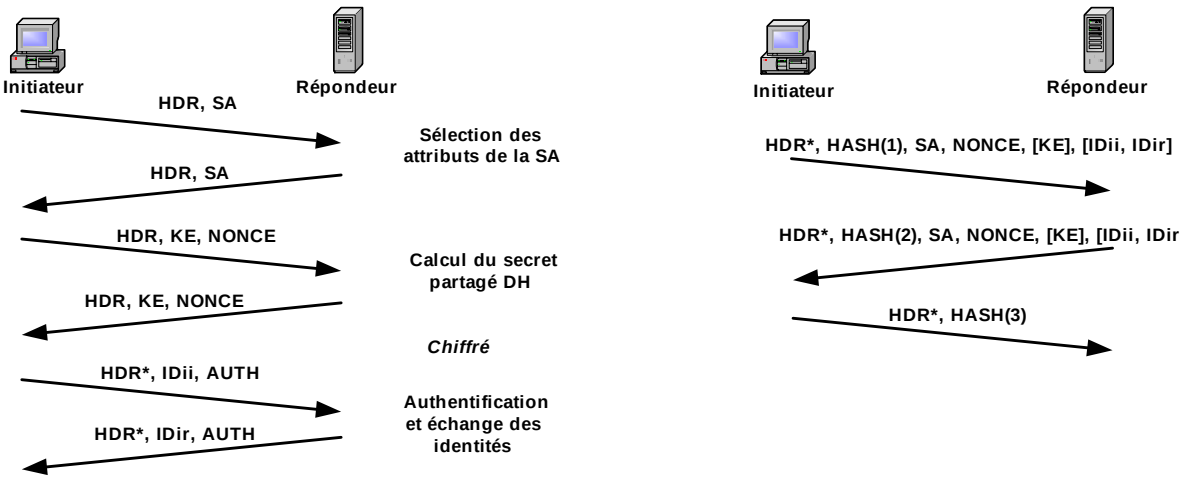
Figure 3-13. Les phases d' IKEv1

Pour les échanges ISAKMP, le RFC 2408 a défini cinq types d'échanges pour la phase 1 : *Base Exchange, Identity Protection Exchange, Authentication Only Exchange, Aggressive Exchange et Informational Exchange*. Pour la phase 2, la définition des types d'échange est laissée au protocole qui utilise ISAKMP pour satisfaire ces propres besoins en terme de

gestion des clés et de sécurité. Dans le cadre du protocole IKE, il utilise une instance de l'échange *Identity Protection Exchange (ou main mode)* et *Aggressive Exchange* comme des échanges de la phase 1 de IKEv1. L'échange *Aggressive Exchange (ou quick mode)* est aussi défini comme un échange de la phase 2 de IKEv1 avec quelques messages optionnels (figure 3-13).

3.6.4 Fonctionnement des deux phases

Lorsque des services de sécurité sont employés, il est nécessaire de savoir sans ambiguïté à quelle SA se rapportent les échanges. Les phases de négociations permettent leur identification par utilisation de champs dédiés. Cette partie est décrite dans la figure 3-14 a.
Lors de la première phase, l'initiateur émet un cookie (*Initiator Cookie*) dans l'en-tête (HDR) ISAKMP du premier message qui comprend :
- Un bloc SA (*SA Payload*), précisant si l'association de sécurité ISAKMP négociée est ou non générique.
- Un bloc proposal (*Proposal Payload*), indiquant que le protocole auquel se rapporte la SA est ISAKMP.
- Un ou plusieurs blocs Transform (*Transform Payload*), chacun d'entre eux proposant différents attributs pour la SA. Leur syntaxe est spécifiée par IKE.



a) Echange Phase 1 d'IKE (Main mode) b) Echange Phase 2 d'IKE (Quick mode)

Figure 3-14. Les Echanges de IKEv1

Le destinataire communique en réponse son propre cookie (*Responder Cookie*) et retourne le triplet (*SA, Proposal, Transform*) correspondant à la solution retenue. A ce stade, le couple (*Initiator Cookie, Responder Cookie*) qui identifie de façon unique la SA ISAKMP, est inséré dans tous les messages des deux phases. Les deux entités continuent la négociation en échangeant leur valeur publique DH à travers les blocs KE afin de créer une clé partagée et trois autres clés pour la dérivation des nouveau clés, le chiffrement et l'authentification des données. A ce point, les nouveaux messages envoyés seront protégés en confidentialité et en intégrité. L'initiateur et le répondeur échangent leurs identités et s'authentifient à travers les blocs ID et AUTH respectivement. Les identités sont des adresses IP ou des champs liés au certificat X.509. Cependant, le bloc AUTH représente trois méthodes d'authentification : des clés partagés, des certificats X.509 ou le chiffrement avec des clés publiques. Au terme de la première phase, les interlocuteurs jouent un rôle symétrique puisque les deux peuvent initier une négociation de phase 2. Une SA ISAKMP est par nature bidirectionnelle.

Dans la phase 2 (figure 3-14 b), les deux communicateurs négocient à nouveaux les paramètres de sécurité de IPSec. Chaque négociation aboutit en fait à deux SA, une dans chaque sens de la communication [Labo00]. Plus précisément, les échanges composant ce mode ont le rôle suivant :

- Négocier un ensemble de paramètres IPSec (paquets de SA).
- Échanger des nombres aléatoires utilisés pour générer une nouvelle clé qui dérive du secret généré en phase 1 avec le protocole Diffie-Hellman. De façon optionnelle, il est possible d'avoir recours à un nouvel échange Diffie-Hellman afin d'accéder à la propriété de *Perfect Forward Secrecy* qui n'est pas fournie si on se contente de générer une nouvelle clé à partir de l'ancienne et des aléas.
- Identifier le trafic que ce paquet de SA protégera, au moyen de sélecteurs (blocs optionnels *IDi* et *IDr*. En leur absence, les adresses IP des interlocuteurs sont utilisées).

Notons finalement, que les exécutions de la phase 1 doivent être rarement faites [Hall00]. En effet, dans la phase 1, les exécutions cryptographiques sont les plus intensives (tableau 3-6). Cette phase est conçue pour échanger en toute sécurité une clé secrète, même dans le cas quand il n'y a absolument aucune protection de sécurité en place entre les deux machines. Cette clé secrète est basée sur des valeurs publiques DH et autres paramètres échangés durant la première phase [Hajj03c]. En revanche, les échanges de la phase 2 sont moins complexes, puisqu'ils sont utilisés seulement après que le mécanisme de protection de sécurité, négocié dans la phase 1, est lancé.
En général, les négociations en phase 1 sont exécutées une fois par jour ou peut-être une fois par semaine, alors qu'on exécute des négociations de la phase 2 une fois toutes les quelques minutes [IbmVPN].

Echanges ISAKMP/IKEv1	Temps en seconde (s)
Phase 1 : Main Mode (3DES, SHA, DH groupe 2, certificat X.509 avec une chaine de certification égale à 1)	0.201004
Phase 1 : Main Mode (3DES, SHA, DH groupe 2, clé partagée)	0.188085
Phase 2 : Quick Mode (3DES, SHA, DH groupe 2) : avec PFS	0.118204
Phase 2 : Quick Mode (3DES, SHA) : sans PFS	0.006008

Tableau 3-6. Temps de négociation des deux phases ISAKMP/IKEv1[6]

3.6.5 Les opérations cryptographiques d'ISAKMP phase 1

Le tableau 3-7 décrit les opérations cryptographiques effectuées pendant la négociation de la phase 1 (identity protection) d'ISAKMP. Les trois modes d'authentification supportés sont :

- L'initiateur et le répondeur s'authentifient avec des certificats X.509 et une signature RSA. Dans ce cas, les deux entités possèdent à signer les données échangées en utilisant la clé privée associée à leur certificat X.509. Chacun de son coté doit vérifier le certificat envoyé et générer des valeurs DH éphémères pour l'échange des clés.
- Dans la deuxième mode d'authentification, chaque entité doit chiffrer avec la clé publique de l'autre bout de la communication, son identité et le nombre aléatoire. L'échange des clés se base aussi sur la génération des valeurs DH éphémères.

[6] Note : Ce benchmark est mesuré sur deux machines Dell (host-to-host) avec des processeurs cadensés à 1Ghz, 256M comme RAM, et Windows 2000Pro comme système d'exploitation.

Dans le troisième mode d'authentification, les deux entités utilisent les clés publiques pour le chiffrement des nombres aléatoires. Il génère des valeurs DH éphémères pour l'échange des données.

Phase 1 (ISAKMP Identity Protection)			
	RSA SIG	**RSA ENC**	**RSA ENC Revised**
Initiateur	$DH_{op} + RSA_{sign} + RSA_{verify}$	$DH_{op} + 2\ RSA_{enc} + 2\ RSA_{dec}$	$DH_{op} + RSA_{enc} + RSA_{dec}$
Répondeur	$DH_{op} + RSA_{sign} + RSA_{verify}$	$DH_{op} + 2\ RSA_{enc} + 2\ RSA_{dec}$	$DH_{op} + RSA_{enc} + RSA_{dec}$

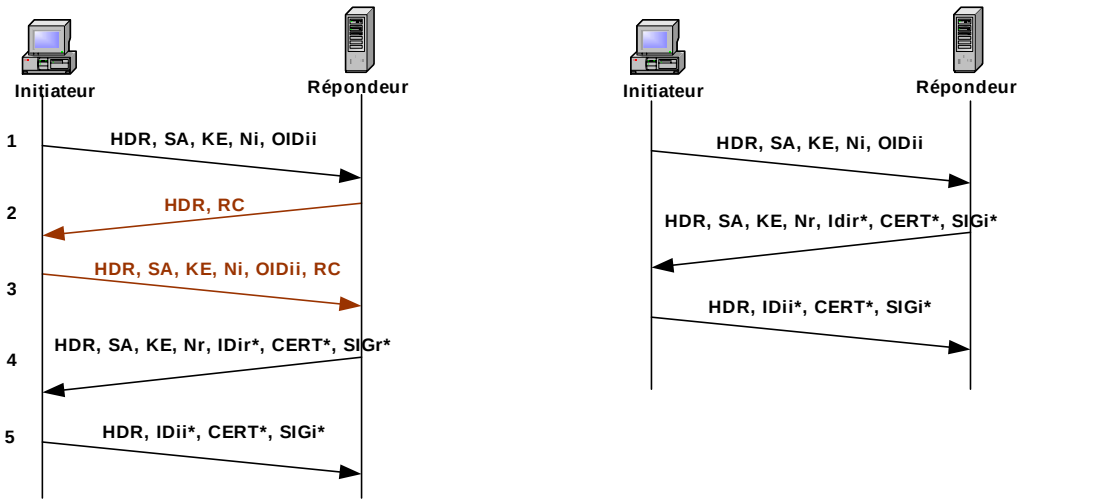
Tableau 3-7. Opérations cryptographiques dans la phase 1 d'ISAKMP

3.6.6 Les successeurs d'ISAKMP et IKEv1

Après trois ans de déploiement de IPSec avec IKEv1, l'IETF a décidé de publier une nouvelle norme relative à la sécurité des VPN en IPSec, plus efficace et plus facile à configurer. Cette norme remplacerait le protocole IKEv1 qui a paru trop fragile et s'est montré trop complexe à mettre en oeuvre. Parmi trois propositions, JFK, SIGMA et IKEv2, et après deux ans de délibération et de tergiversation, le groupe de travail d'IPSec a choisi IKEv2 comme nouveau successeur de IKEv1. Le protocole IKEv2 sera moins flexible que la première version, mais pourra être utilisable sur des VPN, mettant en jeu des fournisseurs d'équipements différents. La norme est actuellement en état de *IETF Internet Draft* mais semble avancer très vite vers la normalisation. Nous donnons ainsi une description des trois propositions SIGMA, JFK et IKEv2.

3.6.6.1 Le protocole SIGMA

SIGMA (ou *SIGnature Mode of Authentication*) est un protocole proposé par Hugo Krawczyk en novembre 2000 à l'IETF [Kraw03] [SIGMA]. La signification originale de SIGMA était «*SIGN-and-MAC*» qui a été conçu en 1994 pour l'inclusion dans Photuris [RFC2522] et qui est devenu plus tard, la base pour la mode de signature de IKEv1. Les principales motivations derrière la conception du SIGMA sont de fournir une solution flexible au problème d'échange des clés avec peu de conditions et de paramètres. SIGMA fourni aussi différents niveaux de protection d'identité pour les deux communicateurs.



a) L'échange complet de SIGMA

b) L'échange abrégé de SIGMA (sans protection contre le DOS)

Figure 3-15. Les échanges de SIGMA

SIGMA est un protocole basé sur Diffie-Hellman et les messages ISAKMP. Son mécanisme de sécurité est basé sur l'échange des valeurs publiques Diffie-Hellman authentifiées avec une signature digitale et un hachage sur les identités des communicants. SIGMA est basé sur Quick mode de IKEv1.
Il permet aussi une protection facultative contre les attaques de déni de services (avec le bloc RC ou *Routability Control*). Cette option est décidée par le répondeur en se basant sur sa politique de protection. De cette façon, l'échange de base de SIGMA est essentiellement composé de trois messages (message 1,4 et 5 dans la figure 3-15a) et cinq messages avec la protection contre le DOS.

Dans la figure (3-15a), l'initiateur envoie dans le premier échange le bloc SA contenant une liste de protocoles proposés (par exemple, AH, ESP, etc.), un nombre aléatoire dans le bloc Ni et la clé publique DH de l'initiateur dans le bloc KE. A la différence de IKEv1, SIGMA ajoute optionnellement le bloc OIDii envoyé par l'initiateur pour permettre au répondeur de prendre des décisions de politique juste après la réception du premier message de l'initiateur. Dans les implémentations, le serveur (ou le répondeur) doit surveiller le nombre de connexions semi-ouvertes puis activer son mécanisme de sécurité (par ex. anti-DOS) si ce nombre excède une certaine limite. Dans ce cas, le répondeur lance son mécanisme de sécurité et envoie le message 2 à l'initiateur qui contient le bloc RC et un cookie généré par le client. Le client doit répondre en renvoyant le premier message après avoir ajouté le bloc RC. Le répondeur continue l'échange en envoyant sa valeur publique DH, un nombre aléatoire et il chiffre son certificat, son identité et son bloc de signature et avec sa clé de session (SKEYID_e). La signature est identique à celle utilisée dans IKEv1 mais SIGMA a changé le calcul des informations haché et signé. Dans SIGMA, l'initiateur signera la valeur hachée HASH_I et le répondeur signera la valeur hachée HASH_R où :

HASH_I = prf (SKEYID, 0 | Nr, IDii_b, MSG_I).
HASH_R = prf (SKEYID, 1 | Nr, IDir_b, MSG_R).

Où MSG_I est la concaténation du message 1 envoyé par l'initiateur et du message 3, à l'expcetion du champ SIG_I. MSG_R est le contenu du message 2 à l'expcetion du champ SIG_R.
La fonction PRF (Pseudo Random Function) est une fonction pseudo aléatoire. Souvent une clé secrète utilisée dans une fonction HMAC et appliquée sur un message. Le résultat de cette fonction est pseudo aléatoire et de taille arbitraire. Le PRF est utilisé pour les dérivations des clés et pour l'authentification des messages [RFC2104].

3.6.6.2 Le protocole JFK

Just Fast Keying ou JFK est un protocole de négociation de clés développé par le laboratoire de recherche AT&T [JFK]. JFK essaye de simplifier le mécanisme de négociation des clés de IPSec et de résister contre les attaques de déni de services.
JFK est à l'origine basé sur le protocole d'échange de clés ISO 9798-2 [ISO9798] avec quelques modifications pour assurer le service de protection partielle des identités.
Il incorpore deux approches pour la protection d'identités de l'initiateur et du répondeur. La première approche appelée JFKi protège juste l'identité de l'émetteur. Ce mécanisme peut être utilisé dans des connexions client serveur où l'identité du récepteur est publique.
Dans la deuxième approche appelée JFKr, l'identité de l'émetteur et du récepteur sera protégée. Ce mécanisme peut être utilisé dans des connexions sécurisées de bout en bout.
Les messages JFK se composent de champs divers comprenant des cookies, des clés publiques DH, des nombres aléatoires, des valeurs signées, des hachages et des valeurs

d'identification. Les messages JFK ne sont pas basés sur ISAKMP mais la session résultante avec les clés dérivées sont compatibles avec IPSec. La figure 3-16 montre l'échange standard de JFK.

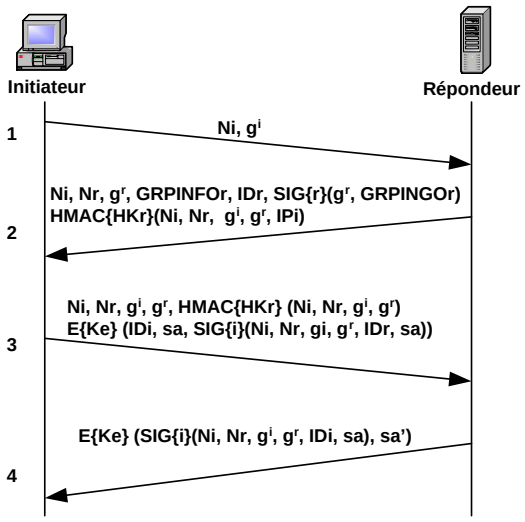


Figure 3-16. L'échange JFK

Dans cet échange, l'initiateur envoie son exponentielle DH (g^i) et un nombre aléatoire (Ni). JFK suppose que l'initiateur connaît un groupe DH (GRPINFO) qui semble être acceptable au répondeur. Avec cette méthode, l'exponentielle publique de l'initiateur peut être réutilisée dans de multiples connexions ou instances JFK. Le répondeur signe son exponentielle DH (g^r) et le groupe GRPINFOr. Le répondeur envoie également un nombre aléatoire (Nr), son identité (IDr), qui peut inclure son certificat public et un HMAC sur l'exponentielle, le nonce (Ni) et l'adresse réseau de l'initiateur (IPi). Ce HMAC est calculé à partir du secret HKr, calculé également par le répondeur. La signature SIG est aussi appliquée sur l'exponentielle du répondeur et le groupe négocié pour assurer un service de PFS liés aux valeurs DH générées.

Dans le message 3, l'initiateur renvoie les données envoyées par le répondeur, y compris l'authentificateur (H*MAC*) pour protéger le répondeur contre des attaques de type DOS. Le message inclut également l'identité de l'initiateur (*IDi*), la demande de service (*SA*), une signature calculée au-dessus des nonces (*Ni, Nr*), de l'identité du répondeur et des deux exponentielles (*gi et gr*). La dernière information est chiffrée sous une clé *Ke* dérivée des exponentielles DH (*gi et gr*) et des nombres aléatoires (*Ni et Nr*). Le répondeur renverra l'information signée dans le message trois et ajoutera des informations spécifiques au répondeur (*sa'*) sous un mécanisme de chiffrage avec la clé *Ke*.

3.6.6.3 Le protocole IKEv2

IKEv2 est proposé par Radia Perlman pour une réécriture et un nettoyage complet du protocole IKEv1. IKEv2 est décrit dans un document unique qui regroupe les informations précédemment réparties dans les RFC 2407 (domaine d'interprétation), 2408 (ISAKMP) et 2409 (IKE).

IKEv2 prend en compte les nombreuses remarques faites au cours du temps sur IKEv1, cherche à clarifier au maximum l'utilisation du protocole et à rendre son implémentation plus facile. Mais IKEv2 introduit aussi une refonte importante, puisque il propose de passer à un échange en quatre messages. De ce fait, le nouveau protocole serait tellement différent de

l'actuel qu'il ne serait pas compatible, d'où le changement de numéro de version majeure. Pour cela, IKEV2 a comme objectives:

1. Simplifier le protocole IKEv1.
2. Enlever les conditions inutiles d'ISAKMP et IKEv1.
3. Incorporer de nouvelles fonctionnalités dans le protocole IPSec.

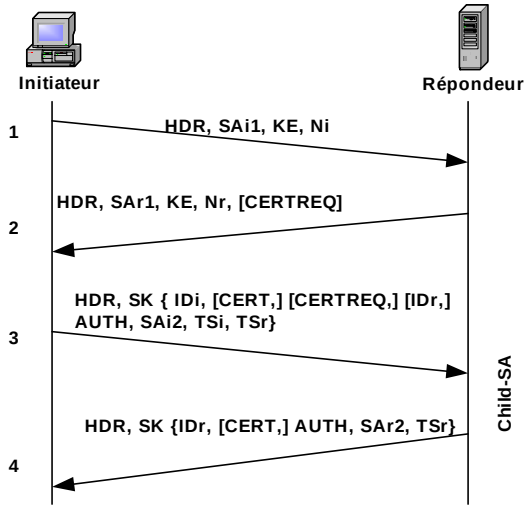


Figure 3-17. L'échange IKEv2

IKEv2 est semblable à IKEv1 dans l'authentification mutuelle des entités et l'établissement des associations de sécurité ISAKMP. En outre, IKEv2 est la seule proposition (en le comparant avec SIGMA et JFK) qui a été conçue pour être extensible. D'une façon simple, IKEv2 a proposé d'adapter une fonction de hachage au-dessus de tous les blocs. Ceci à une implication claire à l'extensibilité des futurs blocs IKEv2 (dans JFK, il est très difficile d'ajouter un futur bloc et de le mettre sous la partie chiffrée ou non chiffrée). La figure 3-17 montre un échange typique d'IKEv2.

Dans le premier échange (figure 3-17), l'initiateur envoie une liste d'algorithmes cryptographiques proposés dans le bloc *SA*, sa valeur publique *DH* et un nombre aléatoire *Ni*. Le répondeur répondra en introduisant dans son bloc *SA*, sa suite admise d'algorithmes, accomplit l'échange *DH* avec le bloc *KE* et envoie son nombre aléatoire *Nr*. A ce stade, les deux entités commencent à produire la clé partagée *SKEYSEED* et ses clés dérivées *SK_e, SK_a* et *SK_d*.
Maintenant, tous les messages dans la deuxième phase (message 3 et 4 sans le bloc *HDR*) sont chiffrés en utilisant la clé de chiffrage *SK*. L'initiateur peut maintenant envoyer son identité avec le bloc d'identification (*IDi*) et un hachage sur les premiers messages en utilisant le bloc *AUTH*. L'initiateur peut sur option, envoyer son certificat qui contient sa clé publique pour prouver sa vraie identité. L'initiateur peut également envoyer une demande de certificat et d'identité du répondeur qui peut accueillir multiples services. Le deuxième échange contient également le bloc *SA2* qui peut servir à la négociation d'une *Shild-SA* et le bloc *TS* (*Traffic Selector*) qui permettent la mise en application de politiques de sécurité. Dans le dernier message, le répondeur affirme son identité avec le bloc *IDr* et envoie sur option son certificat qui contient sa clé publique. L'initiateur hache aussi les 3 messages et les inclut dans le bloc *AUTH* pour assurer l'intégrité de tout l'échange IKEv2 et complète la négociation du Shild-SA.

	IKEv1	IKEv2
Nombre des méthodes d'authentification	8	2
Nombre des modes Phase1	main et agressive	1 : anonymat
Sélection du trafic	plages d'adresses	+ protocoles et ports
Duré de vie	Négociée	choix en local
Clés Phase1	même dans les deux sens, dépendent du mode authentification	différentes pour chaque sens ; ne dépendent pas du mode authentification.
Clé Phase2	groupe clés par protocole et par SPI	utilisation de la chaîne pseudo aléatoire pour le tirage de clés
Proposition d'une plage d'adresses	si la plage ne convient pas, rejet de la proposition	possibilité de proposer une sous plage

Tableau 3-8. Tableau de comparaison entre IKEv1 et IKEv2

3.6.7 Analyse des successeurs

Dans la section précédente, nous avons décrit les trois protocoles d'échange de clés proposés dans le cadre du groupe de travail IPSec à l'IETF. Bienque les trois propositions (JFK, IKEv2, et SIGMA) visent la simplicité en tant qu'objectif principal, le coût et le besoin réels pour développer et pour mettre en application ces nouveaux protocoles restent un facteur important pour tous future déploiement.

Le premier objectif du consortium IPSec était la comptabilité et la réutilisation du code de la solution existante. Ceci n'a pas favorisé le protocole JFK à être la prochaine norme d'IPSec. H. Krawczyk [Can02] a également noté d'autres inconvénients de JFK comme : un mécanisme d'échange des clés non sécurisé, forçant le service de non répudiation entre les deux acteurs et la longue entrée du fonction HMAC.

D'autre part, le protocole SIGMA est extrêmement compatible avec IKEv1. Il a montré que c'est un protocole sécurisé dans sa conception cryptographique. La différence principale avec IKEv2 est que le protocole SIGMA a proposé un échange monophasé. Comme décrit dans différents documents [Perl01] [Hoff02], les deux phases d'ISAKMP et IKEv1 ont été prouvées utiles et pour cette raison cette option (négociation en deux phases) est maintenue dans la nouvelle proposition. En conclusion, le geroupe de travail d'IPSec a préféré combiner les avantages de ces propositions sous le protocole IKEv2 pour être le nouveau successeur de IKEv1.

	IKEv2	SIGMA	JFK
Avantages	Protection d'identité	Protection d'identité	Protection d'identité
	Simplifie le protocole IKEv1 en enlevant les scénarios non utilisés.	% ISAKMP : utilisation du code IKE.	Service de non répudiation.
	% ISAKMP IKE : re-utilisation du code.	Supporte optionnellement un mécanisme contre les attaques de type DOS.	Implémenté en java avec 2000 lignes de code.
		Diminue le nombre de messages échangés pour établir un canal sécurisé.	
Inconvénients	Grande charge du développement dans le cadre de IPSec.	Protocole non extensible.	Attaques de type DOS sur ce protocole à cause des longs messages envoyés entre les deux communiquants
		Intégre en partie dans IKEV2, aucun avancement actuel.	Intégreren partie dans IKEV2, aucun avancement actuel.

Tableau 3-9. Tableau de comparaison entre les trois successeurs de IKEv1

A travers notre contribution dans le mailing liste de IPSec à l'IETF, nous avons remarqué que le groupe de travail d'IPSec considère le protocole ISAKMP comme un protocole excessivement généralisé, dur à évaluer et à comprendre. Cependant, le successeur de IKEv1 sera totalement basé sur le protocole ISAKMP avec moins de fonctionnalités et de scénarios inutilisés dans IKEv1. Néanmoins, ISAKMP a été conçu la première fois pour soutenir la négociation des associations de sécurité (SA) pour des protocoles de sécurité à toutes les couches de la pile de réseau (par exemple IPSec et SSL). A la différence des autres propositions qui traitent surtout les particularités d'IPSec (les solutions de IPSec avec les réseaux NAT et DHCP), ISAKMP reste dans sa conception un protocole générique qui peut être adapté à plusieurs environnements et protocoles. Une méthode d'unifier toutes ces propositions et de mettre à l'épreuve ISAKMP sera l'expérimentation d'ISAKMP avec un nouveau protocole de sécurité. C'est le but de notre première proposition.

3.7 Comparaison entre les différentes solutions

Dans cette section, nous comparons les trois protocoles de sécurité étudiés dans ce chapitre, par rapport aux exigences définies dans le chapitre 1. Les protocoles ISAKMP et IKE sont étudiés ici comme une partie du protocole IPSec

1. *Le choix des mécanismes cryptographiques*

   Les trois protocoles de sécurité se ressemblent par leur phase d'initialisation et leur mécanisme de protection de données.

   Pour la phase d'initialisation des protocoles IPSec (sous entendu, ISAKMP) et SSH, elles sont basées sur la négociation d'un groupe DH pour la génération des clés partagées. Le protocole SSL/TLS, même s'il supporte un échange DH, la plupart de ses négociations sont basées sur le chiffrement asymétrique avec la clé publique du serveur.

   Pour la protection de données, les trois protocoles utilisent des fonctions de chiffrement symétrique et de hachage pour assurer les services de confidentialité et d'intégrité de données. Pour de nombreuses raisons (politiques, ancienne spécification, etc...), ces trois protocoles commencent la négociation avec des algorithmes à faible coût cryptographique, tels que le chiffrement par DES (56 bits) et le hachage par MD5.

2. *L'authentification*

   L'authentification est un service qui diffère suivant le niveau de protection que nous souhaitons établir.

   Avec IPSec, l'authentification des utilisateurs est surtout liée aux matériels réseaux tels que les machines utilisateurs et les routeurs. ISAKMP propose plusieurs méthodes d'authentification telles que les clés partagées, les certificats X.509, etc.

   Les deux protocoles SSH et SSL/TLS sont des protocoles d'échange. Ils authentifient seulement les deux bouts de la communication (client, serveur). Pour le protocole SSL/TLS, Il propose une seule méthode d'authentification basée sur les certificats d'identité X.509. Cependant, SSH donne aux clients un choix entre plusieurs méthodes d'authentification négociées à travers un tunnel sécurisé. Pour cela, même les méthodes d'authentifications qui paraissent pour la première fois non sécurisées (mot de passe), peuvent être utilisées en toute sécurité avec SSH.

3. *Intégrité et confidentialité*

   Ces deux services sont à la base de tout protocole de sécurité. A la différence des deux autres protocoles (SSL/TLS et SSH), l'intégrité et la confidentialité dans IPSec sont

négociées. Elles peuvent être aussi utilisées conjointement et/ou séparément suivant le mécanisme de protection négocié (AH ou ESP).
Avec SSL/TLS et SSH, les services d'intégrité et de confidentialité sont explicites. Les messages de la phase d'initialisation de ces deux protocoles sont protégés en intégrité et les données des applications sont chiffrées et protégées en intégrité.

4. La protection contre les attaques actives et passives

Les trois protocoles étudiés dans ce chapitre proposent des mécanismes de protection pour les phases d'initialisation et les phases de protection de données.

- Si le service d'intégrité est déclenché (cas de SSH, SSL, IPSec AH), le trafic est protégé contre tout changement effectué par un tiers malveillant.
- Si le service de confidentialité est déclenché (cas de SSH, SSL, IPSec ESP), le trafic est protégé contre l'attaque par espionnage d'information et analyse du trafic.
- Si le service de non rejeu est déclenché (cas de SSH, SSL, IPSec- ESP, AH, ISAKMP), les données envoyées ne peuvent pas être rejouées après un certain temps.
- Si le service de protection contre le déni de service est activé (cas de IPSec- AH, ESP, ISAKMP), le trafic est protégé contre le « IP, TCP et UDP Floodings ».

5. La non répudiation

Parmi les protocoles étudiés dans ce chapitre, aucun d'entre eux n'assure une preuve de non répudiation sur les données échangées. Ce service, jugé très lourd cryptographiquement, est souvent laissé aux applications. Cependant, le fait d'utiliser les certificats accompagnés d'une signature dans la phase d'initialisation des protocoles ISAKMP et SSL/TLS, donne au deux différents acteurs une preuve de communication.

6. La protection d'identité

Les protocoles SSH et IPSec (ISAKMP) assurent le service de protection d'identité tandis qu'avec SSH, l'identité du client (identité/mot de passe ou clé publique) est protégée. ISAKMP protège l'identité des deux communicateurs. Notons que l'échange de protection d'identité d'ISAKMP protège les certificats et les clés publiques des communicateurs ainsi que d'autres types d'informations tels que les adresses IP réelles, et les emails.

7. La protection d'échange ou la protection du réseau

Parmi les trois protocoles étudiés dans ce chapitre, IPSec est le seul protocole qui offre une sécurité du trafic au niveau IP. Il peut assurer de ce fait, la sécurisation de toutes les applications basées sur IP. Les protocoles SSH et SSL/TLS assurent une sécurité de tout type de trafic circulant au dessus du protocole de transport TCP.

8. La confiance entre les communicateurs

La confiance entre les communicateurs est obtenue avec les méthodes d'authentification négociées entre les acteurs. Les deux protocoles IPSec et SSL/TLS utilisent les autorités de confiance (PKI) pour fournir un service d'authentification forte basée sur les certificats d'identité X.509. Pour le protocole SSH, plusieurs travaux sont actuellement en cours pour intégrer son mécanisme d'authentification basé sur les clés publiques/privées RSA dans une infrastructure de confiance [Verd03].

9. L'autorisation

Puisque IPSec est un protocole de couche réseau, le service d'autorisation des utilisateurs est souvent laissé aux Firewalls et aux listes de contrôles ACL. Le protocole SSL/TLS ne possède non plus cette fonctionnalité qui est plutôt laissée aux applications.
SSH définit des listes de contrôle statique basées le système d'exploitation Unix en classant les utilisateurs dans des groupes suivant leurs privilèges d'accès.

10. La gestion des clés

Chacun des trois protocoles de sécurité étudiés dans ce chapitre, intègre son propre mécanisme de gestion des clés. Cependant, les trois protocoles possèdent tous les mêmes principes de chiffrement asymétrique et d'échange Diffie-Hellman [Diff76]. C'est avec ce dernier que les trois protocoles assurent le service de protection à long terme des données ou ce qu'on appelle le service du secret parfait (*Perfect Forward Secrecy*).

11. Les VPNs

Par rapport aux trois solutions, le protocole IPSec reste jusqu'à maintenant la solution la plus répandue pour la mise en œuvre des VPNs. Toutefois, IPSec apporte aux deux solutions SSL VPN et SSH VPN un complément de sécurité paramétrable afin de satisfaire l'exigence globale de protection sur le transfert de données à réaliser.

12. Le contrôle d'accès

Actuellement, la plupart des protocoles de sécurité utilisent à la fois IPSec et la technique des Firewalls. En effet, IPSec et les Firewalls ne fournissent pas les mêmes services de contrôle d'accès et de filtrage. Par exemple, les Firewalls permettent de contrôler le trafic sortant d'un site de façon globale. Autrement dit, il est possible d'avoir des règles s'appliquant sur l'ensemble des paquets transitant par le Firewall, quelque soit la source ou la destination. Contrairement à IPSec qui définit précisément les deux protagonistes de la communication avant de créer une SA. En outre, un Firewall avec IPSec peut agir au niveau applicatif (au niveau des requêtes http par exemple).

Le protocole SSL/TLS définit dans le RFC 2246 est un protocole de sécurité de bout en bout qui interdit tout contrôle d'accès hors de la machine serveur. En plus, les requêtes des applications (comme le cas des en-têtes HTTP ou SMTP) passent à travers un tunnel SSL/TLS chiffré. C'est principalement pour ces deux raisons que les caches Proxy sont désactivés lorsque le protocole SSL/TLS est utilisé. Ils sont alors incapables de jouer leur rôle de contrôle d'accès ou de filtrage.

Le mécanisme de contrôle d'accès avec SSH est basé sur la définition des listes de contrôle statique basées sur l'environnement Unix en classant les utilisateurs dans des groupes suivant leur privilèges d'accès. Il possède comme avantage un système d'authentification unique ou SSO (Single Sign On). En effet, le serveur *SSH* permet d'installer un agent d'authentification en frontal d'un ensemble d'applications. L'agent *SSH* intercepte les accès utilisateurs. Une fois l'utilisateur identifié, il transmet les données d'identification à l'application via des variables d'environnement (dans le cas d'un module du serveur Web) ou dans des champs d'en-têtes HTTP.

13. La gestion de délégation entre acteurs

Les trois protocoles étudiés dans ce chapitre n'assurent pas le service de gestion de délégation entre acteurs.

Le protocole SSH présente l'entité intermédiaire comme une entité de confiance. Un serveur SSH peut jouer le rôle d'un serveur intermédiaire en faisant un double tunnel. Le premier est situé entre le client et le serveur SSH. Quant au deuxième, il est situé entre le serveur SSH et le serveur destinataire.

Avec le protocole SSL/TLS, la version standard ne permet pas la délégation du rôle d'authentification. C'est pour cette raison que les Proxy sont désactivés dans le cas d'utilisation de SSL/TLS entre les deux bouts de communications. [Jack01] et [GFD.17] ont mené un travail dans ce domaine afin d'intégrer un mécanisme de délégation des rôles entre les serveurs SSL/TLS et les Proxy intermédiaires.

Quant au protocole IPSec (ISAKMP et IKEv1), il n'intègre pas ce service. En effet, IPSec diffère dans la méthode d'authentification *hop-by-hop* qui établit une série de tunnels sécurisés et dans la méthode *end-to-end* qui établit un seul tunnel entre les deux bouts de la communication.

14. La vérification formelle des protocoles de sécurité

Le modèle que nous proposons (LEVA) vérifie la propriété de secret et d'authenticité de la phase d'initialisation des trois protocoles de sécurité (IPSec, SSL/TLS et SSH).

Avec LEVA, nous avons réécrit ces trois protocoles (Annexe B) puis nous avons utilisé l'outil *Hermes* pour la vérification.

Les résultats de vérification (annexe B, [Huss05]) montrent que les protocoles SSL/TLS, SSH, IKEv1 et IKEv2 satisfont la propriété de *secret*. Ainsi, ils sont cryptographiquement sécurisés.

Néanmoins, la vérification des protocoles se heurte à deux problèmes. Le premier est dû à une grande diversité de modèles. En effet, chaque auteur a son propre modèle favori qui utilise souvent des constructions très particulières. Ces différents modèles sont le plus souvent incomparables du point de vue de l'expressivité. Certains permettent les clés composées, d'autres ne les permettent pas. D'autres modèles permettent le chiffrement asymétrique, etc. De plus, les propriétés des protocoles sont dépendantes du modèle, voire même du protocole à vérifier.

Le deuxième problème vient du fait que cette vérification est limitée aux spécifications théoriques des protocoles et ne vérifie pas le protocole lui-même tel qu'il est implémenté (problème de mémoire, lourdes opérations cryptographiques, attaques de déni de service, etc.).

15. L'extensibilité du protocole

Les trois protocoles de sécurité étudiés dans ce chapitre ont été définis afin d'assurer les services de confidentialité et d'intégrité des informations échangées sur un réseau public.

Depuis, les applications sont devenues plus exigentes et requirent de nouveaux services de sécurité tels que l'anonymat, la non répudiation, etc.

Un protocole extensible permet d'assurer la négociation de ces nouveaux services sans avoir à changer sa spécification. Avec le protocole SSL/TLS, le RFC 3546 [RFC3546] essaye de satisfaire cette exigence en définissant la structure de deux messages génériques échanges au début de la connexion SSL/TLS afin de négocier tout type de service. Dans le groupe de travail d'IPSec, le protocole IKEv2 ne définit pas l'extensibilité comme un

exigence fondamentale. Son travail est plutôt de garder l'interopérabilité avec la version standard dans le cadre du protocole IPSec.

3.8 Conclusion

Dans ce chapitre, nous avons traité les trois solutions de sécurité les plus répandues actuellement à savoir les protocoles SSH, SSL/TLS et IPSec, tout en expliquant les avantages et les inconvénients de chacune de ces solutions.

Pour le protocole SSH, même s'il a pu contourner de nombreuses menaces à la sécurité liées au réseau, il reste rattaché dans sa spécification à une conception logicielle plus qu'un vrai protocole de sécurité. Ceci a rendu ce protocole non applicable dans les nouveaux environnements et ainsi incapable de satisfaire les nouveaux besoins des applications et des utilisateurs.

Le protocole SSL/TLS est actuellement le seul protocole de sécurisation déployé à grande échelle car il se trouve déjà sur la plupart des plates-formes sécurisées de la toile, aussi bien au niveau des serveurs qu'au niveau des clients. En effet, ce protocole a été adapté au delà des applications transactionnelles habituelles. Ainsi, il est utilisé par le consortium WAP (*Wireless Application Protocol*) pour les communications radiophoniques du type GSM, par le protocole EAP pour les environnements sans fil (*WI-FI 802.11*) [RFC2719] et même dans des approches destinées aux cartes à puce avec une version du protocole à disposition des usagers itinérants. L'architecture modulaire de SSL/TLS permet de faire évoluer certaines parties du protocole sans remettre en cause l'ensemble de l'édifice.

Le protocole IPSec et ses sous protocoles associés fournissent aujourd'hui une solution standard et sûre pour la réalisation des réseaux privés sécurisés. IPSec offre également aux autres protocoles un complément de sécurité afin de satisfaire l'exigence globale de protection sur le transfert de données à réaliser.

Le protocole ISAKMP fournit une méthode sécurisée et générique pour la négociation des associations de sécurité et la distribution de clés y compris sur des lignes non sécurisées. Si l'utilisation actuelle de ce protocole se limite au cadre du protocole IPSec, de nouvelles propositions apparaissent actuellement pour l'utilisation de ce protocole dans des nouveaux environnements tels que le domaine du multicast, de la signalisation UMTS [Arkk02], et même dans les réseaux WI-FI. Dans le chapitre suivant, nous mettons à l'épreuve ce protocole dans le cadre de son utilisation avec SSL/TLS.

Enfin, nous terminons ce chapitre par une table de comparaison des trois solutions de sécurité. Nous présentons les critères de comparaison suivants : service de base, services optionnels, méthode d'authentification, confidentialités et chiffrement de données, niveau de protection, facilité de gestion et de déploiement, accès aux ressources, gestions des clés, contrôles d'accès, mécanisme de filtrage et performance cryptographique.

	IPSEC	SSL/TLS	SSH
Type de spécification	IETF (IPSEC WG), Standardisé	IETF (TLS WG), Standardisé	IETF (SSH WG), Draft
Service de base	Confidentialité Intégrité Authentification Non rejeu	Confidentialité Intégrité Authentification Non rejeu	Confidentialité Intégrité Authentification Protection d'identité Non rejeu
Service Optionnel	Protection d'identité		Autorisation, (c.a.d, contrôle d'accès aux comptes). Tunnel sécurisé. Le 'port forwarding'. Le service de SSO (single Sign On)
Méthode d'authentification	Radius, Token ID, des clés partagées ou des certificats X.509	Certificat X.509 et Kerberos	Clé publique RSA ou DSA [DSS], Mot de passe et Kerberos.
Confidentialité et chiffrement de données	40 ou 128 bits RC4. 56-bit DES; 112 ou 168 bits 3DES. 128, 192 ou 256 bits AES. Le niveau de chiffrement est placé par accord entre le client et le serveur IPSEC	Les mêmes algorithmes de chiffrement que IPSec, mais souvent négociés au plus bas dénominateur commun : 40-bit RC4.	Les mêmes algorithmes de chiffrement que IPSec.
Niveau de protection	Accès à toutes les applications et ressources au-dessus de IP : Email, partage fichier, base de données, terminal utilisateur.	Accès aux applications qui supportent le protocole SSL (surtout celles avec un interfaçage Web)	Puisque SSH permet le '*tunneling'* et le '*port forwarding'* l'utilisateur pourra accéder à un grand nombre d'applications au-dessus de TCP.
Facilité de gestion et de déploiement	Plus complexe que le protocole SSL, dû à la condition d'installer et de configurer le logiciel client, quelques matériels traversant les réseaux (routeurs) et les pare-feu	Aucun besoin de distribuer, d'installer, et de configurer le logiciel client. Gestion simple des utilisateurs en leur donnant des comptes Web (nom et mot de passe) et l'URL du serveur SSL	Plus complexe que SSL, puisque les utilisateurs doivent installer et configurer leur logiciel client et même générer une paire des clés RSA ou DSA.
Accès aux ressources	Négociation des ressources dans la deuxième phase de IKE	Dans SSL/TLS, négociation des ressources après l'ouverture d'une session SSH. Dans TLS Extension, les deux entités peuvent négocier les ressources dans leur premier échange	Négociation après l'ouverture d'un tunnel SSH sécurisé
Protection des données	Durant leurs transactions	Durant leurs transactions	Durant leurs transactions
Gestion des clés	Dépend de deux protocoles externes IKE et ISAKMP (basée sur DH ou un chiffrement asymétrique).	SSL utilise une méthode de chiffrement asymétrique (RSA ou DSA) ou des clés DH pour la création d'un secret partagé	Gestion des clés à travers un échange Diffie-Hellman
Contrôle d'accès	Plus adapté pour l'accès à un ensemble défini des utilisateurs. Accès total pour les clients VPN à un segment d'un réseau. Solution idéale pour la sécurisation au niveau réseau. Problème de complexité.	Plus adapté pour l'accès sur demande des utilisateurs. Sécurise tout le trafic du navigateur au serveur principal. Accès " oui non " au niveau d'application basé sur une liste d'accès statiques (ACL). Solution limitée aux applications qui supportent une interface Web ou qui intègrent SSL/TLS.	Comme le cas de SSL, plus adapté pour l'accès sur demande des utilisateurs. Sécurise tout le trafic entre le client et le serveur SSH. Contrôle d'accès au niveau serveur SSH basé sur un fichier de configuration statique. Solution idéale pour les environnements Unix. Problème de gestion des clés chez les utilisateurs et un manque d'interface complète
Mécanisme de filtrage	Par paquet (niveau réseau). Lié aux politiques SAD et SPD de IPSEC.	Impossible après l'ouverture d'une session SSL	Impossible après l'ouverture d'un tunnel SSH
Performance cryptographique	Peut être lente à cause d'un nombre important d'opérations cryptographiques et de gestion des clés	50 fois plus lente qu'une connexion http. Nécessite des accélérateurs Web, des caches Web pour améliorer sa performance	lente à cause du grand nombre d'échanges pour l'établissement d'un canal sécurisé

Deuxième Partie

Extensions des solutions de sécurité existantes

Chapitre 4

4 Intégration d'ISAKMP dans SSL/TLS

4.1 Résumé

Dans ce chapitre, nous intégrons le protocole d'authentification et de négociation des associations de sécurité ISAKMP dans le protocole SSL/TLS et nous montrons les nouvelles perspectives de services basées sur cette approche.
Adopter ISAKMP comme seul protocole de gestion de clés permettra sans aucun doute de réaliser des optimisations intéressantes. Nous montrons la capacité d'ISAKMP à supporter différents types de protocoles et également la possibilité de partager un même canal sécurisé par plusieurs protocoles. Au-delà des optimisations, ISAKMP est nettement plus flexible que le handshake de SSL/TLS.
Ce travail a donné lieu à plusieurs publications [Hajj3a], [Hajj3c] et [Hajj3e].

4.2 Introduction

Actuellement le protocole SSL/TLS (Secure Socket Layer/Transport Layer Secure) est le protocole de sécurisation le plus déployé. Ceci est dû en grande partie à son intégration coté client dans les principaux navigateurs web.
SSL est une couche composée de quatre protocoles modulaires et indépendants dont le protocole Handshake qui a pour charge l'authentification des parties en jeu et la génération des secrets nécessaires pour la mise en œuvre des services de sécurité tels que l'intégrité et la confidentialité.
Grâce à cette nature modulaire, les extensions futures de SSL/TLS sont faciles à intégrer. La cible de ces extensions est en grande partie le protocole Handshake [Tayl04] [Jaco03]. Ce dernier présente néanmoins quelques limitations quant à la faiblesse du protocole Handshake et à l'absence des mécanismes d'autorisation [Klima03] [Apos99] [Shac01] [Shac02].

Plutôt que de modifier tout le protocole SSL/TLS, nous nous sommes focalisés sur une étude pour étendre ce protocole. Nous proposons pour cela l'intégration du protocole ISAKMP dans la phase d'ouverture de la session SSL/TLS. En effet, ISAKMP couvre des besoins très larges. A la différence des autres protocoles d'authentification et d'échange des clés existantes, ISAKMP a été conçu pour être générique et pour supporter tout type de protocole de sécurité.
Afin d'unifier tous les mécanismes de gestion des clés, il est utile d'expérimenter ISAKMP à différents niveaux. Ceci d'autant plus que ce protocole est ouvert pour supporter des services évolutifs tels que les services d'authentification, de partage de secrets et d'autorisation basée sur les certificats d'attributs. D'où l'objectif de notre contribution.

Dans ce chapitre, nous présentons les motivations pour intégrer ISAKMP comme protocole d'échanges des clés et d'authentification dans SSL/TLS. Puis, nous présentons le fonctionnement de notre architecture ainsi que le prolongement du DOI (Domain of Interpretation) existant pour supporter le protocole SSL/TLS.

Pour conclure, nous proposons une analyse de cette solution et ses perspectives notamment en terme d'expérimentation et de futur déploiement.

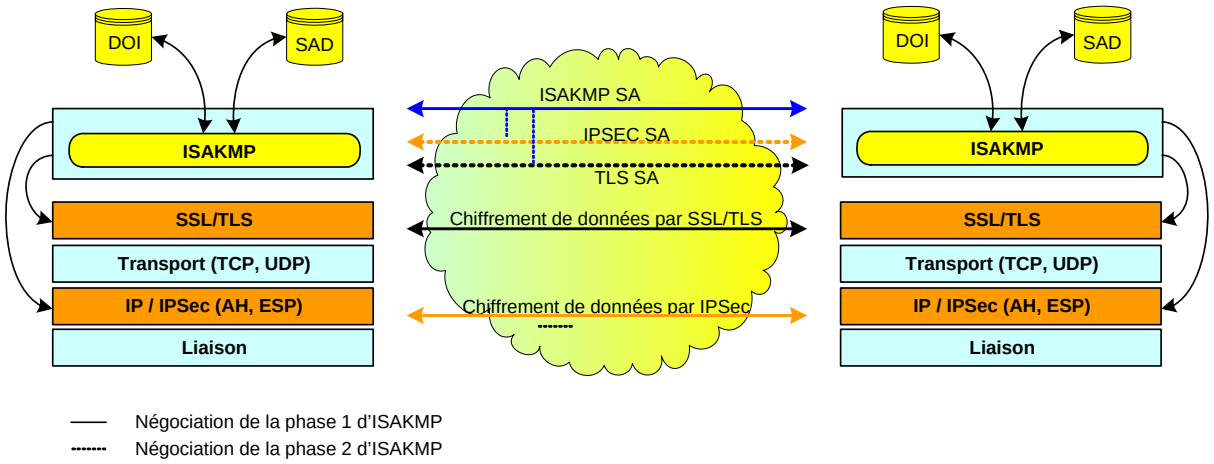


Figure 4-1. Négociation pour plusieurs protocoles de sécurité avec ISAKMP

4.3 Les motivations

Dans le monde des réseaux publics, plusieurs solutions ont été déployées pour la sécurisation des échanges (protocole de sécurité au niveau application, transport, réseau ou même physique) dont chacune a défini ses propres règles de gestion des clés. A la différence des solutions existantes, ISAKMP fournit un cadre générique pour la négociation des associations de sécurité (SA) pour des protocoles de sécurité à toutes les couches de la pile réseau (par exemple, IPSec, TLS, ...) (figure 4-1). Une façon d'unifier toutes ces propositions est d'expérimenter le protocole ISAKMP avec d'autres protocoles de sécurité comme SSL/TLS.

Nos motivations pour adapter ISAKMP au protocole SSL/TLS sont les suivantes :

1. *Support pour la protection d'identité des deux communiquant*
   Comme nous l'avons expliqué dans le premier chapitre, l'accès illicite aux informations concernant les identités des communicateurs constitue non seulement une atteinte à la vie privée des utilisateurs, mais aussi une véritable attaque sur l'infrastructure réseau. En effet, l'identification permet à l'ennemi de connaître des informations telles que le nom de l'utilisateur, le moment ou il accède au réseau, les services qu'il utilise, etc. Ce qui aboutit à une véritable attaque par analyse de trafic. Avec SSL/TLS, ce service concerne surtout la protection de l'identité du client (l'initiateur) puisqu'un serveur SSL/TLS possède toujours une identité publique.

2. *Interopérabilité entre les protocoles de sécurité*
   La première motivation derrière la conception du protocole ISAKMP par le NSA (Nationale Security Agency) était le développement d'un protocole de gestion des clés robuste qui peut être utilisé avec plusieurs protocoles de sécurité (par exemple TLS,

IPSec,…). Ces protocoles pourraient en résultat partager le même code de gestion des clés suivant leur besoin en sécurité. Ceci permet alors :

- De limiter les fonctionnalités redondantes au sein de chaque protocole de sécurité (authentification, intégrité, etc.).
- De simplifier le transfert d'un protocole à un autre et réduire les temps d'établissement d'un canal sécurisé par chaque protocole. Chaque protocole définit seulement son échange en phase 2.
- De permettre une meilleure gestion de l'ensemble des sessions établies par les différents protocoles à différents niveaux de la pile OSI.

3. Échange de plusieurs méthodes d'authentification sous une même SA

Sous la même association de sécurité ISAKMP, les deux entités peuvent établir plusieurs échanges en phase 2 avec différentes méthodes d'authentification. Ceci permet de négocier de nouvelles méthodes d'authentification sans avoir besoin de répéter les lourdes opérations cryptographiques effectuées durant la première phase d'ISAKMP (surtout la génération et l'échange des valeurs publiques DH).

4. Support des certificats d'attributs

Parmi sept types (PGP, X.509, SPKI, etc.) de certificat définis dans le RFC 2408, ISAKMP est le premier protocole d'authentification qui intègre dans sa spécification les certificats d'attribut.

En effet, grâce à son mécanisme de protection d'identité, les certificats d'attribut ainsi que les autres types de certificats sont échangés sous une session sécurisée établie avant la phase d'authentification. Ainsi, nous nous intéressons à ce type de certificat afin d'assurer un service d'autorisation et de contrôle d'accès basé sur une approche de gestion de rôles.

5. Support des mécanismes évolués pour les clients à faibles ressources

Afin de diminuer le volume de données échangées entre les entités, le protocole ISAKMP propose un mécanisme de négociation des URLs des certificats d'identité et des certificats de confiance entre les acteurs.

En effet, il suffit que l'émetteur envoie une URL vers une base de données LDAP (Lightweight Directory Access Protocol) pointant sur son certificat ou sur les certificats de confiance. Ceci est important pour réduire le temps d'échange avec ISAKMP dans le cas où un émetteur a envoyé plusieurs requêtes demandant différents types de certificats d'identité et d'autorité de certification.

6. Flexibilité dans la définition des DOI

Avec ISAKMP, les différents paramètres de sécurité tels que les algorithmes de chiffrement, les algorithmes de signature, les fonctions de hachages et les tailles des clés sont définis à travers le DOI ou le domaine d'interprétation. L'utilisation d'ISAKMP avec un nouveau protocole de sécurité nécessite en premier temps une définition '*simple*' d'un nouveau domaine d'interprétation pour le protocole concerné. Pour SSL/TLS, nous définissons un nouveau DOI que nous appelons TLS DOI (§ 4.4.6).

7. Séparation des méthodes d'échange des clés de la gestion des associations de sécurité

La séparation entre les méthodes d'échange des clés et celles de la gestion des associations de sécurité permet une meilleure interopérabilité entre des systèmes

possédant différentes conditions de sécurité. Elle simplifie également l'analyse des différentes évaluations des serveurs ISAKMP.

8. Négociation de multiples associations de sécurité dans la phase 2 d'ISAKMP
Il est également possible avec ISAKMP pendant un simple échange en phase 2, de négocier des multiples associations de sécurité. Chacune de ces associations correspond à une application spécifique ayant son propre ensemble de clés.

4.4 Intégration d'ISAKMP dans SSL/TLS

4.4.1 Contraintes de base

Après la définition des principales motivations de notre architecture, nous citons trois contraintes inhérentes à sa conception globale. Les trois conditions pour réaliser un déploiement simple et efficace d'ISAKMP avec SSL/TLS sont :

1. Utilisation des échanges et des blocs standard ISAKMP
Afin de faciliter le déploiement de notre architecture et afin de minimiser les tâches de développement de nouveaux blocs ou échanges ISAKMP, nous avons essayé d'utiliser les blocs et les échanges ISAKMP existants. Avec SSL/TLS, nous nous intéressons surtout à l'échange de protection d'identité pour établir la première association de sécurité ISAKMP. Nous nous s'intéressons également à l'échange *Quick Mode* de Oakley pour établir la deuxième association de sécurité propre à SSL/TLS.

2. Transparence par rapport à la couche Record de SSL/TLS
Puisque le changement dans SSL/TLS est au niveau Handshake, les résultats d'échange (clé de chiffrement, d'authentification, …) doivent être transparents au protocole Record de SSL/TLS. C'est-à-dire, le mécanisme de dérivation des clés à partir de la clé matérielle générée avec ISAKMP doit ressembler à celui utilisé par le protocole Handshake de SSL/TLS.

3. Interopérabilité avec le protocole SSL/TLS existant
En plus de l'interopérabilité avec le protocole Record de SSL/TLS, notre approche doit permettre à des clients SSL/TLS qui supportent notre proposition et à des serveurs qui ne la supportent pas de se communiquer et vice versa. La standardisation au sein de l'IETF de 'TLS Extensions' nous a résolu ce problème. Nous utilisons ainsi ce nouveau standard (le RFC 3546) pour la négociation d'ISAKMP durant le premier échange de SSL/TLS.

4.4.2 Architecture

ISAKMP est un protocole de couche application placé au-dessus de la couche transport. Selon la spécification de ce protocole, il doit offrir ses services au dessus du protocole de transport UDP mais il peut être utilisé directement au-dessus de IP [RFC2408]. Actuellement et avec IKEv1, l'IANA [RFC3232] lui a attribué le port 500 d'UDP.

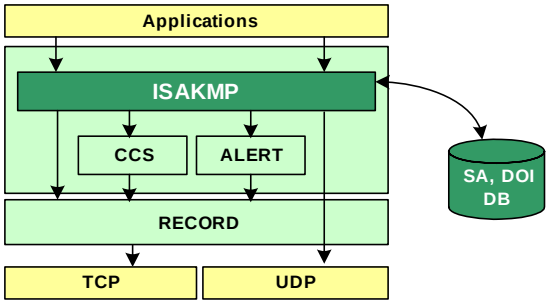


Figure 4-2. Intégration d'ISAKMP dans SSL/TLS

Avec SSL/TLS (figure 4-2), ISAKMP peut fonctionner toujours sur le port 500 d'UDP. En effet, à la fin des deux phases ISAKMP, chaque protocole de sécurité doit chiffrer et/ou authentifier son propre tunnel indépendamment du protocole ISAKMP. Par exemple, le protocole IPSec ESP chiffre les données sur le port 50 alors que le protocole IPSec AH authentifie les données sur le port 51.
L'établissement d'une session sécurisée SSL/TLS avec ISAKMP (figure 4-3) nécessite quatre étapes principales que nous détaillons dans la suite de ce chapitre.

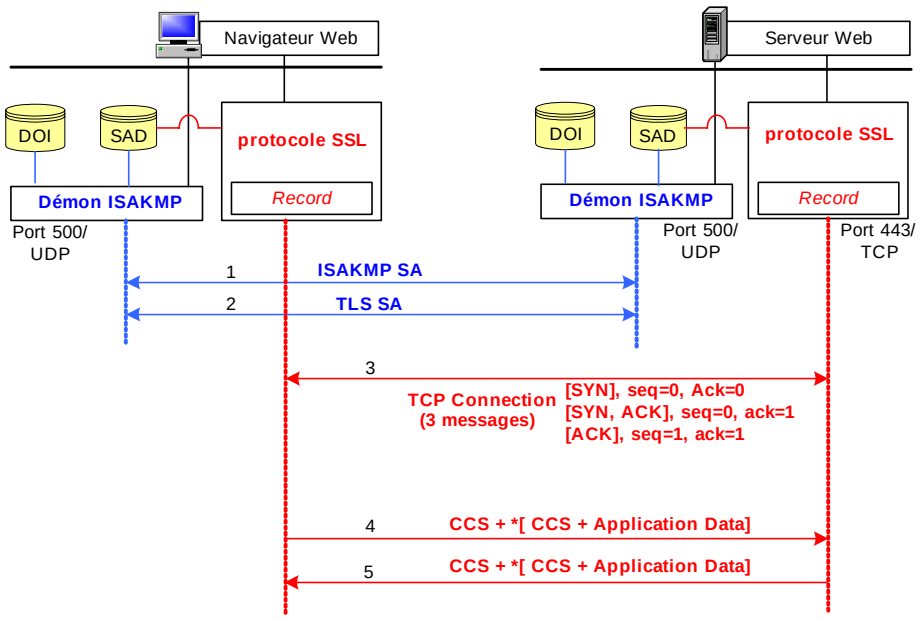


Figure 4-3. Etablissement des différentes sessions pour ISAKMP et SSL/TLS[7]

Les quatre étapes nécessaires pour établir une session SSL/TLS avec ISAKMP sont les suivantes :

1. *Négociation d'ISAKMP SA*
   Durant la première phase, un ensemble d'attributs relatifs à la sécurité est négocié. Les identités des tiers sont authentifiées et les clés sont générées. Ces éléments constituent une première association de sécurité que nous appelons ISAKMP SA.

2. *Négociation de TLS SA*

[7] *[B] signifie que le message B est chiffré symétriquement.

La seconde phase permet de négocier les paramètres de sécurité relatifs à une SA (Security Associations) pour le compte d'un protocole de sécurité donné. Dans notre cas, c'est le protocole SSL/TLS. Les échanges de cette phase sont sécurisés (confidentialité, authenticité...) grâce à l'association de sécurité déjà établie. Celle-ci peut être utilisée pour négocier plusieurs SA « fils » destinées à d'autres mécanismes de sécurité définis à travers des domaines d'interprétations (DOI).

3. Ouverture d'une session TCP

A ce point, ISAKMP a établi tous les paramètres de sécurité (algorithme de chiffrement, clés session, etc.) nécessaires au chiffrement de tout type de données (page Web, fichier, vidéo, etc.) avec la couche *Record* de SSL/TLS (figure 4-3). Maintenant, c'est à l'application qui souhaite utiliser SSL/TLS d'établir une première connexion TCP vers le port défini par chaque application. A titre d'exemple, pour un serveur Web sécurisé avec SSL/TLS, la connexion TCP est établie vers le port 443.

4. Echange des messages CCS et des données chiffrées

Dans la dernière partie (étape 4 et 5), les clés de chiffrement sont activées avec le module CCS (*ChangeCipherSpec*) de SSL/TLS. Les messages *CCS* qui ne font pas partie du protocole Handshake, sont envoyés au début du premier échange de données chiffrées. De plus, afin de protéger ces messages contre l'attaque homme du milieu, nous proposons de les ajouter dans le premier échange chiffré des données.

4.4.3 La négociation des associations de sécurité

Nous décrivons un ordre de messages échangés pendant les deux phases ISAKMP (étape 1 et 2 du paragraphe précédent) afin de fournir un exemple concret de l'intégration d'ISAKMP dans SSL/TLS. Les deux phases possèdent les caractéristiques suivantes :

- Dans la première phase, les deux communicateurs (client et serveur SSL/TLS) initient l'échange de protection d'identité. Cet échange est basé sur une authentification mutuelle forte avec des certificats X.509 obtenus à partir d'une autorité de certification valide. Pour le client, son authentification est plutôt liée au matériel qu'il utilise. Le résultat de cet échange est la génération d'une clé secrète (*SKEYID*) et un ensemble des clés dérivées (*SKEYID_d, SKEYID_a, SKEYID_e*) servant à la génération de nouvelles clés, à l'authentification et au chiffrement des messages de la phase 2.

- Dans la deuxième phase, les deux communicateurs établissent une association de sécurité pour SSL/TLS. L'échange que nous proposons diffère courtement de l'échange *Quick Mode* de Oakley. Nous l'appelons pour cela « *TLS Quick Mode* ». Cet échange permet aux deux entités :

  1. de renégocier leur méthode d'authentification

  Durant le deuxième échange, les deux entités (surtout le client) peuvent présenter de nouvelles identités, négocier leurs méthodes d'authentification et préciser l'adresse d'un serveur SSL/TLS (serveur virtuel ou réel) contenant les ressources demandées. Pour le serveur, sa ré-authentification est inutile sauf à la suite d'une demande de certificat envoyée par le client. Ceci dans le cas où le serveur demandé n'était pas authentifié durant la première phase.

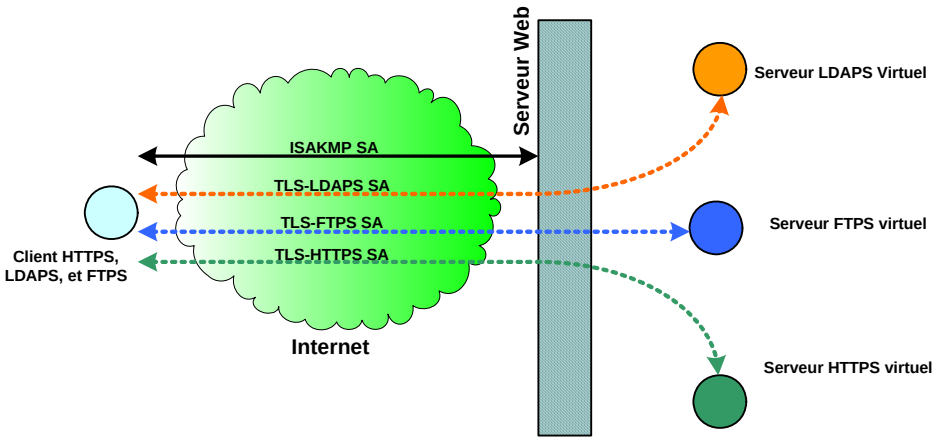


Figure 4-4. Négociation des associations de sécurité avec plusieurs serveurs virtuels

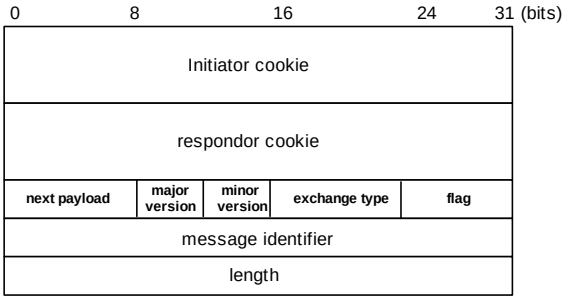
2. *de permettre au client SSL/TLS de rester anonyme*

Puisque dans la plupart des scénarios SSL/TLS le client est anonyme, nous avons introduit une nouvelle méthode d'authentification « *anonyme* » pour les clients. Elle requiert aucune modification ni au niveau des blocs ISAKMP, ni au niveau du mécanisme de génération des clés, mais seulement l'ajout d'un nouveau attribut «*anonymous_client*» dans le DOI de SSL/TLS.

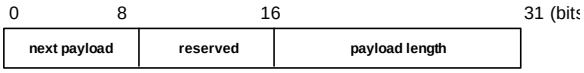
4.4.3.1 La négociation d'ISAKMP SA

Durant cette phase, les deux communicateurs établissent la première association de sécurité appelée ISAKMP SA (figure 4-7). Puisque nous utilisons l'échange de protection d'identité, les deux entités échangent six messages pour négocier leur stratégie de sécurité et pour protéger leur identité.

Tout les échanges commencent avec un en-tête ISAKMP (*HDR*) (figure 4-5a) suivi par un nombre variable de blocs.



a) Le bloc HDR d'ISAKMP



b) En-tête des blocs ISAKMP

Figure 4-5. Formats des en-têtes ISAKMP

Le bloc HDR contient deux cookies (cookie initiateur et cookie répondeur) générés suivant le mécanisme de Photuris [RFC2521] afin d'assurer une première protection contre les attaques de déni de service. Ils sont suivis par le type de l'échange (*type of exchange)* contenant la valeur *Identity protection*, un identificateur du message (*message ID*) qui reste à zéro pendant la phase 1, la version du protocole (major *version, 1, et minor version, 0*), un drapeau (*flag*) pour indiquer l'option spécifique de cet échange ISAKMP (ici a*uthentification*), la longueur du message (*length*) et le type du bloc suivant (*next payload*).

Pour les blocs de données ISAKMP, ils commencent par le même en-tête générique comme indiqué à la figure (4-5b). Le paramètre *next payload* a une valeur de 0 si c'est le dernier bloc dans le message. Autrement, sa valeur est celle du type du bloc suivant (par exemple

Proposal). Le paramètre *payload length* indique la longueur en octets de ce bloc, y compris l'en-tête générique.

Après le bloc HDR, les deux entités commencent à construire les messages de négociation de leur stratégie de sécurité. Le bloc SA est employé pour débuter l'établissement d'une association de sécurité. Dans le bloc SA, le paramètre domaine d'interprétation identifie le domaine sous lequel la négociation a lieu. Dans notre cas, le domaine d'interprétation est TLS DOI identifié avec le numéro TBD. Le paramètre *Situation* définit la politique de sécurité pour cette négociation et l'ensemble d'attributs qui y correspond. Le [RFC2407] définit trois cas (ou situations) suivants : S*IT_IDENTITY_ONLY, SIT_SECREC et SIT_INTEGRITY*. Le premier identifie l'association de sécurité par rapport aux identités, le deuxième assure la confidentialité des messages et le troisième est utilisé si le service d'intégrité est demandé. Avec SSL/TLS, nous n'utilisons que le paramètre S*IT_IDENTITY_ONLY* qui restera le seul mandataire dans notre nouveau domaine d'interprétation.

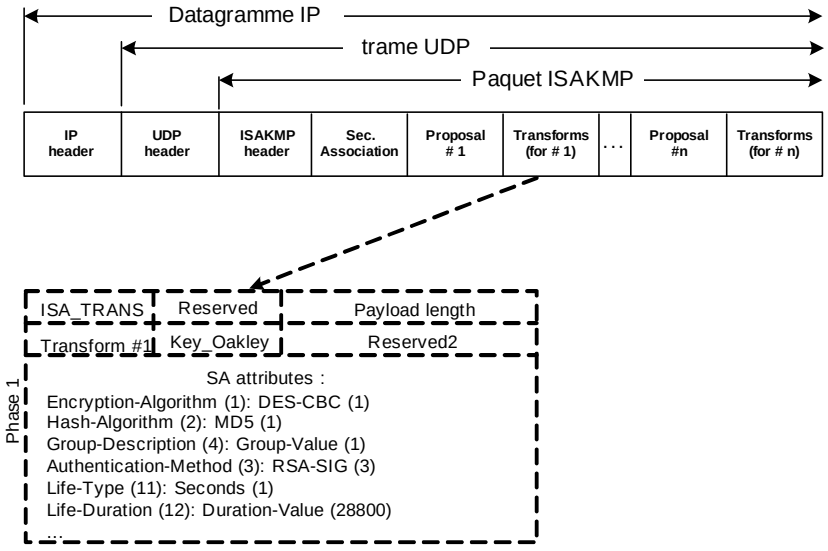


Figure 4-6. Structure d'une proposition ISAKMP (phase 1)

Le bloc SA est toujours suivi par un ou plusieurs bloc *Proposal* (figure 4-6). Ce dernier est à son tour suivi par un ou plusieurs blocs *Transform*. Le bloc *Proposal* indique le protocole choisi pour cette SA (par exemple le protocole TLS) et pour lequel les services et les procédures sont négociés. Il inclut également l'index de paramètres de sécurité (*SPI*) émettrice et le nombre de transformations (le *SPI* reste à zéro jusqu'à que les deux entités fixent leur cookies). Les blocs *Transfom* permettent à l'initiateur d'offrir plusieurs possibilités que le répondeur devra choisir ou rejeter. Ces blocs contiennent essentiellement les algorithmes et les fonctions utilisés pour assurer la confidentialité, l'authentification et l'intégrité des entités et des données (par exemple, 3DES pour le chiffrement, HMAC-SHA-1 pour le hachage) avec les attributs associés (par exemple, la longueur de hachage, le temps de vie du SA, etc.)

Dans le deuxième échange, l'initiateur et le répondeur échangent en clair des nombres aléatoires (*NONCEi et NONCEr*) et des valeurs publiques DH (g^i *et* g^r) à travers les blocs *NONCE* et *KE* respectivement. Ceci permet à chaque entité de disposer des informations nécessaires pour générer la clé secrète partagée (*SKEYID*) ainsi que l'ensemble des clés dérivées (§ 4.4.3.3).

Puisque l'authentification est réalisée par des certificats d'identité, les deux entités échangent aussi le bloc *Certificate Request* (ou *CERT-REQ*) contenant une liste d'autorités de certification valide. En effet, cette demande de certificat est toujours acceptée durant les phases d'échanges ISAKMP. Le répondeur doit, à la réception de ce bloc, envoyer un certificat d'identité qui correspond aux autorités de certification choisies par l'initiateur. En plus, la spécification d'ISAKMP permet d'envoyer plusieurs demandes de certificat. Ceci est obtenu tout simplement, en envoyant plusieurs fois le bloc *CERT_REQ*.
Dans le dernier échange, tous les messages à l'exception des en-têtes ISAKMP sont chiffrés avec la clé secrète déjà créée. Maintenant, les deux entités échangent leur identité (bloc *Identification*), facultativement leur certificat (bloc *Certificate*) et s'authentifient avec une signature digitale sur l'ensemble des paramètres négociés (bloc *Signature*).

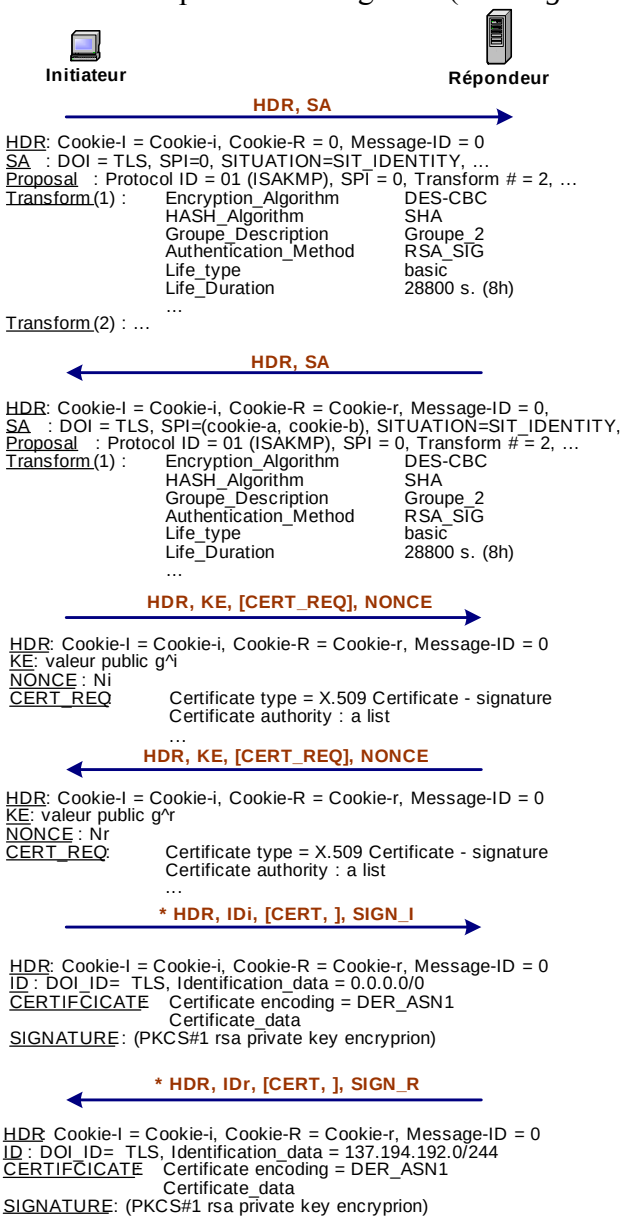


Figure 4-7. L'échanges ISAKMP phase 1

Pour les blocs d'identification, nous proposons que le premier bloc envoyé par le client contient une adresse IP anonyme 0.0.0.0/0 tandis que le bloc envoyé par le serveur contient son adresse IP réelle (par exemple, 137.194.192.210/0). Si le serveur héberge plusieurs serveurs virtuels, il doit alors préciser un intervalle qui englobe toutes les adresses IP des serveurs virtuels ou internes (par exemple, 137.194.192.0/225).
Quand un certificat réel est présent dans le bloc *certificat*, le récepteur peut utiliser l'information directement, après avoir vérifié qu'il a été signé avec une signature valide d'une autorité de confiance. S'il n'y a aucun certificat dans le message alors il est de la responsabilité du récepteur d'obtenir un certificat en utilisant une certaine méthode mise en place.
Le bloc *signature* contient une signature sur un hachage produit par les deux entités sur l'ensemble des paramètres échangés. Les fonctions de hachages générées respectivement par l'initiateur et le répondeur sont :

HASH_I = prf (SKEYID, g^i, g^r, Cookie_i, Cookie_r, SA, IDi) et,
HASH_R = prf (SKEYID, g^i, g^r, Cookie_r, Cookie_i, SA, IDr)

où, *SA* représente le bloc SA envoyé par l'initiateur dans le premier message, y compris toutes les propositions et tous les blocs *Transform* proposés par celle-ci. Les deux identités (*IDi, IDr*), les cookies (*Cookie_i, Cookie_r*), les valeurs publiques de Diffie-Hellman (g^i, g^r) et la clé SKEYID sont également inclus de chaque coté dans le hachage.
Pour terminer, les deux entités doivent vérifier les identités et la signature digitale envoyées dans les messages 5 et 6 avant de passer à la négociation de la deuxième phase d'ISAKMP.

4.4.3.2 La négociation de TLS SA

Après l'établissement de la première association de sécurité, les deux entités commencent à négocier les paramètres de sécurité de SSL/TLS (figure 4-8).
Dans cette deuxième phase, l'initiateur (le client SSL/TLS) indique dans l'en-tête ISAKMP, «*TLS Quick Mode*» comme type d'échange. Il spécifie un identificateur du message (*message_ID*) non nul, ajoute les deux cookies de la phase 1 et met le champ « flag » à 1 pour indiquer que les messages sont chiffrés. Le premier message comprend aussi :

- Un Bloc *HASH* qui suit l'en-tête ISAKMP. Ce bloc contient le résultat de hachage pour toute ou une partie de ce message ISAKMP. Le bloc *HASH* est formé par une fonction prf de la manière suivante :

  HASH_1 = prf (SKEYID_a, *message_ID,* SA, NONCEi, KE)
- Un bloc *SA*, précisant « *TLS10* » comme protocole de sécurité utilisé dans cette phase.
- Un bloc *Proposal* qui se rapporte au protocole TLS10 auquel est associé un SPI (*Security Parameter Index*) choisi aléatoirement par l'initiateur.
- Un ou plusieurs blocs *Transform* pour le bloc *Proposal*. Ceux-ci définissent les différents attributs proposés par l'initiateur. Les principaux attributs sont :

Cipher Suite	: TLS_RSA_3DES_CBC_SHA
Authentification Method	: *Anononymous_client*
SA life	: 28800s
PFS	: no

- Un bloc *NONCE (NONCEi, NONCEr)* qui sert à transporter de nouveaux aléas.
- Un bloc *Key Exchange* (*KE*) qui sert à transporter les données nécessaires à la génération d'une nouvelle clé DH partagée. Ce bloc contient aussi le champ *group* qui indique le nombre premier utilisé pour la génération des clés. Ce champ prend par défaut la valeur « *Modular Exponentiation* » avec 768 bits pour le nombre premier et 2 pour le générateur DH.

- Deux blocs d'identification (*IDi, IDr*). Le premier sert (*IDi*) à identifier le client. Il contient l'adresse 0.0.0.0/0. Le deuxième (*IDr*) sert à identifier un serveur parmi le rang d'adresse envoyé dans le même bloc (*IDr*) pendant la première phase. Le mécanisme que nous proposons ici permet aux clients de préciser depuis leur premier message, le serveur auquel ils souhaitent accéder.
- Un bloc de demande de certificat (*Certificate Request*).

L'emploi des blocs *KE*, *HASH, ID et CERTReq* est optionnel. Le bloc *KE* est utilisé si les deux communicateurs veulent assurer la propriété de PFS (*Perfect Forward Secrecy*) dans la génération des secrets partagés. Comme expliqué dans le RFC 2409, la présence du bloc *HASH* après les blocs *SAs* est explicite dans chaque message. Par contre dans notre échange, ce bloc est optionnel si les deux communicateurs utilisent un mécanisme d'authentification forte tel que la signature RSA (*RSA_SIG*).

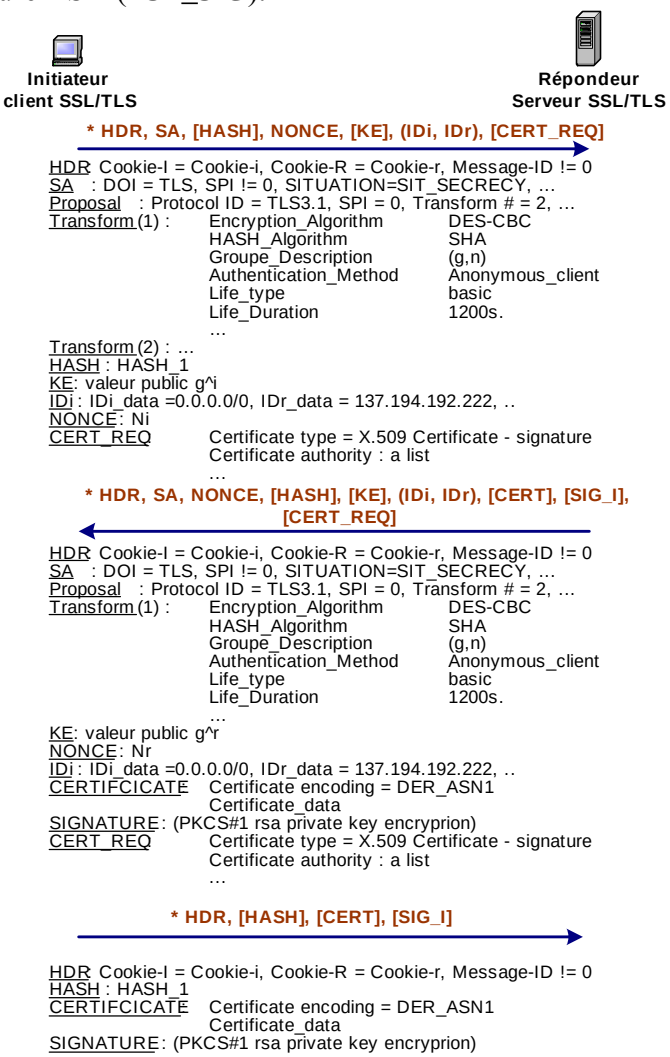


Figure 4-8. L'échange ISAKMP phase 2

Le destinataire (serveur SSL/TLS) communique en réponse le même *Message ID* et retourne la suite (*SA, Proposal, Transform*) correspondant à la solution retenue. Les blocs *Proposal* comportent aussi le nouveau *SPI* du destinataire. Le répondeur choisit de nouvelles valeurs pour les blocs suivants :

- Le bloc *NONCE* contient un nouveau nombre aléatoire.
- Le bloc *KE* contient la valeur publique DH du répondeur.
- Le bloc *HASH* contient maintenant la valeur HASH_2 définie de la méthode suivante :

HASH_2 = prf (SKEYID_a, NONCEi, *message_ID,* SA, NONCEr, KE)

- Le bloc c*ertificate* contient le certificat (ou l'URL du certificat) du répondeur (serveur identifié par le bloc IDr) envoyé suivant le type demandé par l'initiateur.
- Le bloc *signature* assure l'authentification des messages envoyés durant la phase 2. Il est établi sur le hachage (*HASH_2*) déjà calculé. Pour cela, la présence du bloc *signature* rend l'envoi du bloc *HASH_2* non nécessaire.

Les blocs *HASH*, *KE, CERT et SIG_I* sont optionnels et suite à la première demande de l'initiateur. Si le serveur souhaite authentifier son client, il envoie aussi le message *Certificate Request* tout en précisant le type et les autorités auxquels il fait confiance.
Dans le dernier message, l'initiateur authentifie les échanges à travers le bloc *HASH* ou à travers une signature générée avec son certificat (si demandé) sur le hachage des données échangées. Le hachage est calculé de la manière suivante:

HASH_3 = prf (SKEYID_a, 0, *message_ID,* NONCEi, NONCEr)

Quand le serveur SSL/TLS reçoit ce message, il vérifie ses informations (hash ou signature). Les deux entités peuvent alors commencer à employer la partie *Record* du protocole SSL/TLS pour protéger leurs flux de données.

La totalité des échanges de la phase 2 ne peut pas être identifiée à l'aide des *SPIs* puisque ceux-ci ne sont véhiculés que dans des blocs *Proposal*. Le champ Message ID, reporté systématiquement dans l'en-tête ISAKMP des messages échangés, permet alors de conserver la trace de la négociation en cours. La seconde phase se termine par l'établissement d'une ou plusieurs SA, chacune étant identifiée par le nom du protocole utilisé et par son SPI. Comme les SPI ont des valeurs distinctes pour chacun des interlocuteurs, la protection bidirectionnelle d'une communication par un protocole négocié résulte ainsi de l'utilisation conjointe de deux associations de sécurité élémentaires.

Les opérations cryptographiques à chiffrement asymétrique de TLS SA se ressemblent à celles expliquées dans le paragraphe 3.5.5 du chapitre précédent. En effet, les deux entités peuvent ne pas se procéder à une deuxième authentification pendant la phase 2, ni même à la propriété de PFS. Dans ce cas, il n'y a pas des opérations cryptographiques lourdes puisque tous les messages seront protégés par un chiffrement symétrique. Dans le seconde cas, le tableau 4-1 montre les opérations effectuées par le client et le serveur SSL/TLS qui souhaitent négociés *TLS Quick mode* avec une authentification forte par signature et certificat X.509.

Phase 2 (TLS Quick Mode)		
	RSA SIG + PFS	**RSA SIG**
Initiateur (client)	$DH_{op} + RSA_{sign} + RSA_{verify}$	$RSA_{sign} + RSA_{verify}$
Répondeur (serveur)	$DH_{op} + RSA_{sign} + RSA_{verify}$	$RSA_{sign} + RSA_{verify}$

Tableau 4-1. Opérations cryptographiques avec authentification mutuelle dans TLS SA

4.4.3.3 Générations des clés (phase 1 et 2)

En utilisant une fonction *prf*(*clé, message*), les deux entités vont générer un ensemble de clés pour le chiffrement des messages ISAKMP et la dérivation de nouvelles clés.

L'intégration d'ISAKMP dans SSL/TLS nécessite un mécanisme de génération des clés qui respecte le mécanisme de dérivation des clés produit par le protocole Handshake de SSL/TLS, défini dans le RFC 2246 (voir paragraphe 8.1). Ceci, afin d'obtenir les mêmes paramètres de sécurité utilisés par le protocole Record de SSL/TLS.

Le mécanisme que nous proposons diffère de celui utilisé dans le protocole IKEv1 par le point suivant :

- La génération de la clé SKEYID ne dépend pas de la méthode d'authentification utilisée pendant la négociation de la phase 1 d'ISAKMP.

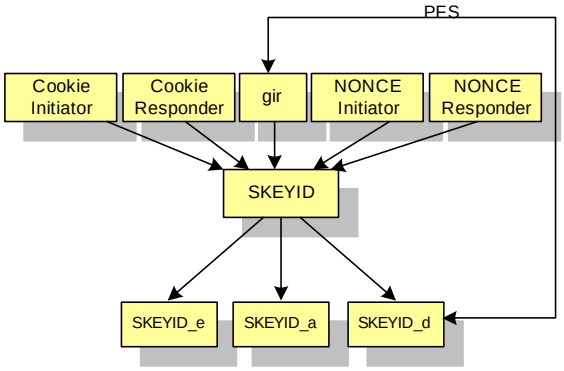


Figure 4-9. Dérivation des clés pour ISAKMP

Durant la phase 1, les deux entités vont dériver d'un nombre secret nommé SKEYID un ensemble des clés de chiffrement (*SKEYID_e*), d'authentification (*SKEYID_a*) et de dérivation des clés (*SKEYID_d*) (figure 4-9). La clé SKEYID est générée à partir des nombres aléatoires (*NONCEi, NONCEr*) et de la clé secrète DH (g^{ir}). Ceci est fait de la manière suivante :

$$SKEYID = prf\ (NONCEi \mid NONCEr, g^{ir})$$

A partir de *SKEYID*, les clés *SKEYID_a, SKEYID_e* et *SKEYID_d* sont dérivées :

$$SKEYID\_d = prf\ (SKEYID, g^{ir} \mid Cookie\_i \mid Cookie\_r \mid 0)$$
$$SKEYID\_a = prf(SKEYID, SKEYID\_d \mid g^{ir} \mid Cookie\_i \mid Cookie\_r \mid 1)$$
$$SKEYID\_e = prf(SKEYID, SKEYID\_a \mid g^{ir} \mid Cookie\_i \mid Cookie\_r \mid 2)$$

Ces clés serviront seulement au chiffrement et à l'authentification des messages en phase 2. Maintenant, à partir de *SKEYID_d*, les clés de chiffrement et d'authentification de SSL/TLS sont déduites de la clé matérielle suivante (figure 4-10) :

$$KEYMAT = prf(SKEYID\_d, [\ g(new)^{ir} \mid\ ]\ protocol \mid SPI \mid Ni \mid Nr)$$
$$+ prf(SKEYID\_d, K1 \mid [g(new)^{ir} \mid\ ]\ protocol \mid SPI \mid Ni \mid Nr)$$
$$+ prf(SKEYID\_d, K2 \mid [g(new)^{ir} \mid\ ]\ protocol \mid SPI \mid Ni \mid Nr)$$
$$+ [\ldots]\ ;$$

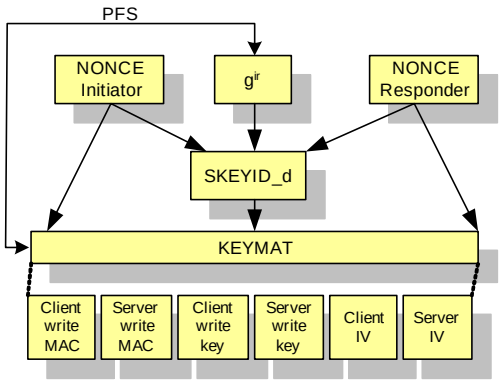


Figure 4-10. Dérivation des clés pour SSL/TLS

où *SPI* est celui de la SA TLS en question, $g(new)^{ir}$ est le nouveau secret commun éphémère de Diffie-Hellman et *protocol* est le nom du protocole utilisé (par exemple *TLS-HTTP*, etc.). Notons que le champ contenant le nouveau secret DH est ajouté si le service de PFS est négocié durant la deuxième phase d'ISAKMP.
Le calcul de KEYMAT est réitéré autant de fois qu'il est nécessaire pour que le bloc de chiffrement soit suffisamment long pour en extraire les éléments suivants :
client_write_MAC_secret, server_write_MAC_secret, client_write_key, server_write_key, client_write_IV et *server_write_IV* sont définis dans le RFC 2246.
La taille et la longueur de ces éléments sont négociées dans le premier échange (bloc *proposal*) après avoir été définies dans le domaine d'interprétation de SSL/TLS.

4.4.3.4 Scénario d'authentification par un certificat d'attribut

Le standard 2408 définit une méthode d'authentification par certificat d'attribut (CA). Dans le projet RNRT ICARE [ICARE], nous avons développé un nouveau type de certificat d'attribut en XML (figure 4-11). Ce type possède plusieurs avantages par rapport au standard X.509, tels que :

- La flexibilité du format XML sur le format ASN.1 [deme04].
- L'utilisation conjointe avec les nouveaux standards cryptographiques : XML-DSIG et XML-ENCRYPTION.
- La possibilité de délégation des rôles entre acteurs.
- L'absence d'une lourde infrastructure de gestion des privilèges : le prototype que nous avons développé est sous forme d'un portail Web avec des fonctionnalités avancées pour la génération et la distribution des certificats d'attributs.

Pour utiliser ce type de certificats avec ISAKMP, nous devons l'ajouter dans la liste de définition des 'types de certificats'. La liste ci-dessous montre les types actuels supportés par le standard ISAKMP ainsi que notre nouveau type : *ICARE Attribute Certificate.* TBD désigne un numéro qui doit être attribué par l'IANA.

Certificate Type	**Value**
NONE	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate - Signature	4
X.509 Certificate - Key Exchange	5
Kerberos Tokens	6
Certificate Revocation List (CRL)	7
Authority Revocation List (ARL)	8

SPKI Certificate	9
X.509 Certificate - Attribute	10
ICARE Attribute Certificate	TBD

La figure suivante montre un certificat d'attribut en XML qui englobe le certificat d'identité qui s'y raccroche.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
</ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
</ds:SignatureMethod>

<ds:Reference URI="#IcarePaquet">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments">
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
<ds:DigestValue>BPPdZmNw08WFVWSaxC8cbAwWrr8=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
kbJ3ZiJhhD4J3+u8MOpliw88sZVBz1isCO46bRjJq6DDTf/Za9ITt884OJ3nKgKth06Q5QdB4jUl
TX3oNrbbfPRFKshvi8l6GH8JGJaRODs2az4MWe9ugRLf/kY9Tn25vSVHgg3HPaxbaWiyw3jX3DBt
qo/jVcUZ65Q9tnYsdXo=
</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIICWjCCAcMCAQUwDQYJKoZIhvcNAQEEBQAwgYUxCzAJBgNVBAYTAkZSMQwwCgYDVQQIEwNJREYx
FjAUBgNVBAcTDUdFTk5FVklMTElFUlMxDzANBgNVBAoTBlRIQUxFUzEMMAoGA1UECxMDVEFJMREw
….
AQABMA0GCSqGSIb3DQEBBAUAA4GBAHf6ZMphMPN6PG/g3JozC+o42rcziFQE3k7OZE3cJaroRnyH
2lDDFXEUVmZ/bkC5VAs4hl4boVYiO3WMMLOw0y5zBbNFDJZmJWwbztmndseUD6BG7TAM9TE95K0v
drFr8jo07faoD5oCVGPuxd3308Lpwir0pQi3GeEXn+EGHJjR
</ds:X509Certificate>
</ds:X509Data>
<ds:KeyValue>
<ds:RSAKeyValue>
<ds:Modulus>
06iChrynsFnQ1vE81AvMz3J6UI15I8H3PqfkCiHWfRkCy11cFQqWlgwd+rVoECwmpoUmU15TV8l6
xwJmUKzXi1tFnoF759XwhMtk4Fr/YJhcrpLUvh77t2nBdqPEfHVPmi/y3WmROIEbDBBGVEP0uOJm
P7UPeS2+HdZJ0agtiz0=
</ds:Modulus>
<ds:Exponent>AQAB</ds:Exponent>
</ds:RSAKeyValue>
</ds:KeyValue>
</ds:KeyInfo>

<ds:Object Id="IcarePaquet">
<AttributeCertificate>
<InfoCertificat>
<Version>V1.0</Version>
<Type>AttributeCertificate</Type>
<Id>13</Id>
</InfoCertificat>
<Content>
<Holder>
<Identity>
<UserDN>EmailAddress=hajjeh@enst.fr, CN=IBRAHIM</UserDN>
<Serial>1</Serial>
<PublicKey>
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDfjrkIwAgmyhHvuKkbnKTFpU0MEnoOalr3JuIH&#xD;
3N6NJRgVmlojJsuJZG60qcbwNTlCdnrrp4cggpfoioXVtBnpvxtX3MN6zIrfOLg/60ZnX6Eaz17y&#xD;
1kyM9bS2Wvea3V+Ly64qcdN7CvjW/R3orMCfwhhwZcuHTq6nyrSg7fcTxQIDAQAB
</PublicKey>
</Identity>
</Holder>
```

```
<Attribut>
<Validity>
<ValidityFrom>2003/06/16 17:23:19</ValidityFrom>
<ValidityTo>2003/06/27 00:00:00</ValidityTo>
</Validity>
<Droits>Enseignant</Droits>
<Application>CONSOTEL</Application>
<Resource>//tripolis.enst.fr:portail/index.jsp</Resource>
</Attribut>
</Content>
</AttributeCertificate>
</ds:Object>
</ds:Signature>
```

Figure 4-11. Format d'un certificat d'attribut XML développé dans le projet ICARE

Avec le scénario précédent, le client devrait dans la deuxième phase de négociation indiquer la valeur *ICARE Attribute Certificate* comme méthode d'authentification. Si le serveur SSL/TLS accepte cette méthode d'authentification, le client envoie son certificat d'attribut en format BER (binaire) dans le bloc *Certificate* tout en précisant le type *ICARE Attribute Certificate* et il signe toutes les données déjà envoyées. A la réception, le serveur vérifie, le certificat d'identité du client, sa relation avec le certificat d'attribut (Serial Number), la validité du certificat d'attribut, la ressource à laquelle il souhaite accéder et la signature.

4.4.4 ISAKMP et le protocole Record de SSL/TLS

La dernière étape de notre architecture est le chiffrement de toutes les données échangées par le protocole *Record* de SSL/TLS.
Nous proposons un mécanisme simple qui ne nécessite aucun changement au niveau du protocole *Record*. En effet, ce dernier est activé lorsque les deux entités reçoivent les messages CCS (*Change Cipher Spec*) durant le dernier échange du protocole Handshake de SSL/TLS. Suivant le RFC 2246, les deux messages CCSs ne font pas partie du protocole Handshake. Ils se composent d'un message simple (un byte de valeur 1) envoyé par le client et le serveur pour informer la partie de réception que les données suivantes sont protégées avec les nouvelles clés négociées. Ce message informe le protocole *Record* de copier immédiatement les clés classées dans le «pending state» vers les «curent state».

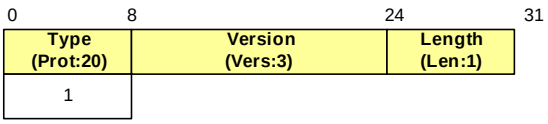


Figure 4-12. Le format du message CCS de SSL/TLS

Afin de protéger l'envoi des CCS par une personne tierce malveillante, nous proposons de les concaténer avec la première chaîne de données protégées par les clés de chiffrement déjà activées (figure 4-2). Si les messages CCSs n'existent pas, la connexion SSL/TLS est rejetée.

4.4.5 ISAKMP et TLS Extensions

Le RFC 3546 (*TLS Extensions*) propose une nouvelle extension au protocole SSL/TLS vers de nouvel environnement. Ce travail a renforcé notre proposition et a résolu tous les problèmes de compatibilité des clients SSL/TLS qui veulent négocier ISAKMP à la place du Handshake de SSL/TLS et des serveurs SSL/TLS qui ne soutiennent pas cette option et vice versa.

4.4.5.1 Procédure pour la définition de nouvelles extensions

La nouvelle extension que nous proposons dans ce chapitre et dans le chapitre suivant respecte la procédure de définition de nouvelles extensions avec TLS Extensions. Nous relevons ci-dessous certains points qui devraient être pris en considération par ce protocole :

- Les extensions doivent être définies de telle sorte que chaque extension envoyée par le client soit connue par le serveur. Ce dernier répond avec une extension du même type.

- Dans le cas où le serveur n'accepte pas l'extension, un message d'erreur est envoyé. En général, les messages d'erreurs ALERT de SSL/TLS devraient être envoyés au serveur. Un champ dans l'extension du serveur est envoyé au client.

- Les extensions doivent éviter toute attaque qui force à l'utilisation (ou à la non utilisation) d'une caractéristique particulière manipulant les messages Handshake. Ceci est vrai si les extensions sont ajoutées au message Finished de SSL/TLS. Les développeurs du protocole doivent être conscients que ces extensions ne changent pas le sens des messages envoyés dans le Handshake de SSL/TLS.

4.4.5.2 Négociation de l'échange ISAKMP

Afin de permettre à un client TLS de négocier un échange ISAKMP au lieu du Handshake standard de TLS, un nouveau type d'extension devrait être ajouté aux messages *ExtendedClientHello* et *ExtendedServerHello*. Ces deux messages incluent l'extension de type "*isakmp_key_negotiation*" avec la valeur "*ISAKMP_KN_Request*" pour le champ "*extension_data*". La syntaxe SSL/TLS de ce champ est définie comme suit :

```
enum {
        false(0), true(1);
        } Boolean;
struct{
        Boolean ISAKMPSA_Present;
        Select (ISAKMPSA_Present) {
                Case true: Begin_ISAKMP_Phase2
                Case false: Begin_ISAKMP_Phase1
                } request;
        } ISAKMP_KN_Request;
struct {
  ISAKMP_phase2_Exch  ISAKMP_phase2_Exch_list<1..2^16-1>;
        } Begin_ISAKMP_Phase2;
struct {
  ISAKMP_phase1_Exch  ISAKMP_phase1_Exch_list;
        } Begin_ISAKMP_Phase1;
struct {
        DOI doi_id ;
        } ISAKMP_phase1_Exch;
struct {
        DOI doi_id ;
        SPI_value spi-value;
        } ISAKMP_phase2_Exch;
opaque SPI_value<1..2^16-1> ;
opaque Proposal_number[1];
opaque doi_id[4] ;
```

La présence de l'extension "*isakmp_key_negotiation'* dans le premier échange SSL/TLS donne au client et au serveur TLS la possibilité d'utiliser une association de sécurité ISAKMP existante ou bien de créer une nouvelle association.
Si une association de sécurité ISAKMP est déjà établie entre le client et le serveur TLS, ils peuvent commencer directement par un échange phase 2 pour la négociation des paramètres de sécurité propres à SSL/TLS. Sinon, les deux entités vont établir les deux phases de négociation.
Si "*isakmpsa_present*" contient la valeur booléenne *false*, le client TLS demande l'ouverture des deux phases ISAKMP SA et TLS SA. Le client TLS envoie seulement l'identificateur du DOI TLS (par exemple 5). Dans ce cas, le client TLS est forcé de rejeter les communications avec les serveurs qui ne connaissent pas cette extension ou cette marque de DOI.
Si "*isakmpsa_present*" contient la valeur booléenne *true*, le client TLS commencera une TLS SA basée sur une association de sécurité ISAKMP déjà établie. Le client TLS peut envoyer une liste d'associations de sécurité ISAKMP établies. Le client envoie aussi l'identificateur de chaque association, le champ *SPI_id* et le DOI qu'il veut utiliser.

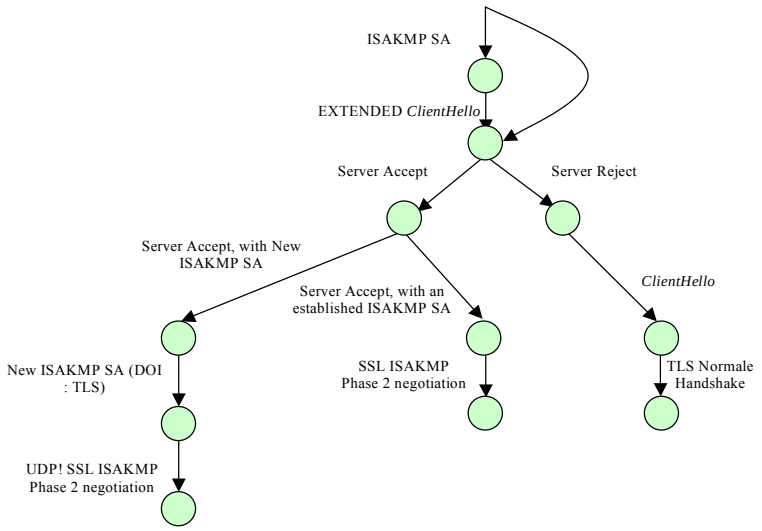


Figure 4-13. Automate client pour la négociation d'ISAKMP avec TLS Extensions

4.4.5.3 La gestion des erreurs

Cette section définit de nouveaux types d'erreurs pour l'usage avec ISAKMP et TLS. Les deux nouvelles alertes définies sont : *unpresent_isakmp_sa* et *unrecognized_spi_value.*
Ces alertes ne doivent être envoyées qu'à la suite de l'utilisation de TLS Extension entre le deux communicants.

- *unsupport_isakmp_sa* : cette alerte est envoyée par le serveur si le client demande l'ouverture d'une négociation TLS SA avant l'établissement d'une association de sécurité ISAKMP.
- *unrecognized_spi_value* : cette alerte est envoyée par le serveur s'il ne reconnaît pas l'identificateur de la phase envoyé par le client.

Les autres types cités dans la liste suivante sont définis dans les RFC 3546 et 2246.

```
enum {
   close_notify(0),                  /*RFC 2246*/
         ….                          /*RFC 2246*/
   no_renegotiation(100),            /*RFC 2246*/
```

```
    unsupported_extension(110),          /* RFC 3546 */
    certificate_unobtainable(111),       /* RFC 3546 */

    unrecognized_name(112),              /* RFC 3546 */
    bad_certificate_status_response(113),/* RFC 3546 */
    bad_certificate_hash_value(114),     /* RFC 3546 */
    unpresent_isakmp_sa (TBD)
    unrecognized_spi_value (TBD)
} AlertDescription;
```

4.4.6 Proposition d'une DOI pour SSL/TLS

L'intégration d'ISAKMP dans SSL/TLS exige le prolongement du DOI existant. Pour la phase 1, toutes les définitions dans IPSec DOI [RFC2407] peuvent être employées sans aucun changement avec notre TLS DOI, y compris la manière dont les pairs sont authentifiés. Cependant, avec le TLS DOI, une seule exception est à ajouter :

- Toutes les négociations d'ISAKMP et SSL/TLS peuvent fonctionner sur le port 500 de UDP. Si les implémentations souhaitent utiliser un deuxième port, TBD (ToBeDefined) à la place du port standard d'IKE, les communicants doivent envoyer la valeur zéro dans le bloc d'identification pendant la première phase (IKE standard permet 0 ou 500).

Les procédures concernant la phase 2 ont lieu sans aucun changement à quelques exceptions près :

- Un nouvel identificateur pour SSL/TLS (*protocole ID*) est défini.
- Un nouveau type d'échange est défini.
- Un nouveau type de *Transform* est défini.
- De nouveaux attributs sont ajoutés.

La figure 4-12 schématise le domaine d'interprétation du protocole SSL/TLS.

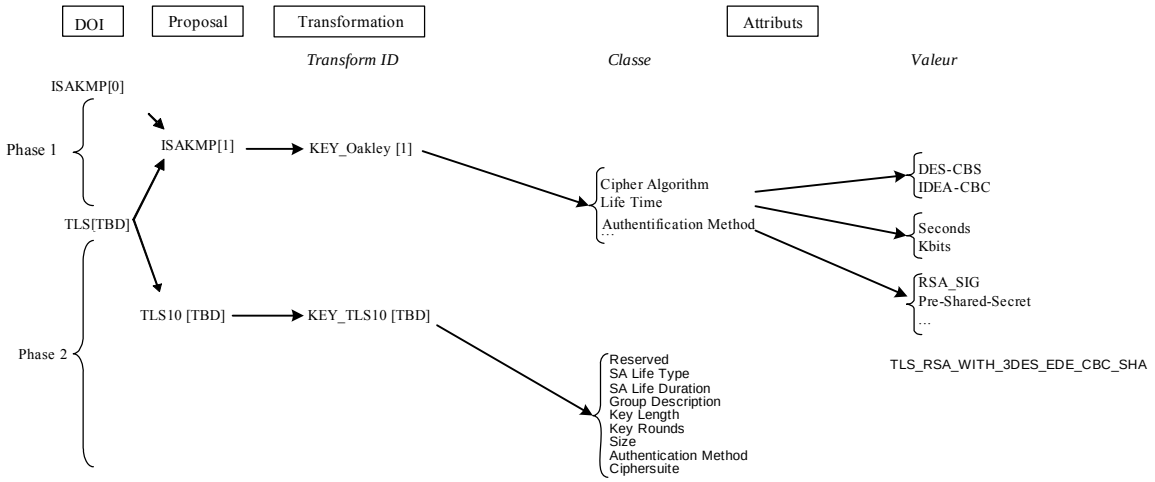


Figure 4-14. Présentation symbolique du domaine d'interprétation d'ISAKMP SSL/TLS

4.4.6.1 L'attribution des numéros

Dans ISAKMP, tous les DOIs doivent être inscrits à l'IANA dans le RFC 3232 [RFC3232]. Nous désignons par TBD les numéros qui doivent être assignés par l'IANA.

La première valeur à identifier est celle du domaine d'interprétation de SSL/TLS. Le domaine d'interprétation prend une valeur de 32 bits qui identifie le contexte dans lequel les associations de sécurité sont évaluées. Suivant l'IANA, la demande d'attribution d'un

nouveau numéro de domaine d'interprétation doit être accompagnée par des spécifications publiques, telles que les RFC de l'IETF.

DOI	Valeur	Référence
ISAKMP	0	[RFC2408]
IPSEC	1	[RFC2407]
GDOI	2	[RFC3547]
TLS	TBD	

De plus, avec le TLS DOI, des numéros doivent être assignés pour les paramètres suivants : *Protocol Identifiers, TLS Transform Identifiers, Security Association et Attribute Type Values.*

4.4.6.2 Définition d'une Situation

Dans ISAKMP, le champ *situation* fournit des informations qui peuvent être employées par le répondeur pour déterminer la politique et la manière de traiter les demandes d'association de sécurité entrantes. Pour le TLS DOI, le champ *Situation* occupe quatre octets avec la valeur suivante :

Situation	Value
SIT_IDENTITY_ONLY	0x01

Le type SIT_IDENTITY_ONLY indique que l'association de sécurité est identifiée par les identités utilisées dans l'association de sécurité actuelle. Toutes les implémentations d'ISAKMP avec TLS doivent supporter ce type de situation en incluant deux blocs d'identification dans l'échange de la deuxième phase et doivent empêcher toute négociation qui ne respecte pas ce type d'échange.
Le type des ces blocs d'identification est défini par le RFC 2407. Le tableau ci-dessous présente les valeurs assignées pour les différents types des blocs d'identification. Les valeurs de ces types sont décrites dans la section 4.6.2.1 du [RFC2407].

ID Type	Value	Référence
RESERVED	0	[RFC2407]
ID_IPV4_ADDR	1	[RFC2407]
ID_FQDN	2	[RFC2407]
ID_USER_FQDN	3	[RFC2407]
ID_IPV4_ADDR_SUBNET	4	[RFC2407]
ID_IPV6_ADDR	5	[RFC2407]
ID_IPV6_ADDR_SUBNET	6	[RFC2407]
ID_IPV4_ADDR_RANGE	7	[RFC2407]
ID_IPV6_ADDR_RANGE	8	[RFC2407]
ID_DER_ASN1_DN	9	[RFC2407]
ID_DER_ASN1_GN	10	[RFC2407]
ID_KEY_ID	11	[RFC2407]

4.4.6.3 TLS Protocol Identifier

L'identificateur du Protocole (*Protocol ID*) est un champ sur 8 bits qui identifie le protocole de sécurité négocié. PROTO_TLS10 doit être explicitement ajouté à la liste pour identifier l'utilisation du protocole SSL/TLS v1.0.

Protocol ID	Value	Référence
RESERVED	0	[RFC2407]
PROTO_ISAKMP	1	[RFC2407]
PROTO_IPSEC_AH	2	[RFC2407]
PROTO_IPSEC_ESP	3	[RFC2407]

PROTO_IPCOMP	4	[RFC2407]
PROTO_TLS10	TBD	

La syntaxe proposée par ISAKMP a été spécifiquement conçue pour tenir compte de la négociation simultanée des multiples protocoles de sécurité durant la phase 2 et dans une seule négociation de la phase 1. Par conséquent, la suite des protocoles énumérés ci-dessus forme l'ensemble des protocoles qui peuvent être négociés simultanément.
Le fait d'ajouter les protocoles définis au RFC 2407 à notre table est lié au fait que cette proposition ne nécessite aucun changement au niveau de notre architecture ou aux négociations faites par les entités communicantes. En effet, chaque négociation de la phase 2 est identifiée par un SPI et par le nom du protocole utilisé. Ceci permet aussi d'unifier plusieurs DOI proposés actuellement [RFC3547] [Tsch03] [Arkk02]. La décision concernant les protocoles pourrait être négociées simultanément dépend de la politique de chaque hôte.

Le type PROTO_ISAKMP indique la protection de message exigée durant la phase 1 d'ISAKMP. Toutes les implémentations de DOI d'ISAKMP avec TLS doivent comprendre PROTO_ISAKMP. Notons aussi que ISAKMP réserve la valeur 1 pour toutes les définitions des DOIs. Le type PROTO_TLS fournit les paramètres nécessaires pour l'association de sécurité utilisée dans SSL/TLS.

4.4.6.4 TLS Exchange type

Pour TLS, un seul nouveau type d'échange doit être ajouté : *TLS_Quick_Mode*. La liste devient alors :

Exchange Type	**Value**	**Référence**	
NONE	0	[RFC2408]	
RESERVED	1	[RFC2408]	// Base
Identity Protection	2	[RFC2408]	
RESERVED	3	[RFC2408]	// Authentication Only
RESERVED	4	[RFC2408]	// Aggressive
Informational	5	[RFC2408]	
TLS_Quik_Mode	TBD		

4.4.6.5 TLS Transform Identifier

Transform Identifier est un champ sur 8 bits qui identifie la politique à utiliser pour assurer les services d'authentification et d'intégrité dans SSL/TLS.
Nous définissons une nouvelle liste 'KE_TLS10' qui contient la liste des algorithmes indiquée dans le standard TLS v1.0 de l'IETF. La définition de cette liste est explicite pour l'intégration de SSL/TLS avec ISAKMP.

Transform ID	**Value**
Reserved	0-1
KE_TLS10	TBD

4.4.6.6 TLS Security Associations Attributes

Pour chaque *Tranform ID* défini dans KE_TLS, une liste d'attributs est définie. Les attributs de KE_TLS sont composés d'une valeur et d'un type *B* pour basic (16 bits) et *V* pour longueur variable.
Les attributs décrits comme *Basic* ne doivent pas être codés comme variables. Les attributs de longueur variable peuvent être codés en tant qu'attributs *Basic* si leur valeur est sur deux

octets [RFC 2409]. Toutes les restrictions énumérées dedans [RFC2409] s'appliquent également à TLS DOI.
Les attributs décrits ci-dessous sont similaires à ceux décrits dans le document IPSec DOI. Dans le but de réutiliser le code de IPSec DOI, les paramètres non employés par TLS DOI sont définis comme étant de classe *RESERVED* (valeurs 4, 8 et 9).

Class	**Value**	**Type**	**Réference**
Reserved	0		
SA Life Type	1	B	
SA Life Duration	2	V	
Group Description	3	B	
RESERVED	4	B	// Encapsulation Mode
RESERVED	5	B	// Authentication Algorithm
Key Length	6	B	
Key Rounds	7	B	
RESERVED	8	B	// Compress Dictionary Size
RESERVED	9	V	// Compress Private Algorithm
Authentication Method	TBD	B	
Ciphersuite	TBD	B	

Pour garder l'interopérabilité entre les différentes implémentations, toutes les réalisations doivent pouvoir négocier les attributs suivants : *SA Life Type, SA Life Duration, SA Life Duration, Authentification Method et Ciphersuite*

Nous expliquons dans la suite, les deux nouveaux attributs : *Authentification Method et Ciphersuite.* Pour les autres attributs, ils possèdent les mêmes valeurs définies dans le RFC 2409.

4.4.6.6.1 Authentication Method

Dans SSL/TLS, l'authentification du serveur est toujours obligatoire alors que celle du client est optionnelle. La seule méthode d'authentification du serveur s'effectue avec des certificats et des signatures RSA ou DSS [DSS]. Le client a le choix entre une authentification anonyme (*Anonymous_client*), une clé partagée (*Pre-shared key*), le chiffrement asymétrique et la signature (RSA ou DSS).
La liste des méthodes d'authentification est la suivante:

Class	**Value**	**Réference**
Pre-shared key	1	//Pre-Shared Key
DSS signatures	2	
RSA signatures	3	
Encryption with RSA	4	
RESERVED	5	//Revised encryption RSA
Anonymous_client	6	

4.4.6.6.2 CipherSuite

Un changement important par rapport à IKE v1, est que dans TLS DOI nous négocions une suite de chiffrement au lieu de négocier les algorithmes.
Ceci permet non seulement d'effectuer moins d'échanges, mais aussi de réduire le taux d'erreurs dans des négociations qui sont peu susceptibles d'être utiles [Kraw01].

La liste des suites de chiffrement est la suivante:

Class	Value
TLS_NULL_WITH_NULL_NULL	1
TLS_RSA_WITH_NULL_MD5	2
TLS_RSA_WITH_NULL_SHA	3
TLS_RSA_EXPORT_WITH_RC4_40_MD5	4
TLS_RSA_WITH_RC4_128_MD5	5
TLS_RSA_WITH_RC4_128_SHA	6
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	7
TLS_RSA_WITH_IDEA_CBC_SHA	8
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	9
TLS_RSA_WITH_DES_CBC_SHA	10
TLS_RSA_WITH_3DES_EDE_CBC_SHA	11
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	12
TLS_DH_DSS_WITH_DES_CBC_SHA	13
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	14
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	15
TLS_DH_RSA_WITH_DES_CBC_SHA	16
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	17

4.5 Conclusion

Les protocoles de gestion des clés forment une composante essentielle pour la mise en œuvre des services de sécurité. De nombreux protocoles comme SSL/TLS proposent leur propre protocole de gestion des clés. Le protocole ISAKMP peut être employé comme une plate-forme standard pour la négociation des paramètres de sécurité de tout protocole de sécurité. L'intégration d'ISAKMP dans SSL/TLS implique plusieurs avantages :

- *Etablissement des canaux sécurisés*
  ISAKMP fournit une méthode standard et expérimentée pour l'établissement des canaux sécurisés durant la négociation des paramètres de sécurité. Les étapes de négociation du protocole (identification et authentification) sont largement validées et prouvées comme sécurisées [Sierr02] [Can02].

- *Économisation des ressources*
  Une fois que les deux hôtes établissent un canal sécurisé, il est possible de convenir diverse future utilisation des SAs [Perl01]. D'ailleurs, les associations de sécurité appartiennent à différents protocoles de sécurité, ce qui permet d'économiser les ressources. En effet, il est possible de négocier plusieurs SAs dans la même phase. En outre, le niveau de sécurité de chaque SA est également élevé.

- *Flexibilité*
  ISAKMP est nettement plus flexible que d'autres protocoles, en particulier le Handshake de SSL/TLS et a donc la capacité de répondre à des besoins de services de sécurité beaucoup plus larges. Même si le protocole ISAKMP est considéré trop ouvert pour être une norme standard, cette flexibilité est l'un de ses avantages parce que c'est la seule manière de fournir une solution optimale pour chaque système d'information dans Internet.

Ce chapitre a montré la capacité et la nécessité de migrer les protocoles de sécurité courants vers ISAKMP. Cette tâche est d'autant plus difficile que IP est entrain de migrer vers la version 6. Avec IPv6, tous les hôtes seront en mesure de comprendre la négociation d'ISAKMP. Ainsi, il sera aisé de tirer profit d'un simple processus de négociation pour les différents protocoles de sécurité.

Chapitre 5

5 Génération d'une preuve de non répudiation avec SSL/TLS

5.1 Résumé

Dans ce chapitre, nous proposons une architecture générique pour la fourniture optionnelle du service de non répudiation à travers le protocole SSL/TLS.
Notre approche TLS-SIGN permet aux deux entités SSL/TLS de générer une preuve sur le contenu des données échangées entre elles. De plus, nous expliquons que notre proposition permet une séparation claire entre la fourniture du service de non répudiation et la conception et le développement des applications.
Ce travail a donné lieu à deux publications : [Hajj04a] et [Hajj04b].

5.2 Introduction

SSL/TLS est actuellement le protocole de sécurité le plus répandu. Il permet de sécuriser les transactions Internet grâce à l'authentification du client (un navigateur la plupart du temps) et du serveur, et au chiffrement de toutes les données échangées. SSL/TLS utilise le protocole fiable TCP afin de fournir:

- *La confidentialité* et *l'intégrité* des données transmises et,
- *L'authentification* des communicateurs (authentification obligatoire pour le serveur et optionnelle pour le client).

De nos jours, SSL/TLS est sans aucun doute le protocole de sécurisation le plus utilisé par la majorité des serveurs et navigateurs Web. Cependant, comme d'autres protocoles de sécurité, SSL/TLS ne fournit pas un service de non répudiation. Ainsi, l'application doit contrôler elle-même les échanges signés, ainsi que le mécanisme de stockage des données [Wich99]. Ceci induit une complexité au niveau applicatif et alourdit les tâches de développement et de conception des applications.

Afin de remédier à ce problème, nous proposons une intégration générique du service de signature TLS-SIGN (TLS Signature) dans le protocole SSL/TLS. Le service de signature est optionnel et peut être utilisé avec tous les types d'applications d'une manière transparente et avec le minimum d'effort de programmation. Notre approche est générique du fait que plusieurs types de signature, d'échange et d'authentification peuvent être négociés pendant la phase d'initialisation de SSL/TLS. Afin de garder l'interopérabilité avec les versions existantes de SSL/TLS, TLS-SIGN est négocié grâce au nouveau standard TLS Extensions

décrit ultérieurement dans ce chapitre. De plus, TLS-SIGN exige que les deux entités, client et serveur soient authentifiées en utilisant leurs certificats X.509. Ceci garantira que le signataire des messages SSL/TLS est identique à celui de l'expéditeur de ces messages.

5.3 Motivations

Actuellement, les applications du type e-commerce sont de plus en plus exigeantes en termeq de protection à long terme des données et de génération des preuves d'expédition et de réception des données transmises. L'intégration d'un service de signature dans un protocole de sécurité comme SSL/TLS mène à plus d'interopérabilité et permet de fournir des preuves d'expédition et de réception d'une manière fiable et à travers des formats standard. La preuve[8] correspond à des données signées basées sur des normes proposées par l'IETF telles que S/MIME [RFC2311], PKCS#7 [RFC2315] ou même XML-DSIG[9] [RFC3275].

Nous avons proposé d'intégrer un service de signature à SSL/TLS pour les raisons suivantes:

1. Signature des transactions e-commerce

Les applications de type e-commerce sont actuellement les plus exigeantes en terme de besoins de sécurité. Il y a une demande insistance d'un équivalent électronique à la signature manuscrite. Les technologies requises à cette fin (S/MIME, PKCS#7v1.5, CMS [RFC3369]) ont été disponibles pendant plusieurs années mais elles sont totalement indépendantes des protocoles de sécurité en service, comme les protocoles SSL/TLS et IPSec. L'intégration d'un service de signature dans un protocole de sécurité mènera à plus de sécurité au niveau des transactions et fournira aux entités communicantes des preuves d'expédition et de réception des données. Ces preuves peuvent être présentées plus tard à un tiers pour la résolution des conflits.

2. Minimisation de la tâche de développement des modules de sécurité

Actuellement, lorsque les applications de sécurité exigent la présence des services de sécurité pour assurer la protection de leurs échanges, les développeurs et les concepteurs de ces applications proposent l'utilisation des mécanismes de protection standard tels que l'utilisation du protocole SSL/TLS ou du protocole SSH. En effet, ces protocoles intègrent des fonctions simples pour assurer l'authentification, la confidentialité et l'intégrité des données. A titre d'exemple, avec l'API OpenSSL, c'est en faisant appel à des fonctions comme *ssl_write* ou *ssl_read,* les données échangées entre un client et un serveur SSL/TLS sont protégées. Par contre, si les applications exigent une preuve de non répudiation, c'est alors aux développeurs de fournir ce nouveau module de signature.

Les développeurs qui sont dans la plupart des cas ni des individus non experts dans le domaine cryptographique ni dans le domaine des standards de signatures, procèdent aux développements des modules non conformes aux standards de signature. Ce qui amène à des implémentations non interopérables au niveau des mécanismes de signature et au niveau des méthodes de stockage.

TLS-SIGN réduira alors au minimum l'effort déployé par les développeurs et les concepteurs des applications dans la définition des services de non répudiation rattachés à chaque application. Ainsi, TLS-SIGN constitue un module générique qui peut être utilisé par de

[8] La preuve est un moyen qui permet d'étayer ses prétentions contre un adversaire devant un juge [cout04].

[9] La signature en XML (ou le XML-DSIG) est un effort commun du consortium de World Wide Web (*W3C*) et de l'IETF pour l'authentification des messages XML

simples appels à des procédures et fonctions similaires à celles de l'API OpenSSL (par exemple, *ssl_sign_write* et *ssl_sign_read* qui permettent respectivement la signature et la vérification des données).

3. Service de non répudiation générique

Un service de non répudiation est générique si ce service ne dépend pas du format de signature, des algorithmes échangés ou de tout autre paramètre de sécurité.
D'autres solutions telles que SOAP-DISG (*Simple Object Access Protocol- Digital Signture*) [Snell02] intègre un service de non répudiation au niveau de données. SOAP-DSIG assure la non répudiation des données XML échangées au-dessus d'un tunnel HTTP. Il utilise pour cela le format de signature XML-DSIG [RFC3275]. Généralement, une implémentation de SOAP est faite au niveau d'un serveur Web sécurisé pour manipuler les transactions SSL/TLS d'une manière transparente par rapport aux applications. Ainsi, le client commence à signer les données SOAP en utilisant le standard XML-DSIG. Ensuite, il envoie les données à la couche Record de SSL/TLS pour les transmettre dans un tunnel SSL/TLS sécurisé. Les données signées reçues par le serveur sont déchiffrées puis vérifiées en signature au niveau de chaque application. Cette solution a deux limitations par rapport à notre proposition.

La première limitation concerne la signature au format XML-DSIG. En effet, avec l'approche SOAP, toutes les données sont signées en utilisant le nouveau standard XML-DSIG. Cependant, dans le domaine des standards de signature numérique, XML-DSIG n'est pas le format le plus déployé. D'autres types comme S/MIME, CMS, PKCS#7 sont largement répandus surtout dans les navigateurs Web et les outils de messagerie. Ils sont également utilisés pour la multi- et la co- signature des données.

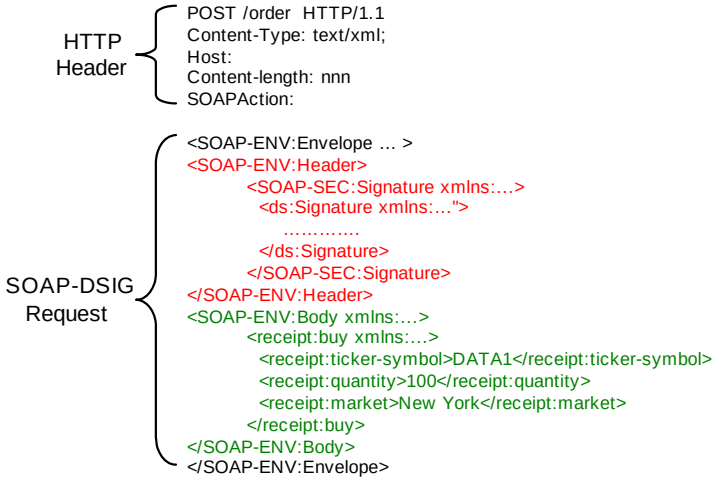


Figure 5-1. Une requête HTTP Request avec un message XML-DSIG

Quant à la deuxième limitation, elle concerne SOAP/XML-DSIG et le canal HTTP. En effet, SOAP a été conçu pour être déployé au-dessus du protocole HTTP. Il a pour but d'offrir un mécanisme standard pour l'échange de données en se basant sur XML (figure 5-1 et 5-2). Ainsi, cette technologie est réduite actuellement à des transactions Web de type B2B (*Business to Business*). Plusieurs autres types de transactions tels que, par exemple, le transfert des messageries et le transfert des fichiers ne peuvent bénéficier de cette technologie.

A la différence de SOAP-DSIG, l'objectif de TLS-SIGN est alors de fournir un module générique de non répudiation facilement intégrable avec tout type d'applications et qui ne sera

pas limité à un format ou à un environnement défini. TLS-SIGN bénéficie des avantages de SSL/TLS. Ainsi, il est transparent pour les applications et il négocie tous ses paramètres de sécurité (algorithme de chiffrement, fonction de hachage, taille des clés, etc.) à travers le protocole Handshake de SSL/TLS.

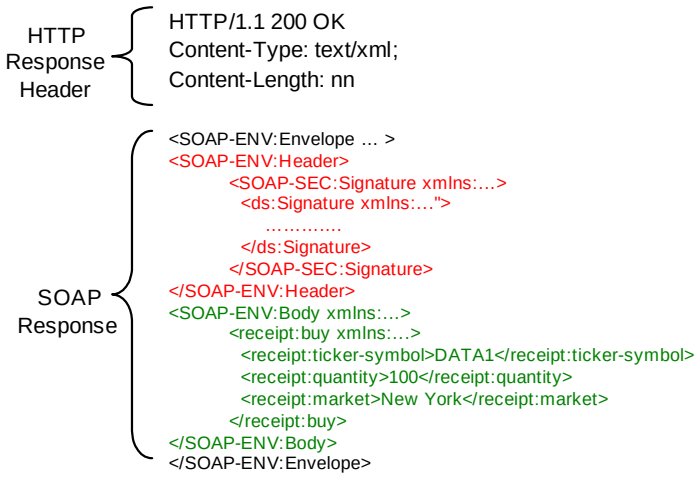


Figure 5-2. Une requête HTTP Réponse avec un message XML-DSIG

4. Modularité du protocole SSL/TLS.

Le fait que SSL/TLS soit un protocole modulaire rend notre approche facilement intégrable à ce protocole. Ainsi, notre proposition ne requiert aucun changement aux différents modules de SSL/TLS. D'autre part, la standardisation de 'TLS Extensions' rend notre proposition plus intéressante. En effet, TLS Extensions négocie d'une manière générique de nouvelles fonctionnalités et de nouveaux services dans le protocole SSL/TLS. Avec notre proposition, TLS Extensions négocie le service de non répudiation. En outre, il permet de garder l'interopérabilité avec la version standard du protocole SSL/TLS.

5.4 L'approche TLS-SIGN

5.4.1 Présentation générale

Le protocole SSL/TLS assure la protection des données et l'authentification des entités communicantes. Cependant, il manque à ce protocole la fourniture du service de non répudiation.

Le service de signature TLS-SIGN est intégré comme un module de plus haut niveau du protocole Record de SSL/TLS (figure 5-3). TLS-SIGN est optionnellement employé si le client et le serveur conviennent de l'utiliser pendant la phase d'initialisation du protocole. Quand TLS-SIGN est négocié, les étapes suivantes ont lieu :

1. Le client et le serveur échangent des messages de déclenchement de TLS-SIGN et négocient le format de signatures et quelques autres paramètres.
2. Le client et le serveur SSL/TLS présentent leurs certificats X.509 pour s'authentifier. Les certificats X.509 des deux entités (client et serveur) doivent contenir l'extension de signature.
3. Les deux entités (et facultativement un tiers de confiance) stockent toutes les données.

Dans les sections suivantes, nous détaillons ces étapes indiquées ci-dessus.

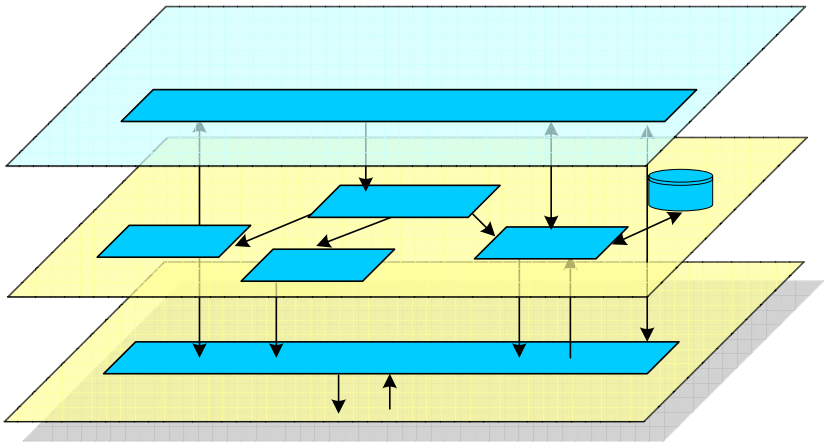


Figure 5-3 Architecture de TLS-SIGN

5.4.2 Contraintes de base

L'utilisation de notre proposition avec SSL/TLS oblige le client et le serveur à respecter quelques contraintes dans la phase d'ouverture d'une session TLS. Les contraintes sont les suivantes :

1. L'authentification explicite du client

TLS-SIGN est basé sur une session SSL/TLS sécurisée, afin de vérifier l'intégrité des données et de se protéger contre les attaques homme du milieu et l'attaque du rejeu des paquets. D'autant plus que le service de non répudiation est considéré comme une version plus forte du service d'authentification [RFC2828], l'authentification du client et serveur SSL/TLS est obligatoire avant toute opération de signature de données. Cette authentification protégera également le serveur SSL/TLS contre l'attaque de déni de service induite par la vérification et la signature exhaustive des données venant des clients anonymes. Le certificat client peut également être utilisé par le serveur comme preuve de communication puisque durant la phase d'initialisation de ce protocole, le client signera tous les paramètres négociés avec le serveur, ainsi que le temps de communication.

2. Négociation des suites de chiffrement et de hachage non nulles

Au cas où le module de signature est activé, la liste des suites de chiffrement est limitée à celle qui possède RSA ou DSA comme algorithme de chiffrement asymétrique avec un hachage non nul. Le tableau suivant montre les suites de chiffrement qui ne doivent pas être utilisées avec TLS-SIGN.

Echange de clés	**Signature**	**Chiffrement symétrique**	**Hachage**
RSA	RSA	DES-56 ou DES-40 3DES-168 RC2-40 ou RC2-56 RC4-40 ou RC4-128 ou RC4-56 IDEA-128	SHA1 SHA1 MD5 MD5 SHA1
DH	RSA	DES-40 DES-56 3DES-168 RC4-128 ou RC4-64	SHA1 SHA1 ou MD5 SHA1 ou MD5 SHA1

	~~NONE~~	~~DES-40 ou DES-56~~ ~~3DES-168~~ ~~RC4-40 ou RC4-128~~	~~SHA1~~ ~~SHA1~~ ~~MD5~~
	DSS	DES-40 ou DES-56 3DES-168 RC4-128	SHA1 SHA1 SHA1

Tableau 5-1. Suites de chiffrement reconnues par SSLv3 et Intégrées dans OpenSSL 0.9.6g

5.4.3 Signature électronique et identification

L'objectif de TLS-SIGN est de fournir aux deux parties communicantes l'évidence qui est stockée et présentée ultérieurement à un tiers. Ceci permet de résoudre un conflit si une entité communicante nie d'être impliquée dans la communication ou dans l'échange de données.
TLS-SIGN fournit les deux types de non répudiation suivants:

- La non répudiation de l'origine
- La non répudiation de la réception

La non répudiation de l'origine protège le destinataire confronté à un expéditeur niant avoir envoyé le message. D'autre part, la non répudiation de la réception joue le rôle inverse du précédent qui consiste à démontrer que le destinataire a bien reçu le message que l'expéditeur lui a envoyé à un instant spécifique.
Dans un cadre plus juridique, la directive européenne 1999/93 du 13 décembre 1999 envisage dans ces définitions une signature électronique avancée susceptible d'atteindre les objectifs juridiques attendus [Coud04]. Le texte définit ainsi cette variété de signature qui garantit l'identification et l'intégrité : une signature électronique qui satisfait aux exigences suivantes :

a) être liée uniquement au signataire ;
b) permettre d'identifier le signataire ;
c) être créée par des moyens que le signataire puisse garder sous son contrôle exclusif ;
d) et être liée aux données auxquelles elle se rapporte de telle sorte que toute modification ultérieure des données soit détectable.

La directive renvoie au modèle des PKI comme un système de confiance entre acteurs. Par contre, elle ignore le côté structurant d'une politique de certification ainsi que l'autorité de certification.
Pour cela, en se basant sur la notion de confiance dans SSL/TLS et l'utilisation des certificats X.509v3 pour lier l'identité à la signature, TLS-SIGN aboutit à une preuve de non répudiation reconnue par la directive européenne. Cette preuve de non répudiation peut être stockée soit localement chez les deux communicateurs ou chez un tiers de confiance.

5.4.4 Les phases de vérification

La figure 5-4 illustre une méthode standard définie dans [RFC2828] pour la vérification et le stockage des données. Partant de cette méthode, nous l'avons améliorée en l'adaptant à notre architecture TLS-SIGN. Nous utilisons le terme '*critical action*' pour designer l'acte de communication entre les deux entités.

La fourniture du service de non répudiation nécessite les cinq étapes suivantes:

1. Le client demande explicitement un service de non répudiation avant tout envoi de données.
2. Si le serveur accepte la demande du client, ce dernier commence alors à générer des messages signés en utilisant les algorithmes négociés ainsi que sa méthode

d'authentification. Cette étape est établie d'une manière similaire pour le serveur SSL/TLS si le service de non répudiation de la réception est demandé par le client.

3. Les données signées sont ensuite envoyées par l'initiateur (client ou serveur) et elles sont localement stockées par l'émetteur ou par un tiers. Ceci, sera utilisé ultérieurement si nécessaire.
4. L'entité qui reçoit la preuve procède à la vérification des données.
5. La preuve est maintenant stockée par l'entité réceptrice pour tout usage ultérieur.

Puisque SSL/TLS ajoute le temps (*GMT*) dans son premier échange pour protéger la négociation contre le rejeu des paquets et parce que tous les messages de négociation des paramètres de sécurité sont signés par le certificat des deux acteurs (client et serveur), la valeur de temps peut être aussi stockée avec les données signées comme preuve de communication.

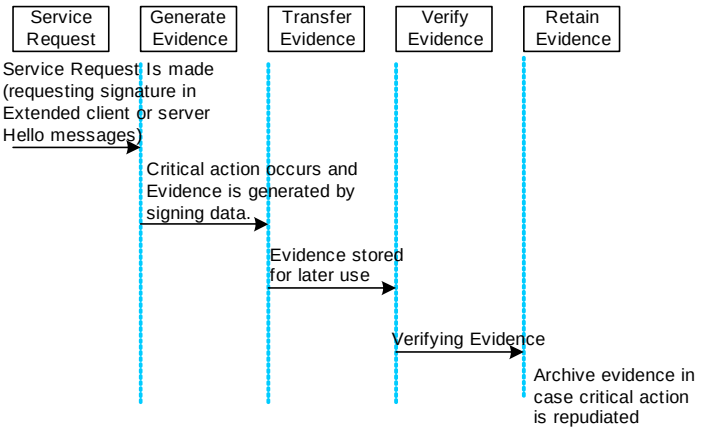


Figure 5-4. Phases de négociation du service de non répudiation et du stockage des données

5.4.5 Le protocole Handshake

5.4.5.1 Initialisation du service TLS-SIGN

Afin de permettre au client SSL/TLS de négocier notre service de signature, nous avons ajouté un nouveau type d'extension (signature) aux messages *ExtendedClientHello* et *ExtendedServerHello* définis dans [RFC3546]. Ces deux messages incluent l'extension de type «*signature*» avec la valeur «*signature_request*» pour le champ «*extension_data*». La syntaxe SSL/TLS [RFC1832] de ce champ est définie comme suit:

```
enum {
          pkcs7_1.5(0), smime (2), xmldsig(255);
        }ContentFormat; /* le format de données signées */

struct {
          ContentFormat          content_format;
          SigMethod              sig_meth;
          Boolean                bool;
          Signature_type         sign_type<1..2^16-1>;
          third_party            url_third;
        } signature_request;

enum {
            x509cert(0), x509cert_url(1), (255);
        } SigMethod; /* méthode de signature */
```

```
enum {
        false(0), true(1);
      } Boolean; /* activé ou désactivé le protocole de signature*/

opaque Signature_type<1..2^16-1>; /* type de signature : sign_with_proof_of_origine,
sign_with_double_proof, etc. */
opaque third<1..22^16-1> ; /* URL d'un tiers de confiance */
```

Le client commence le service TLS-SIGN en envoyant le message *ExtendedClientHello* avec l'extension de signature contenant le type de signature (non répudiation avec preuve de réception, non répudiation avec preuve d'expédition, etc.), le format de données signées (PKCS#7, S/MIME, etc.), l'URL d'un tiers de confiance (l'URL d'une base de données LDAP) et une valeur booléenne. Cette dernière est égale à *true* si le client souhaite signer les données et à *false* si on veut arrêter ce service. Le client précise également sa méthode de signature de données. Pour le moment, notre proposition nécessite que le deux entités utilisent leur certificat X.509 d'authentification pour signer les données échangées entre elles.
Le serveur peut accepter ce module en renvoyant la même extension «signature» dans le message *ExtendedServerHello*. Dans ce cas, le serveur devrait choisir une suite de chiffrement qui contient des algorithmes de chiffrement et de hachage non nuls, tels que par exemple, l'algorithme RSA pour le chiffrement asymétrique et SHA-1 pour le hachage. Puisque avec notre service de signature, l'authentification du client est obligatoire, le serveur doit aussi envoyer le message *CertificateRequest* puis le message *ServerHelloDone*. Le client répond en envoyant son certificat et il continue la négociation du Handshake avec le serveur.

Si le serveur ne connaît pas l'extension demandée par le client, il envoie une erreur fatale «*unsupported_exception*» comme défini dans le RFC 3546. Afin de protéger le serveur contre certains risques (risque d'attaques, manque de mémoire, etc.), nous lui donnons le choix de refuser la demande du client en envoyant un nouveau message d'erreur non fatale «*server_politique_depended*».
Dans le dernier cas, le serveur peut demander la négociation du Handshake standard de TLS à la place de TLS-SIGN en envoyant au client le message *ServerHello*. Ici, le client peut refuser la connexion en envoyant le nouveau message d'erreur fatale «*signature_is_needed*» ou bien il continue le Handshake standard de TLS. Le serveur peut toujours arrêter le module de signature en envoyant un message d'erreur (ALERT) au client.
L'automate[10] suivant schématise ces cas pour le client et le serveur TLS.

[10] Note : le message ClientHelloSIGN signifie le message ExtendedClientHello avec l'extension de signature.

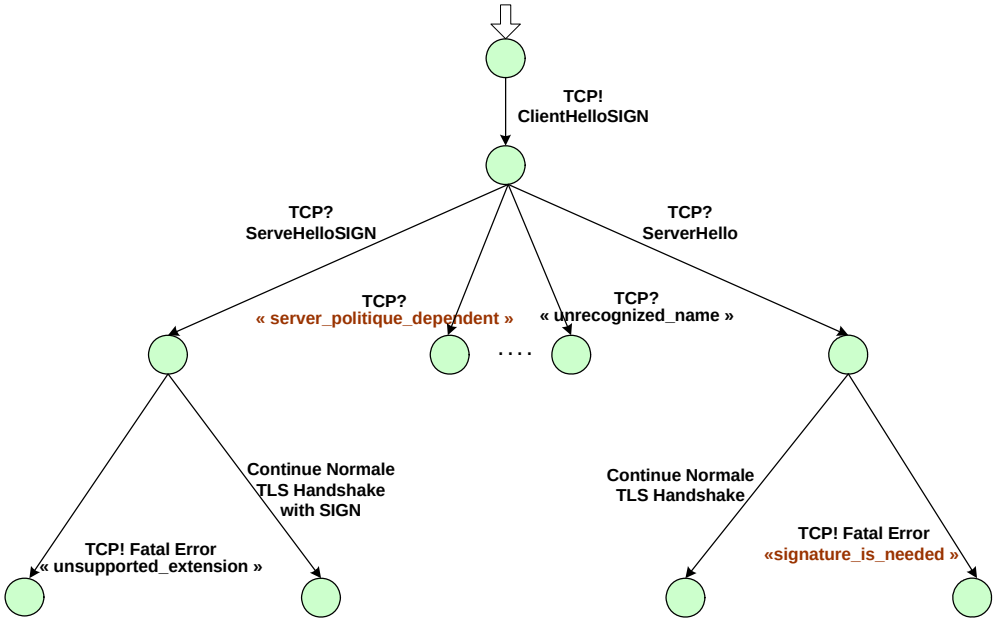


Figure 5-5. Automate d'un Handshake TLS avec l'extension de signature vue par le client

Notation	
TCP ! messageA :	Emission du messageA par le client
TCP ? messageB :	Emission du messageB par le serveur

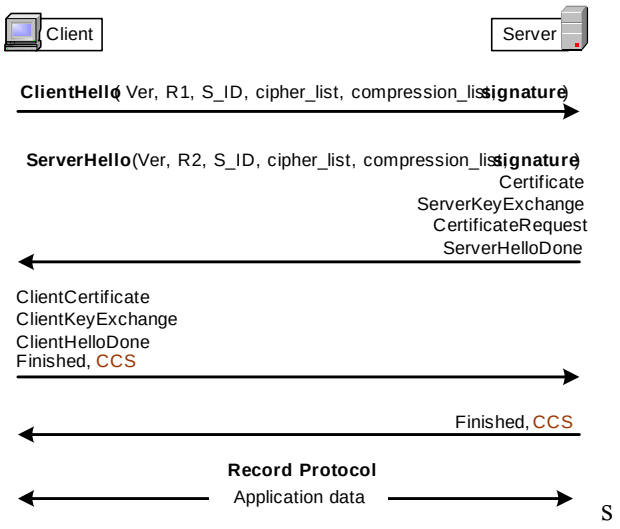


Figure 5-6. La négociation de TLS-SIGN

5.4.5.2 Négociation à travers une session SSL/TLS abrégée

La session abrégée (*Resumed Handshake*) est un échange rapide défini dans le protocole SSL/TLS pour réduire au minimum le nombre d'opérations cryptographiques ainsi qu'un nombre significatif de messages SSL/TLS.

Avec TLS-SIGN, la session abrégée sera également employée pour contrôler l'ouverture et la fermeture du service de non répudiation. Ainsi, un client TLS-SIGN peut demander au serveur de désactiver le service de non répudiation en envoyant le message *ExtendedClientHello.* Ce message contient la même extension de signature négociée dans l'ancienne session SSL/TLS à l'exception du champ Boolean qui sera initialisé à false pour indiquer la demande de désactivation du service de signature des données.

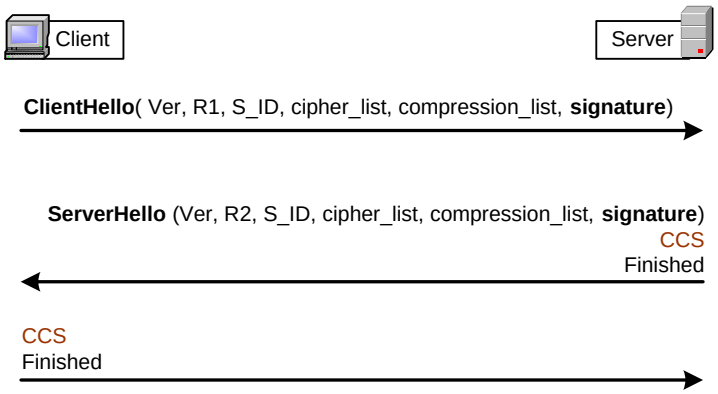


Figure 5-7. Session TLS-SIGN abrégée

5.4.6 TLS-SIGN et la couche Record de SSL/TLS

Après la phase de négociation du module TLS-SIGN, les données reçues de la couche applicative sont envoyées directement au module TLS-SIGN qui signe ces données, ajoute un nouvel en-tête et envoie le résultat à la couche Record.

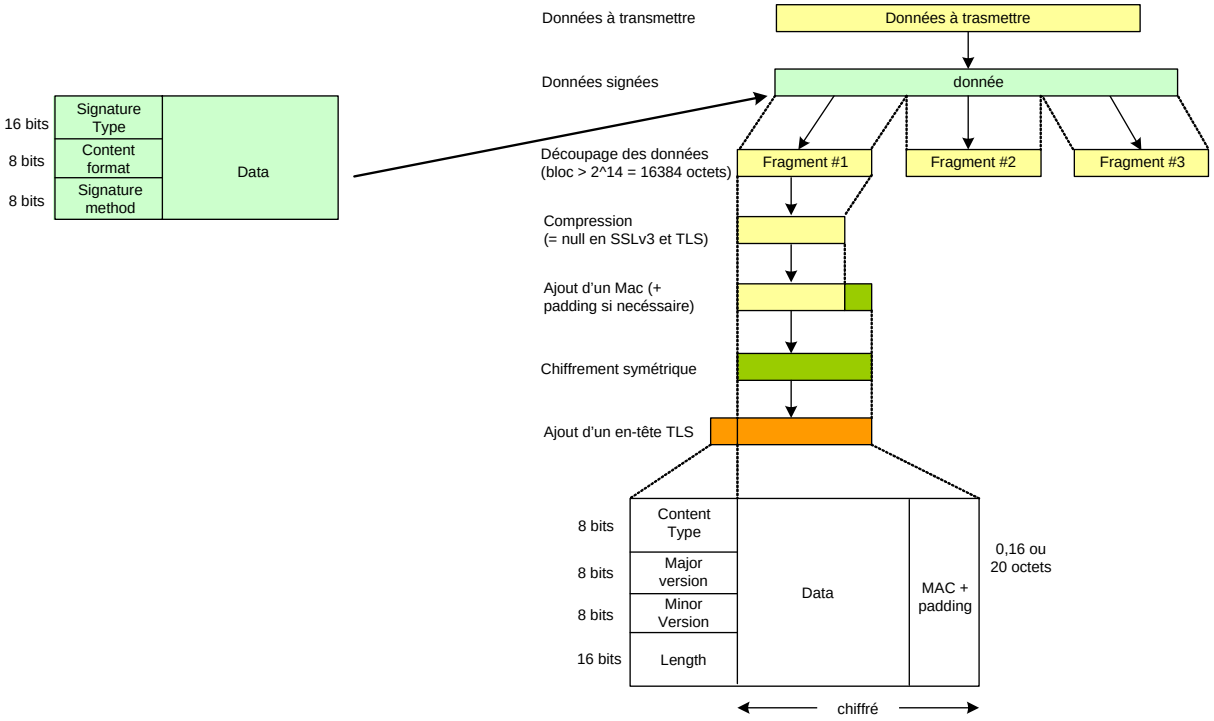


Figure 5-8. Le format du message TLS-SIGN et les étapes du protocole Record

En effet, si TLS-SIGN est activé, les données signées sont portées par un nouveau message (figure 5-8) avec un en-tête de 2 octets contenant : le type de signature (*sign-with_proof_of_origine, sign_with_double_proof, etc.*), le format du contenu (*signed_smime_file, signed_xml_dsig_data, etc.*) et la méthode de signature. Actuellement, la seule méthode proposée pour la signature est celle avec les certificats X.509 présentés pendant la phase d'initialisation de SSL/TLS.

TLS-SIGN envoie ces données à la couche Record qui fragmente, compresse, ajoute un MAC et chiffre les données. Finalement, il ajoute un en-tête et transmet le résultat sur un canal TCP fiable.

5.5 Scénario d'une application TéléTVA avec TLS-SIGN

Le procédé TéléTVA permet aux entreprises, préalablement dotées d'un certificat numérique, de remplir directement leurs obligations déclaratives (ou TVA) via Internet (figure 5-9). Dans un scénario de communication avec TLS-SIGN, un client A (le directeur de l'entreprise par exemple) se connecte à un site sécurisé avec SSL/TLS pour déclarer les impôts de son société. La première connexion ① (voir figure 5-9) du client peut se faire sans ou avec une authentification forte coté client.

Après la soumission de toutes les informations nécessaires, le client arrivera à une nouvelle page où le paiement devrait être confirmé ainsi qu'une signature explicite sur le formulaire du TVA.

C'est à ce point là, qu'une session TLS-SIGN est négociée ② entre les deux communicateurs en utilisant les deux messages génériques de TLS Extensions avec l'extension de signature expliquée dans les sections précédentes.

La signature ③ permet de protéger le destinataire confronté à un client niant avoir fait une demande d'achat. Il donne aussi au client une preuve juridique d'avoir payé ses obligations déclaratives.

La seule différence pour le client est que ce dernier doit saisir manuellement les paramètres suivantes : le format et le type de signature, l'URL d'une base de données LDAP et un mot de passe protégeant sa clé privée.

Maintenant, le formulaire est signé, stocké localement ④ ou chez un tiers ⑥ de confiance puis envoyé vers le serveur. A la réception de ces informations par le serveur, ce dernier vérifie la signature du client, stocke les données localement ⑤ ou chez le même tiers de confiance ⑦. Le serveur génère aussi une signature sur les données envoyées par le client si ce dernier demande une preuve de réception.

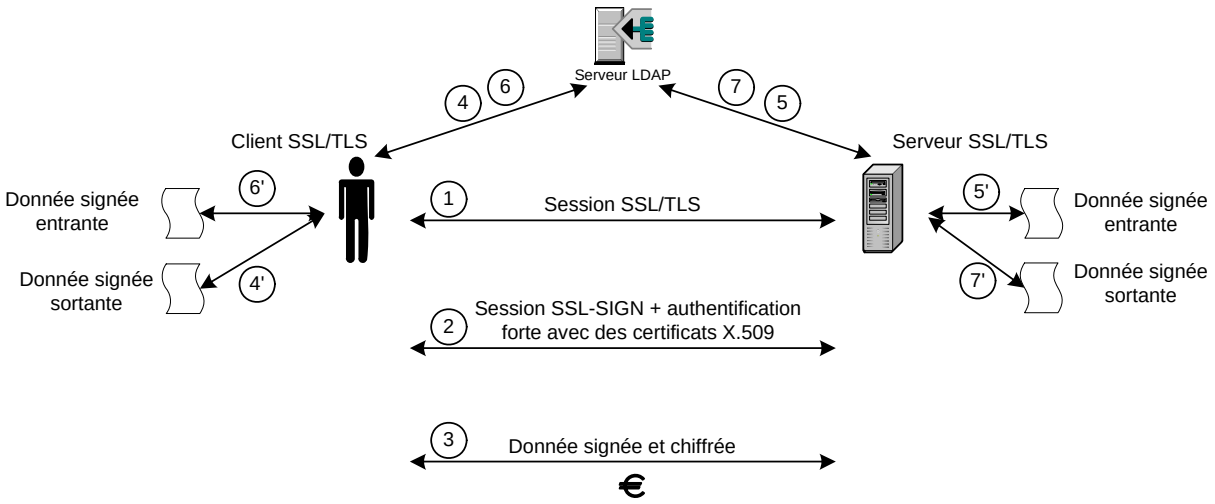


Figure 5-9. Phases de négociation et de stockage des données

Pour arrêter la génération des preuves, le client devrait envoyer le message *ExtendedClientHello* contenant la valeur *false* dans le champ *boolean*. Ce message est envoyé dans une session SSL/TLS abrégée et protégée en confidentialité.

La section suivante décrit en détail une implémentation de TLS-SIGN avec OpenSSL.

5.6 Implémentation

Cette section contient l'ossature d'une implémentation TLS-SIGN basée sur OpenSSL [OpenSSL]. OpenSSL est actuellement l'API la plus utilisée. Elle est du domaine public et les sources sont donc disponibles.

Dans les deux figures qui suivent, nous donnons notre méthode d'intégration de SSL_SIGN dans OpenSSL sous forme de deux fonctions *SSL_sign_write* et *SSL_sign_read* qui ressemblent au deux fonctions *SSL_read* et *SSL_write* utilisées par SSL/TLS.

```
Fd = socket( …)
Bind(fd)
Connect(fd)
SSL_library_init()
meth=[SSL|v2|v23|v3] | TLSv1]_client_method
Ctx=SSL_set_cipher_list(ctx,cipher)
SSL_CTX_load_verify_locations(ctx, CA_FILE,0)
SSL_CTX_set_verify(ctx,SSL_VEIFY_PEER, NULL)
Ssl=SSL_new(ctx)
SSL_set_fd(ssl,fd)
SSL_connect()
SSL_write | SSL_read | SSL_sign_write | SSL_sign_read | SSL_audit
SSL_shutdown(ssl)

Close(fd)
        SSL_free(ssl)
         SSL_CTX_free(ctx)
```

Figure 5-10. Exemple d'une application SSL/TLS coté client

```
Fd = socket(…)
Bind(fd)
Listen(fd)
        SSL_library_init()
meth=[SSL|v2|v23|v3] | TLSv1]_client_method
Ctx = SSL_CTX_new(meth)
SSL_CTX_set_cipher_list(ctx, cipher)
SSL_CTX_use_certificate_file(ctx,cert_file_type)
SSL_CTX_set_default_pqsswd_cb_userdata(ctx,pqssword)
SSL_CTX_use_PrivateKey_file(ctx,filename,filetype)
SSL_CTX_check_PrivateKey(ctx)
Accept(fd)
Fork() or thread_create()
                        ssl = SSL_new(ctx)
SSL_set_fd(ssl,newfd)
SSL_accept(ssl)
SSL_write | SSL_read | SSL_sign_write | SSL_sign_read | SSL_audit

Close(newfd)
SSL_free(ssl)
                SSL_CTX_free(ctx)
Close(fd)
```

Figure 5-11. Exemple d'une application SSL/TLS coté serveur

L'établissement d'un canal sécurisé avec SSL/TLS implique les étapes suivantes :

1. Etablissement d'une connection TCP avec le serveur qui implique un message aller-retour si aucune erreur ne se produit.

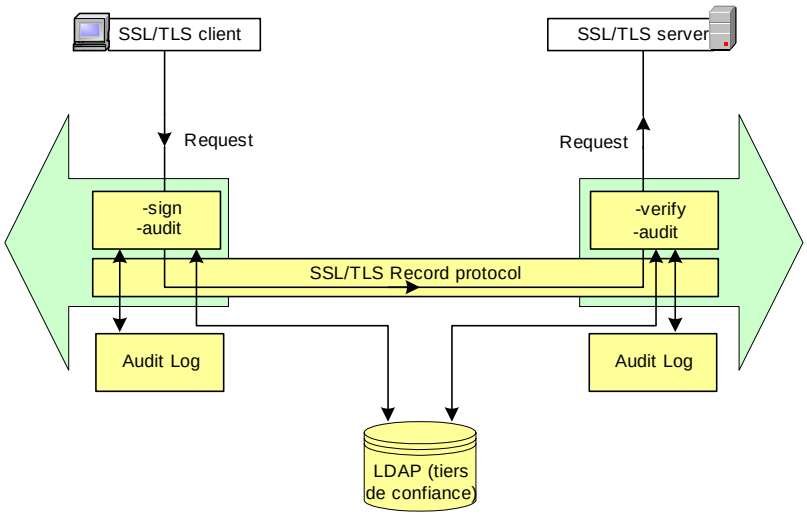


Figure 5-12. L'implémentation de TLS-SIGN avec OpenSSL

2. Sur le canal TCP, le client et le serveur établissent une voie de transmission sécurisée SSL/TLS. L'application fait appel à la librairie SSL *SSL_library_init()* pour ouvrir premièrement une Socket *SSL SSL_new()*. *SSL_new* est équivalente à l'appel *Socket()* de TCP.

3. L'application utilise *SSL_connect()* pour établir une connexion SSL/TLS avec le serveur. Dans le Handshake de SSL/TLS, le client et le serveur négocient les algorithmes de chiffrement, de hachage et la méthode d'authentification en faisant appel à *SSL_set_cipher_list()*. Le client et le serveur utilisent un chiffrement à clé publique pour échanger les clés secrètes de session qui seront employées pour chiffrer et déchiffrer les messages de la couche application.

4. Sur le canal SSL/TLS, le client et le serveur peuvent échanger un ou plusieurs messages (message HTTP, FTP, etc.). Les applications utilisent pour cela les deux fonctions *SSL_write(..)* et *SSL_read(..)* à la place des appels au *write()* et *read()* de TCP.

5. Si le module de signature TLS-SIGN est activé, les applications doivent utiliser les deux fonctions *SSL_sign_write(…)* et *SSL_sign_write(…)* au lieu des appels *SSL_write(..)* et *SSL_read(..)*. En plus, avec le module de signature une nouvelle fonction *SSL_audit(…)* doit être aussi activée afin de stocker les informations signées localement ou dans une base de données LDAP. Les deux fonctions *SSL_sign_write() et SSL_sign-read()* possèdent comme entrée les données à signer, sa longueur, le type de signature ainsi que le format de signature utilisé. Par exemple, les entrées de la fonction SSL_sign_write sont :

   SSL_sign_write ("donnée à signer", strlen("donnée à signer"), "signature_with_double_proof", "XML-DSIG").

   Pour la fonction *SSL_audit(),* elle prend comme entée respectivement les trois valeurs suivantes : l'URL de la base de données LDAP d'un tiers de confiance, l'URL de deux

fichiers log locaux (entrant et sortant) et *SSL_audit*(null, null, C:\ssl-sign\entrat\s_id1.log, c:\ssl_sign\sortant\s_id1.log).

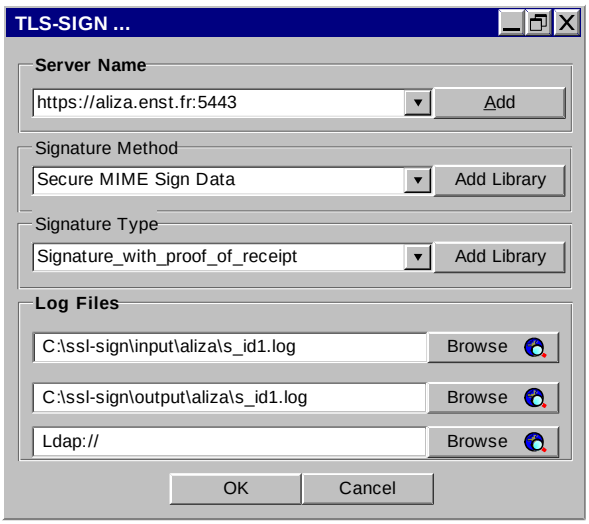


Figure 5-13. Interface de configuration de TLS-SIGN

6. Pour fermer la connexion, l'application peut utiliser les fonctions *SSL_shutdown()* et *ssl_free()*.

5.7 Conclusion

Dans ce chapitre, nous avons présenté une architecture d'intégration d'un service de signature dans le protocole SSL/TLS dans le but de protéger les transactions Internet avec un service générique de non répudiation.
Contrairement à l'approche qui consiste à limiter le service de non répudiation aux applications, notre approche présente plusieurs avantages. Ces derniers concernent la protection à long terme des données et la génération des preuves génériques de non répudiation.
Il est à noter que nous avons implémenté un premier service TLS-SIGN en utilisant la bibliothèque de sécurité *GNUTLS* [GNUTLS] qui supporte le standard TLS Extensions.
Dans le souci d'offrir des fonctionnalités encore plus performantes, nous visons d'ajouter à notre prototype les points suivants:

1. Afin de faciliter son déploiement, notre architecture devrait s'intégrer aux navigateurs Web tels que le navigateur Web de Mozilla.
2. Notre architecture doit être étendue afin de s'interfacer avec les certificats d'attribut. Avec le service TLS-SIGN, les certificats d'attribut seront employés dans le contrôle d'accès à travers des mécanismes de délégation de rôles et de signature.

Troisième Partie

SEP, un protocole de sécurité au niveau d'échange : conception et validation.

Chapitre 6

6 Le protocole SEP (Secure and Extensible Protocol)

6.1 Préambule

Nous avons choisi pour ce chapitre un style de rédaction assez détaillé du protocole SEP. Ceci pour deux raisons. La première est de permettre une meilleure maîtrise pour les implémenteurs. La seconde raison est pour offrir une meilleure pédagogie aux lecteurs. Néanmoins, certaines parties fastidieuses sont reportées au niveau de l'annexe C. Notamment la partie qui décrit les unités de protocoles de données en XDR (External Data Representation standard) [RFC1832] et la déscription en XML-DTD (Extensible Markup Langage - Document Type Definition) [W3Cdtd] de la base de données des politiques de sécurité.

6.2 Résumé

Dans ce chapitre, nous présentons un nouveau protocole de sécurité appelé SEP (Secure and Extensible Protocol). Ce protocole rassemble les avantages de plusieurs solutions de sécurité dans une architecture dynamique et robuste.
A travers un mécanisme de négociation qui intègre une autorité de confiance intermédiaire (IA), SEP supporte quatre architectures s'adaptant aux besoins spécifiques des utilisateurs et des applications.
Nous montrons également que SEP supporte la négociation de plusieurs services optionnels tels que le PFS (Perfect Forward Secrecy), la mise en œuvre de VPN et le SSO (Single Sign On).

6.3 Introduction

Le rôle joué par l'Internet dans les échanges électroniques a considérablement évolué depuis quelques années, créant de nouvelles opportunités et également de nouveaux besoins en terme de sécurité. Ainsi, étant donné qu'il est impossible de maîtriser l'ensemble des infrastructures physiques situées entre l'utilisateur et la machine distante, et après l'étude de plusieurs protocoles de sécurité dans cette thèse, on constate que l'accent est mis de plus en plus sur les protocoles au niveau applicatif. Ces protocoles sont sollicités non seulement pour assurer une sécurité de bout en bout des transactions mais également pour leurs mécanismes flexibles d'authentification et de contrôle d'accès. Cela traduit le fait que la sécurité est traitée de bout en bout et doit pouvoir être adaptée en fonction des types d'information.

Cependant, si SSL/TLS est largement étudié à travers l'état de l'art de cette thèse et à travers nos deux contributions (chapitre 4 et 5), il reste cerné par un problème d'interopérabilité avec

sa version standard largement déployée. Ce qui rend la satisfaction des besoins définis dans le chapitre 1 irréalisable.

Dans ce chapitre nous proposons un protocole de sécurité au niveau de données. Le protocole SEP (Secure and Extensible Protocol) garantit plus de sécurité tant pour le transfert des données que pour l'authentification du client et du serveur. Il fournit également plusieurs services tels que la protection de l'identité des usagers, le contrôle d'accès, le traçage des transactions, et la renégociation des clés.

6.4 Scénario d'utilisation de SEP : accès à distance sécurisées

Les réseaux privés virtuels (ou VPN) sont utilisés pour permettre à des utilisateurs itinérants d'accéder au réseau privé. L'utilisateur se sert alors d'une connexion Internet pour établir la connexion VPN. SEP es un protocole qui permet la négociation d'un tunnel sécurisé entre des clients nomades et des serveurs d'applications placés derrière des passerelles d'accès SEP (figure 6-1). Ceci permet à notre protocole de fournir un VPN au niveau d'échange qui possède les caractéristiques suivantes :

1. L'anonymat du client vis-à-vis des entités non autorisées.
2. Le multiplexage de plusieurs applications dans le même tunnel SEP.
3. La centralisation de l'authentification du client au niveau d'un intermédiaire de confiance SEP.
4. Le contrôle d'accès et le filtrage des paquets aux extrémités des réseaux.

Dans ce paragraphe, nous présentons un exemple illustrant l'établissement d'une session SEP. Dans la suite de ce chapitre, nous détaillons d'autres architectures de communication supportées par SEP.

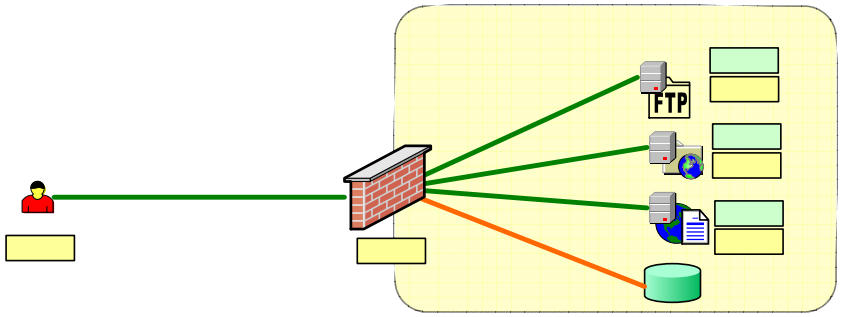


Figure 6-1. Accès distant sécurisé au réseau d'une entreprise avec SEP

Supposons qu'un client A souhaite accéder aux deux serveurs B (serveur Web avec le port 80) et C (serveur de messagerie avec le port 25). L'accès au serveur B nécessite une authentification par certificat X.509 alors que l'accès au serveur C nécessite une authentification par un nom et un mot de passe (figure 6-1).
Le client A initie une session SEP vers un des serveurs internes (par exemple B) tout en précisant l'ensemble des services qu'il souhaite accéder (Web et messagerie). Une passerelle P située du côté des serveurs intercepte la demande du client et interroge une base de données centralisée pour donner au client le niveau de sécurité proposé par chaque serveur d'application interne. La réponse de la passerelle inclut les choix cryptographiques qu'elle souhaite appliquer pour protéger les trafics de données, les méthodes d'authentification, etc.

Le client reçoit alors une réponse SEP venant d'un tiers (passerelle P) qui ne l'a pas directement interrogé. Pour cela, la passerelle P ne doit pas seulement prouver son identité auprès du client mais aussi la délégation faite par les serveurs internes qui lui permettra d'authentifier les clients. La passerelle présente un certificat d'attribut (cf. §6.11.6.3) contenant la délégation faite par B et C ainsi que le certificat X.509 (ou son hachage) qui lui appartient.

Nous supposons dans SEP que les certificats X.509 sont déjà distribués dans la phase de distribution des certificats. En effet, le client peut récupérer les certificats d'une base de données LDAP ou à travers un message SEP spécifique.
Ensuite, le client et la passerelle utilisent un mécanisme inspiré de SSL/TLS pour l'échange et la génération d'une clé de chiffrement et un ensemble des clés dérivées. Si l'authentification est de bout en bout, la passerelle P transmet l'ensemble des paramètres session au serveur interne en utilisant une ancienne session SEP ou bien avec un mécanisme de chiffrement asymétrique.

A ce stade là, A, P, B et C possèdent la clé de chiffrement et un ensemble de clés dérivées. Sous un tunnel sécurisé, le client procède à plusieurs authentifications selon l'ordre des ressources demandées dans le premier échange. Il présente son certificat X.509 puis son mot de passe. Notons que si par exemple les serveurs B et C exigent une authentification par certificat, le client aurait dû s'authentifier une seule fois.
Une fois les authentifications terminées, la passerelle signale aux serveurs pour qu'ils activent leurs services de confidentialité et d'intégrité de données.
Ainsi, le client peut envoyer toutes ces données (Web et messagerie) dans le même tunnel SEP établi au dessus du protocole de transport TCP. Le choix de TCP est lié seulement aux deux applications Web et Messagerie. Puisque nous proposons qu'un port soit dédié à notre protocole (le port 300), un mécanisme de port forwarding sera utilisé entre les bouts de la communication SEP.
Notons finalement que tout le trafic SEP qui passe par P sera contrôlé au niveau des adresses IP, et des en-têtes SEP qui identifient les sessions SEP et les types des applications. Dans la suite, la passerelle SEP sera représentée comme une autorité intermédiaire ou (IA)

6.5 Processus de conception

La conception d'un protocole de sécurité nécessite une détermination des besoins et des exigences que nous voulons couvrir dans ce protocole. Pour cela, il faut planifier, organiser, diriger et contrôler la conception et le développement de ce protocole afin de l'orienter vers les exigences souhaitées.

La figure 6-2 présente une série de cercles concentriques correspondant aux différents stades du cycle d'évaluation de SEP. Lorsqu'on projette une modification importante à un stade, il faut revenir au stade central, celui de la planification. La représentation du cycle de vie sous forme d'une spirale, permet d'indiquer un nombre arbitraire de stades. Ainsi, la conception du protocole SEP se compose de six stades (ou étapes) (figure 6-3) :

1. *Planification : Evaluation des buts et description des besoins*

La planification comprend les étapes suivantes:

- Identifier les buts;
- Déterminer les solutions existantes de sécurité;

- Recueillir des renseignements (sur les défaillances et les vulnérabilités relevées dans le passé);
- Décrire les nouveaux besoins;

Après avoir établi les buts à atteindre, il faudra examiner l'ensemble des solutions existantes afin de déterminer l'avantage et l'inconvénient de ces solutions ainsi que les besoins non couverts par ces solutions. Cette analyse doit montrer s'il est nécessaire de développer une nouvelle solution ou un nouveau protocole de sécurité.

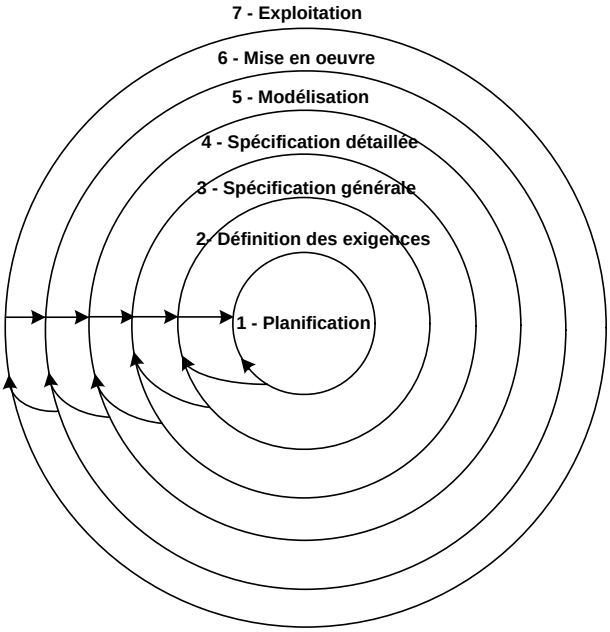


Figure 6-2. Le cycle d'évaluation du protocole SEP

2. Définitions des exigences

L'analyse des besoins non couverts par les solutions actuelles de sécurité forme la base pour une définition plus claire des exigences que nous souhaitons satisfaire dans notre protocole de sécurité SEP. Ce stade du cycle de vie est effectué au niveau fonctionnel. Il faut saisir les besoins réels des applications, des programmeurs, des administrateurs réseaux ainsi que des utilisateurs finaux d'une manière simple et sans avoir besoin des analyses techniques détaillées.

3. Spécification générale : Architecture et différents services offerts par SEP

Au cours du stade des spécifications générales, on voudrait essentiellement :

- Sélectionner le niveau de protection offert.
- Exposer le principe de fonctionnement.
- Présenter les différents services de base ou optionnels.
- Sélectionner les méthodes cryptographiques adaptées à ce protocole ainsi que les mécanismes d'authentifications et de partage des clés.

Dans le cas de définition d'une solution étendue de sécurité, les options étudiées peuvent correspondre à des définitions simples et lisibles de son architecture et de son mécanisme de fonctionnement.

4. Spécification détaillée : Architecture et services de sécurité

C'est à ce stade que se fait la conception de l'architecture du protocole. Après avoir examiné les spécifications générales et définit les exigences, il faut produire des spécifications détaillées du protocole de sécurité. Au cours de ce stade, on voudrait essentiellement :

- *Etablir l'architecture du protocole ainsi que ses différents modules*
  L'architecture d'un protocole de sécurité comprend une schématisation des différentes composantes de base (ou optionnelles) de ce protocole ainsi que l'exposition des services apportés par chaque module.
- *Définir les différents types d'échange de ce protocole*
  Il faudra dans cette étape établir une spécification détaillée des différents types de négociation utilisée par ce protocole.
- *Définir des scénarios de communication.*
  L'élaboration des scénarios et des architectures réelles permet de mettre en œuvre et de déployer ce protocole de sécurité.

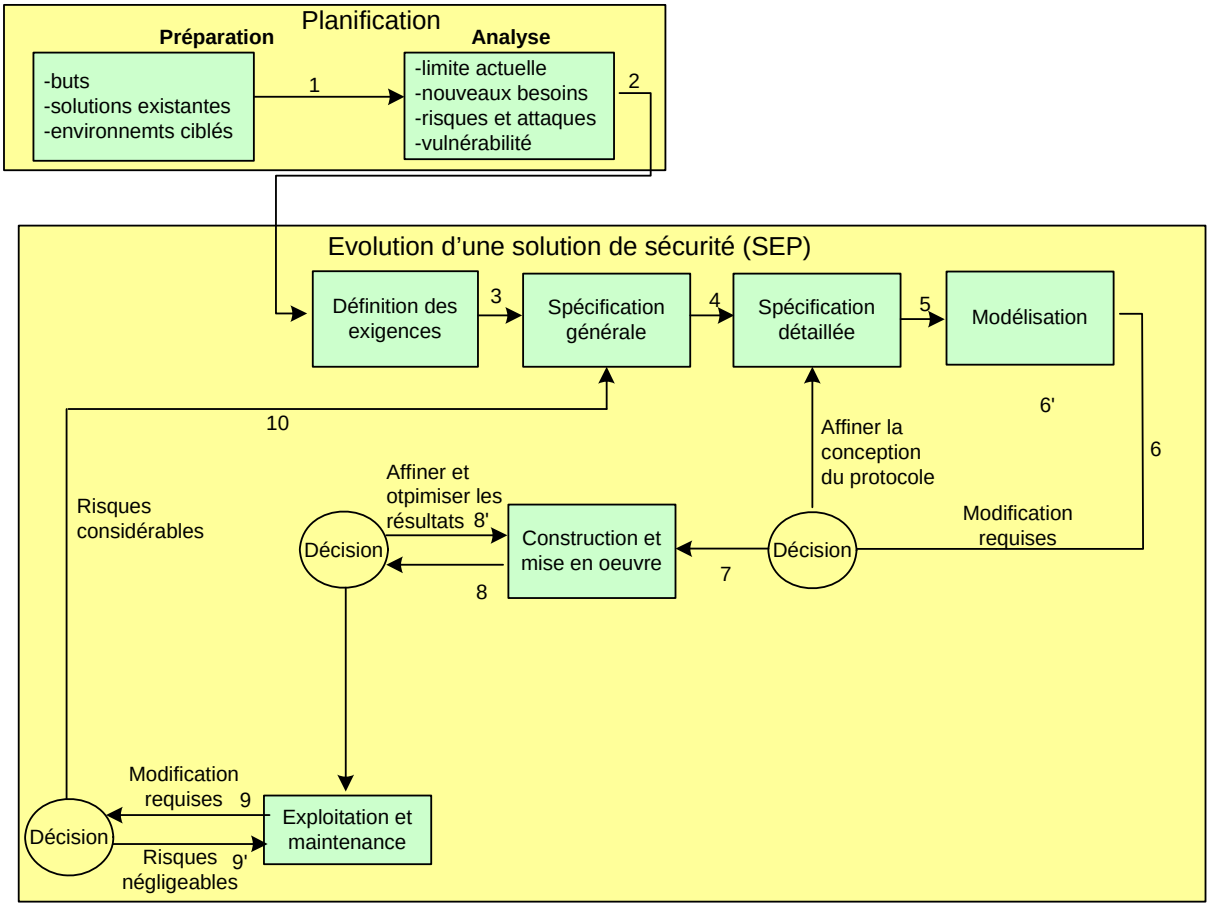


Figure 6-3. Processus d'évaluation du protocole SEP

5. Modélisation : Modélisation du protocole de sécurité

A ce stade, nous devons utiliser un langage formel pour modéliser notre protocole de sécurité. La modélisation doit nous permettre:

- d'évaluer la performance et la capacité de ce protocole.
- de valider cryptographiquement la phase d'initialisation de ce protocole ainsi que la robustesse des secrets générés.

Ce stade amène à prendre des décisions afin de stabiliser tous les scénarios envisagés dans le stade de conception détaillée du protocole. Une fois que la modélisation rend ce protocole

stable, nous pouvons continuer vers le stade de construction et de mise en œuvre de notre protocole.

6. *Mise en oeuvre : Tests d'évaluation du protocole*

Du point de vue de l'ensemble de l'architecture, les tests et l'évaluation du protocole garantissent que les fonctionnalités commandées au stade des spécifications détaillées répondent au cahier de charges établi. Au plan de la sécurité, on soumet le protocole à des tests de vulnérabilité contre certains types d'attaques, afin de s'assurer que les exigences de la sécurité précisées dans les spécifications détallées sont satisfaites.
Dans ce stade là, nous choisissons le langage de programmation et les différents logiciels nécessaires pour implémenter ce protocole.

7. *Exploitation : Exploitation et maintenance*

Pour exploiter un protocole de sécurité, il faut revoir sans cesse la sécurité globale de l'infrastructure. L'objectif de sécurité visé au cours du stade d'exploitation du protocole consiste à maintenir les risques résiduels surtout ceux liés au réseau public à un niveau acceptable.
L'exploitation de ce protocole amène à prendre des décisions concernant la satisfaction de toutes les exigences de sécurité qui ont été attribuées. Le résultat de cette évaluation peut servir à affiner ce protocole ou bien à faire des modifications qui peuvent atteindre le stade de la spécification générale du protocole.

6.6 Les exigences

Lorsqu'on parle de la sécurité des données, on évoque généralement les protocoles de sécurité et les exigences ou services qu'assurent ces protocoles (tels que la confidentialité, l'intégrité et l'authentification). Dans ce contexte, l'aspect de définition des exigences apparaît important, notamment la définition des capacités et des limites du protocole SEP, mais aussi quelques services optionnels que ce protocole peut satisfaire.
Dans cette section, nous rappelons les principales exigences décrites précédemment et qui nous ont conduits à la conception de SEP. Nous les détaillons après dans les paragraphes 6.7 et 6.8 de ce chapitre. Les principaux besoins et exigences pour la conception de SEP sont les suivants:

- Simple négociation de nouveaux services.
- Intégration des fonctionnalités des protocoles de sécurisation des réseaux et des échanges
- Protection d'identité de l'initiateur vis-à-vis des personnes non autorisées.
- Mise en œuvre d'un VPN.
- Multiplexage de plusieurs types de trafic.
- Support de plusieurs méthodes d'authentification.
- Négociation des paramètres de sécurité pour chaque service.
- Centralisation des politiques de sécurité.
- Audit et traçage des transactions.
- Intégration d'un mécanisme de délégation des rôles.
- Utilisation des algorithmes de chiffrement robustes et fiables.

6.7 Les participants

Le protocole SEP comprend plusieurs participants nécessaires à son bon fonctionnement. Ces participants sont : le client, le serveur origine et l'autorité intermédiaire (figure 6-4). Nous décrivons dans ce qui suit les fonctionnalités de chacun de ces participants.

1. *Le client SEP*

Généralement, le client est une simple entité qui demande l'ouverture d'une session SEP pour accéder à certaine application ou ressource en utilisant le protocole d'authentification et de négociation des services (ASNP) de SEP.

2. *Le serveur origine SEP*

Le serveur origine est le destinataire final dans une session SEP. Généralement, un serveur d'origine fournit l'accès à une ou plusieurs applications à travers une connexion directe avec l'émetteur ou bien à travers une autorité SEP intermédiaire (IA). L'accès aux applications se fait suivant des politiques de sécurité définies par chaque application.

3. *L'autorité intermédiaire*

L'autorité intermédiaire (IA pour Intermediate Authority) ou le médiateur est un serveur de confiance placé du coté des clients ou des serveurs SEP. Ce serveur est utilisé pour le contrôle d'accès, l'authentification et le filtrage des transactions SEP.

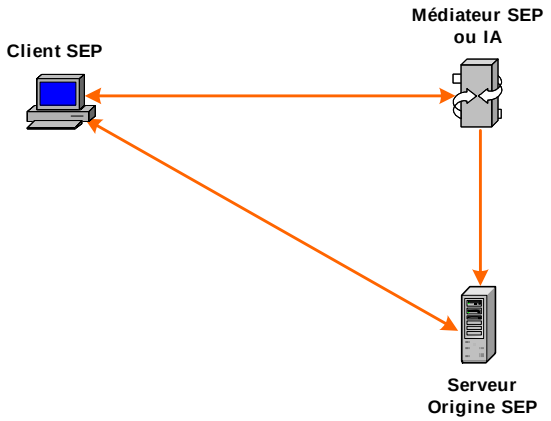


Figure 6-4. Eléments de la sécurité du protocole SEP

Si la IA SEP est placée du coté des clients, elle fournit un ou plusieurs canaux sécurisés établis avec des serveurs ou des IAs SEP externes. Elle assure les tâches d'authentification et de négociation des paramètres de sécurité à la place des clients SEP

Si la IA SEP est placée du coté des serveurs, elle fournit un point d'accès sécurisé vers les réseaux internes des entreprises et permet un mécanisme d'autorisation et de filtrage à plusieurs niveaux de la couche OSI.

Une IA SEP est *active* si le serveur origine lui a déléguée le rôle d'authentification vis-à-vis des utilisateurs. On dit qu'une IA SEP est *passive* si elle reste transparente pour toute négociation SEP entre les deux bouts de la communication.

6.8 Les architectures de communication dans SEP

Entre les trois participants à SEP (client, IA et serveur origine), il existe plusieurs types de communication qui permettent à SEP de fournir les fonctionnalités des protocoles de

sécurisation des réseaux et des échanges. Les quatre architectures de communication dans SEP sont :

1. *Connexion client - serveur*

La connexion client/serveur (figure 6-5) correspond à une communication directe entre un client SEP et un serveur origine SEP sans passer par une IA intermédiaire. Le client se connecte directement au serveur origine SEP en précisant l'ensemble de ressources auxquelles il souhaite accéder. Dans ce cas, l'authentification, la confidentialité et l'intégrité sont assurées de bout en bout.

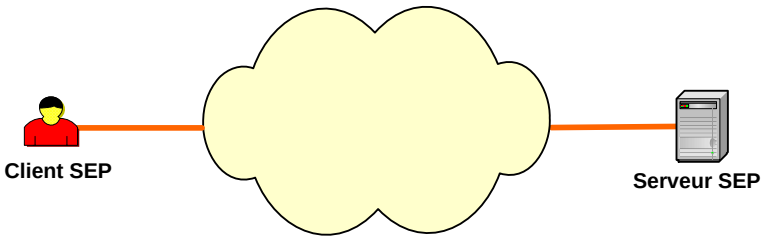


Figure 6-5. Connexion client-serveur

2. *Connexion client ---- IA - serveur*

Dans la deuxième architecture (figure 6-6), les serveurs origines, sont placés derrière une autorité intermédiaire SEP (IA). La IA joue le rôle d'un point d'accès sécurisé vers des services internes hébergés sur un ou plusieurs serveurs origine SEP. La IA aura pour rôle d'authentifier les clients SEP et de négocier l'ensemble des paramètres de sécurité.

Pour authentifier les clients, la IA SEP dispose d'un certificat d'attribut délégué par les serveurs origines. Ce certificat d'attribut donne à l'IA l'habilitation à exercer le rôle d'authentificateur vis à vis des clients SEP.

Dans ce cas, c'est seulement le client et la IA SEP qui s'authentifient. La confidentialité et l'intégrité des données peuvent être de bout en bout ou bien entre le client et l'autorité intermédiaire SEP.

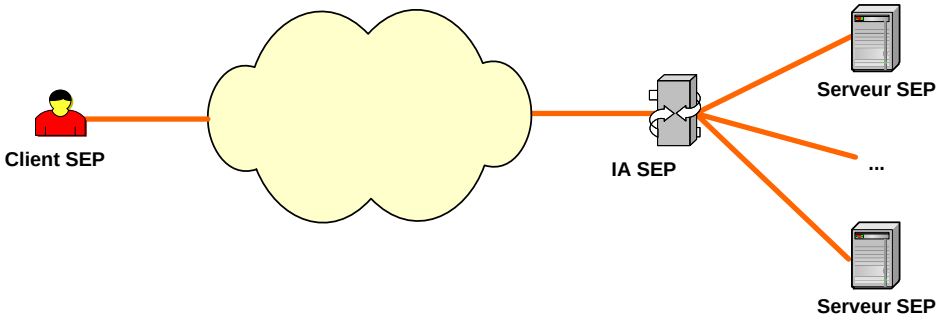


Figure 6-6. Connexion client----IA-serveur

3. *Connexion client - IA ---- serveur*

L'architecture ci-dessous (figure 6-7) est le symétrique du cas précédent. La IA joue le rôle d'un Proxy proche des clients SEP pour fournir un service de contrôle d'accès et d'authentification avec les réseaux externes. La IA peut être utilisée par les administrateurs pour le traçage des connexions et le filtrage des données sortant des entreprises.

Ainsi, si un client souhaite se connecter à un serveur SEP externe, c'est à l'IA SEP de s'authentifier et d'établir un tunnel sécurisé entre le serveur origine et elle-même ou bien directement avec le client interne.

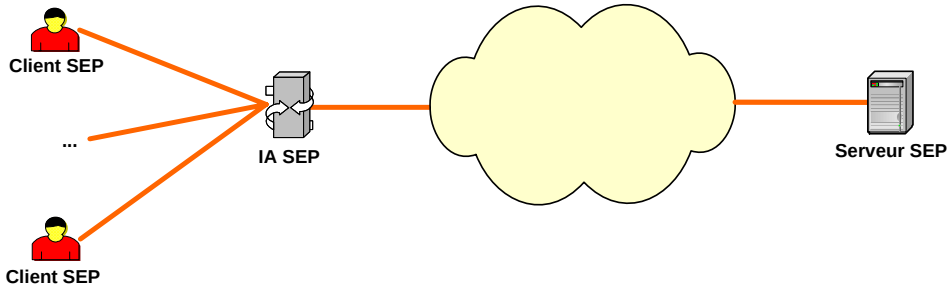


Figure 6-7. Connexion client-IA----serveur

4. Connexion client - IA1 ----- IA2 - serveur

La quatrième architecture (figure 6-8) présente la mise en œuvre d'un tunnel SEP entre deux réseaux différents. Deux autorités intermédiaires sont placées de part et d'autre du réseau Internet. A la demande d'un client SEP, la IA1 et la IA2 vont s'authentifier et établir un tunnel SEP entre elles. Ce tunnel est soit de bout en bout (entre le client et le serveur), soit entre la IA1 et la IA2.

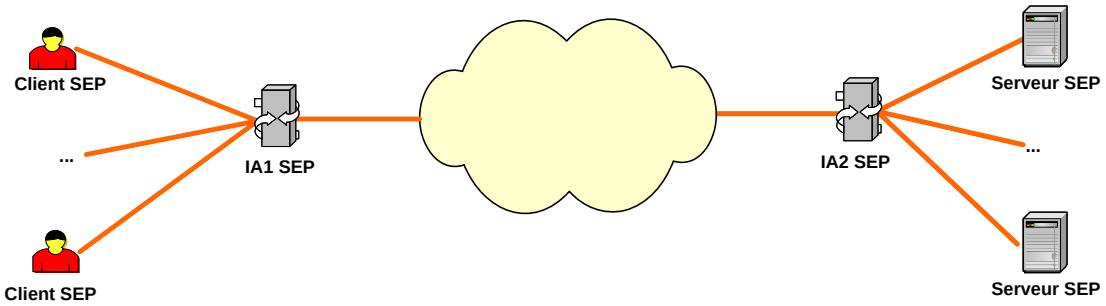


Figure 6-8. Connexion client-IA1----IA2 - serveur

6.9 *Les services fournis par SEP*

1. L'authentification des communicateurs

SEP permet une authentification des divers communicateurs. Dans le cas ou une IA est *active*, cette dernière présente un certificat d'identité et un certificat délégué par le serveur origine lui donnant le droit d'authentifier les utilisateurs. Si la IA SEP est *passive*, c'est le serveur origine qui s'authentifie avec un certificat d'identité auprès des clients. L'authentification par certificat d'identité est toujours accompagnée d'une signature sur l'ensemble des paramètres échangés.

Les clients SEP ont le choix entre plusieurs méthodes d'authentification. Une particularité de SEP est qu'il permet au client durant une même session, d'avoir plusieurs authentifications : par exemple, une authentification par certificat X.509 puis une deuxième par un mot de passe. Ceci afin d'éviter la négociation d'une nouvelle session SEP pour accéder à de nouvelles ressources.

2. La confidentialité de l'information

Les informations transitant entre les entités sont sécurisées durant le transport à travers le réseau. Avec SEP, la confidentialité peut être maintenue de bout en bout entre l'émetteur et le

destinataire, même dans le cas de l'utilisation d'un serveur intermédiaire. Afin de fournir la confidentialité, nous utilisons le chiffrement conventionnel par AES ou 3DES.

3. L'intégrité des données

SEP garantit que le contenu du message n'est pas altéré pendant le transport. En effet, les messages SEP sont protégés par un HMAC, associé à SHA-1 ou MD5.
Avec SEP, les algorithmes de chiffrement, de hachage et de signatures sont négociés durant le premier échange SEP pour chaque service négocié.

4. Non rejeu

L'anti-rejeu (parfois appelé « intégrité de séquence ») est assuré dans toutes les configurations SEP grâce à un compteur Sequence Number (SN) placé dans l'en-tête des messages SEP et protégé en intégrité.
Dans le cas d'utilisation du protocole de transport TCP, le SN est concaténé avec les autres données protégées en intégrité dans le champ MAC des messages SEP.
Dans le cas d'utilisation du protocole de transport UDP, le SN est géré de la même manière suivante que celle d'IPSec ESP. Ainsi, le premier datagramme d'un flux entre deux correspondants SEP, possède un SN égal à 1. A chaque nouveau datagramme émis par une pile SEP, le SN est incrémenté de 1. A la réception d'un message, SEP gère une fenêtre glissante (de 32 ou 64 positions par exemple) permettant de préciser les numéros de séquence déjà reçus dans cette fenêtre. L'algorithme de détection du rejeu est le suivant :

- Un datagramme avec un SN plus petit que la borne inférieure de la fenêtre est détruit;
- Un datagramme avec un SN déjà validé dans la fenêtre est un rejeu et doit être traité en conséquence (alerte à l'opérateur, journalisation,…);
- Un datagramme avec un SN dans la fenêtre mais non encore validé est accepté;
- Un datagramme avec un SN dépassant la borne supérieure de la fenêtre provoque le glissement de celle-ci vers ce nouveau SN.

Le fait d'ignorer les datagrammes dont le SN est en deçà de la fenêtre peut entraîner des réémissions de datagrammes gérées par TCP (mais pas par UDP, pour lequel les datagrammes sont définitivement perdus).

5. La mise en œuvre d'un VPN

La principale caractéristique de SEP est qu'il permet la mise en œuvre d'un VPN au niveau d'échange entre un client et un serveur destinataire à travers une autorité intermédiaire. En effet, les paquets SEP sont identifiés par un en-tête qui permet de multiplexer les données de plusieurs applications et de plusieurs clients. Ceci grâce à un double identificateur session (GSID et LSID) placé dans les en-têtes SEP ainsi qu'un champ (Protocol Type) qui contient le type de donnée négocié (HTTP, FTP, ASNP, etc.).
Les applications comme HTTP et FTP peuvent bénéficier des services de sécurité de SEP sans avoir à changer leur port. En effet, la même technique de port forwarding utilisée avec SSH va permettre à SEP de multiplexer plusieurs types de trafic sans avoir à changer les ports de chaque application. De ce fait, SEP devient un protocole transparent aux applications et qui ne nécessite aucune intégration des modules de sécurité au niveau applicatif.

6. Le service de Single Sign On ou SSO

Le service de Single Sign On (SSO) est proposé récemment comme un nouveau service dans de nombreux protocoles comme le protocole SAML [SAML] pour les Web Services et le protocole SSH [SSH-Arch]. Même si la notion de Single Sign On diffère d'un protocole à un

autre (SSH utilise la notion d'un agent SSH pour transporter la clé publique du client entre les serveurs alors que SAML est basé sur une architecture centralisée), ce service est indispensable pour notre protocole où les différentes applications peuvent être cachées derrière une IA qui oriente après les requêtes client vers le service demandé. Ainsi, l'autorité intermédiaire IA est une entité centrale pour appliquer le service SSO entre un client qui demande l'accès à plusieurs ressources distribuées sur plusieurs serveurs SEP et possédant tous la même méthode d'authentification. Pour cela, le nombre d'opérations d'authentification d'un client durant une session SEP est relatif au nombre d'opérations d'authentification distinctes. Par exemple, si le client souhaite accéder à 4 serveurs dont 3 possèdent la même méthode d'authentification, le client est obligé de présenter seulement deux identités.

7. *Le contrôle d'accès*

Grâce à l'intégration d'un intermédiaire de confiance (IA) dans l'architecture et les échanges SEP, la IA de SEP fournit un point unique et stratégique où l'audit et le contrôle d'accès peuvent être imposés. Ces IAs bénéficient également d'une base de données relationnelle qui centralise les différents types d'accès et les règles de sécurité nécessaires pour chaque type d'application.

6.10 Les caractéristiques de SEP

Pour couvrir les exigences précédemment décrites, SEP présente les caractéristiques suivantes:

1. *La protection d'identité*
   - *Cacher l'identité de l'usager vis-à-vis d'un tiers non autorisé*

Le protocole SEP assure ce besoin. L'identité du client est liée à sa méthode d'authentification. A titre d'exemple, dans le cas d'une authentification par des certificats X509, l'identité du client peut être un nom, un email ou même un certificat d'attribut.

- *Cacher l'identité de l'usager vis-à-vis d'une autorité SEP intermédiaire*

Pour plusieurs raisons, un client SEP peut demander l'établissement d'une session SEP sécurisée directement avec le serveur origine. Si la politique de l'entreprise et la politique du serveur origine permettent une telle méthode, l'authentification du client se fait directement avec le serveur origine. En effet, dans ce cas, la IA se comporte comme une entité *passive* qui déroute la demande du client vers le serveur origine demandé dans la première connexion.

2. *Le service de délégation des rôles*

Avec SEP, les clients peuvent se connecter directement au serveur origine sans savoir que leur trafic passe par une IA SEP. Cependant, si cette dernière est *active*, les clients sont dans l'obligation de prouver leur identité auprès d'un tiers (le médiateur IA) inconnu. Ainsi, un malveillant peut jouer le rôle de cette IA d'authentification avec un simple détournement du trafic IP.
Pour empêcher un tel type d'attaque, la IA SEP présente un certificat délégué par les serveurs origine pour pouvoir exercer le rôle d'authentification auprès des utilisateurs SEP.

3. *Centralisation de politiques de sécurité*

La définition des politiques de sécurité et la gestion du contrôle d'accès sont toujours un problème dans les protocoles de sécurité. SEP propose un mécanisme centralisé pour la

gestion de tous les paramètres de sécurité. Ceci est réalisé à travers un service de base appelé DBP. Ce service contient :

- Une politique d'accès défini par chaque application.
- Un ensemble de paramètres de sécurité (algorithme, méthode d'authentification, etc.) associé à l'application.
- Un mécanisme de filtrage associé à l'application.

Une particularité à SEP est de permettre la distribution de son service base de donnée DBP entre plusieurs implémentations SEP. Ceci permet aux IAs de contrôler l'ensemble des serveurs internes SEP ainsi que les différentes politiques appliquées. Ca permet aussi de faciliter la configuration des politiques de sécurité entre les entreprises dans un large domaine de communications.

4. Utilisation des algorithmes de chiffrement robustes et fiables.

SEP doit garantir l'utilisation des meilleures pratiques de sécurité et techniques de chiffrement pour protéger tous les participants à une communication. Un des types d'attaque que subissent les protocoles existants comme SSL/TLS et IKEv1 est l'utilisation des algorithmes ou suites de chiffrement jugés comme non sécurisés tels que par exemple, RSA avec 512 bits pour le chiffrement asymétrique ou DES avec 56 bit pour le chiffrement symétrique. Le problème vient en effet des combinaisons (cas de IKE) et des listes (cas de SSL/TLS) exhaustives d'algorithmes dont la plupart n'ont jamais été implémentées. Pour cela, SEP est basé sur une liste très limitée des algorithmes et des protocoles cryptographiques sûrs tels que AES, RSA 1024 et SHA-1.

5. Support pour plusieurs méthodes d'authentification

Afin de permettre un meilleur déploiement de notre protocole, SEP propose plusieurs mécanismes d'authentification pour ses clients. A la différence des serveurs ou IAs SEP qui doivent explicitement présenter un certificat X.509 pour prouver au mieux leur identité, le protocole SEP est plus flexible du côté des clients qui peuvent différer par leur type d'environnement. Il propose à ces deniers, suivant le service que les clients demandent, une protection complète de toute méthode d'authentification (mot de passe, clé partagée, certificat, etc.).

6. Négociation simple de nouveaux services

A partir du premier échange SEP, le destinataire doit connaître tous les services et applications que l'initiateur souhaite accéder. En effet, le client demande, à partir de son premier échange avec le serveur, l'ensemble des ressources auquel il souhaite accéder. Ces ressources peuvent être soit sur le même serveur origine ou partagées entre plusieurs serveurs origines placés derrière une IA SEP.

Après cette négociation, le serveur demande au client une ou plusieurs méthodes d'authentification suivant les services négociés.

7. Protection à long terme des données

La plupart des protocoles de sécurité assurent les services de confidentialité et d'intégrité de données au moment de la transaction entre les communicateurs. Ainsi, lorsque les données arrivent aux applications, toute modification de donnée n'est plus protégée. La protection à long terme des données permet à une entité de prouver qu'une communication aura lieu avec une autre entité ainsi que les données échangées entre elles. Le service cryptographique qui permet d'assurer ce type de protection est le service de signature numérique.

8. *Le PFS (Perfect Forward Secrecy)*

Cette propriété est garantie si la découverte par un adversaire d'un secret à long terme ne compromet pas les clés des sessions générées précédemment. SEP garantit cette propriété d'une manière optionnelle à travers un échange SEP abrégé qui se déroule après un échange complet. En effet, cette propriété est toujours coûteuse en terme d'opérations cryptographiques et son utilisation ne sera nécessaire qu'à la suite d'une demande de l'un des participants à une session SEP.

9. *Le filtrage des transactions.*

SEP assure un service de filtrage dynamique à plusieurs niveaux de la couche OSI. Le mécanisme de filtrage est défini par chaque application (voir §6.11.6.2). SEP propose un filtrage des paquets au niveau réseau et un autre au niveau des en-têtes SEP.

10. *Le 'log' ou traçage des transactions*

Deux populations, très différentes de par leur besoin, leur culture informatique et leur maîtrise de l'informatique, sont concernées par les protocoles de sécurité: les utilisateurs finaux et les administrateurs systèmes et réseaux.

Les utilisateurs exigent la sécurisation de leur communication vers les serveurs destinataires sans avoir à s'inquiéter de la performance de leur réseau.

Les administrateurs quant à eux, souhaitent toujours assurer un fonctionnement correct de leurs machines et une meilleure sécurité de leur réseau.

Ainsi, les besoins des administrateurs nous semblent intéressants afin de permettre aux administrateurs de mieux surveiller les réseaux et les différents types d'accès sur les systèmes. Avec SEP, les administrateurs utilisent les IAs comme points d'audit et de traçage des connexions. En effet, ces points d'accès peuvent donner des audits et des statistiques de trafic sur toutes les connexions entrantes et sortantes de leur réseau.

6.11 Le protocole SEP

Dans ce paragraphe, nous définissons l'architecture de notre protocole SEP ainsi que ses différents sous protocoles.

6.11.1 Les sous protocoles

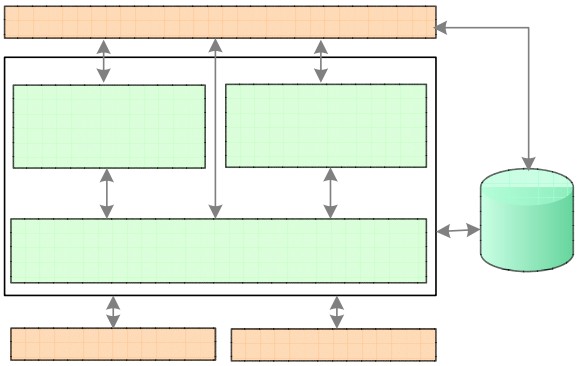


Figure 6-9. Architecture de SEP

SEP se compose de trois sous protocoles et d'une base de données :

1. *Le protocole de transfert [TRANSFER]*

Le protocole TRANSFER est la couche basse du protocole SEP. Ce protocole a pour fonction de fournir un service de transfert des données à travers un protocole fiable comme TCP ou bien à travers un protocole non connecté et non fiable comme UDP. SEP utilise le port 300 et s'inspire de [RFC2246] et de [Resc03] pour fournir le transfert de données au dessus des protocoles TCP et UDP.
Il fournit également la confidentialité et l'intégrité des messages. Suivant le protocole de transport et le service négocié, TRANSFER fragmente les données reçues par la couche applicative, compresse, authentifie, chiffre les fragments puis ajoute un en-tête SEP. L'en-tête SEP contient le type de données et le type de l'échange négociés entre les deux entités communicantes. Une fois les messages arrivés chez le destinataire, TRANSFER reconstitue les fragments et l'envoie aux applications.
A travers l'en-tête SEP, qui reste en clair, TRANSFER fournit un service de contrôle d'accès et d'autorisation suivant la politique définie par chaque application.

2. Le protocole d'authentification et de négociation des services [ASNP]

Le protocole d'authentification et de négociation des services (ASNP : Authentication and Service Negotiation Protocol) permet aux différentes entités SEP de s'authentifier, de négocier leurs paramètres de sécurité et une suite de services optionnels. Durant cette phase, les communicateurs négocient pour chaque service leurs paramètres de sécurité et échangent les informations nécessaires au chiffrement de leur trafic de données.

3. Le protocole de gestion des alarmes [ALARM]

Le protocole de gestion d'alarmes est chargé de signaler les erreurs rencontrées pendant la vérification des messages ainsi que toute incompatibilité qui pourrait survenir pendant la phase d'initialisation du protocole. Il permet également de signaler quelques paramètres session tels que l'expiration d'une session SEP ou la fermeture d'une connexion.

4. La base de données des politiques de sécurité [DBP]

DBP est constituée d'un ensemble de règles stratégiques. Chaque règle comprend la définition des paramètres de sécurité ainsi que les politiques de négociation pour chaque application.
DBP ressemble à une base de données relationnelle qui s'interface avec tous les autres protocoles SEP. Cette base de données doit être définie au niveau de chaque application et elle sera par la suite négociée par le protocole ASNP.

Dans la suite de ce chapitre, nous détaillons les protocoles SEP ainsi que ses différents types d'échange.

Service	Fonctionnalités
Politique [DBP]	- *Définition des services optionnels et de leurs paramètres de sécurité.* - *Définition des règles de sécurité.* - *Définition du niveau de filtrage.*
Protocole de Transfert [TRANS]	- *Création d'un tunnel SEP au-dessus de TCP ou UDP.* - *Fragmentation et défragmentation des messages SEP.* - *Chiffrement et authentification des messages.* - *Mise en œuvre des règles de sécurité définies dans DBP.*
Protocole d'authentification et de négociation des services [ASNP]	- *Négociation au dessus de TCP* - *Authentification des entités SEP (client, IA, serveur).* - *Négociation des différents services.* - *Négociation des paramètres de sécurité pour chaque service*

	- *Négociation de la clé de sessions.* - *Identification des sessions.* - *La protection d'identité du client.* - *Le service de PFS.*
Protocole de Gestion des alarmes [ALARM]	- *Définition des messages d'alarme pour les différents cas d'utilisation du SEP et leurs différents services*

Afin de faciliter la construction et la mise en œuvre de SEP, nous proposons d'utiliser la syntaxe SSL/TLS (cf. Annexe C) pour décrire les différents composants et services SEP. Une description complète de tous les champs SEP se trouve également dans l'annexe C.

6.11.2 La session SEP

SEP utilise le concept d'établissement de sessions entre les communicateurs. Une session SEP est une association entre un client et une IA et/ou un serveur SEP qui sera créée après la phase d'initialisation de ce protocole. Une session SEP peut être partagée par de multiples applications chacune ayant négociée ses propres paramètres de sécurité cryptographiques.
En outre, SEP suppose que plusieurs clients peuvent partager un tunnel SEP existant. Pour cela, une session SEP doit pouvoir différencier les trafics de chaque client.
Pour ce faire, nous proposons d'ajouter dans l'en-tête des paquets SEP un identificateur session (*Session ID*) sur 32 bits. Cet identificateur est la concaténation de deux valeurs :

- *Global Session ID* (*GSID*): un nombre aléatoire sur 16 bits généré par les IAs ou les serveurs origines SEP. Ce numéro sera utilisé pour identifier d'une manière unique, une session SEP.
- *Local Session ID* (*LSID*): un nombre aléatoire sur 16 bits généré par les clients SEP et qui permet d'identifier les paquets de chaque client SEP.

Les paramètres de sécurité d'une session SEP sont:

- *L'identificateur Session (GSID + LSID).*
- *Les participants :* une session SEP garde toujours dans une base de données les adresses IP des participants ainsi que les ports utilisés.
- *Le certificat du serveur ou IA:* certificat de signature qui correspond à la version 3 de X.509.
- *Le Master secret :* un secret partagé entre les entités avec une taille de 48 octets. A partir de cette valeur, nous générons aussi trois clés de session: *Master_e, Master_a, Master_d.*
- *Les services négociés :* une session SEP peut contenir plusieurs type de trafic revenant à chaque service négocié. Chaque service négocié sera suivi par une liste de paramètres. Les paramètres les plus importants sont :

    - *Les méthodes d'authentification du client :* suivant les services négociés, le client fera plusieurs ou aucune authentification au cours d'une même session.
    - *L'algorithme de hachage :* il définit l'algorithme d'authentification des messages et la taille de hachage.
    - *L'algorithme de chiffrement symétrique :* il définit l'algorithme de chiffrement symétrique des données (3DES, AES, etc.). Seuls les algorithmes de chiffrement par bloc sont utilisés avec SEP.
    - *L'algorithme d'échange des clés :* il définit l'algorithme d'échange des clés utilisé par SEP.

La négociation des services, des méthodes d'authentification et des suites de chiffrement se fait en clair pendant l'établissement de la session. Le tableau suivant contient les algorithmes négociés par le protocole ASNP de SEP.

Fonction	**Algorithme**
Echange de clés	RSA, DH
Chiffrement symétrique en blocs	AES, 3DES, DES
Hachage	MD5, SHA

Tableau 6-1. Algorithmes négociés par le protocole ASNP de SEP

6.11.2.1 Synopsis de calcul de paramètre

La figure suivante illustre le calcul de la clé maître dans une session SEP. La clé maître ou *master_secret* est générée à partir des deux nombres aléatoires, *Nonce_I* et *Nonce_R,* échangées en clair entre les entités SEP finales et la valeur DH secrète, g^{xy} résultant de l'échange des valeurs DH publiques. Si le chiffrement asymétrique est utilisé à la place de la mécanisme DH, le troisième paramètre utilisé pour la génération du *master_secret* sera le nombre aléatoire *pre_master_secre*t envoyé par le client dans le message KE.
Cependant, et afin d'empêcher de multiples attaques sur l'utilisation des valeurs publiques DH éphémères dans la génération des clés [Kraw03], nous préférons n'utiliser ce mécanisme qu'optionnellement et à travers un échange SEP abrégé et protégé en confidentialité.

La génération de la clé *master_secret (48 octets)* revient à utiliser une fonction PRF+ de la manière suivante :

master_secret = PRF+ (*pre_master_secret*, «*RSA KE*», *Nonce_I*, *Nonce_R*)

où
« RSA KE » est un label de type caractère.
Le symbole « | » signifie une concaténation, et
PRF+ (key,data) = T1 | T2 | T3 | T4 | ...
Avec:

T1 = prf (key, data | 0x01)
T2 = prf (key, T1 | data | 0x02)
T3 = prf (key, T2 | data | 0x03)
T4 = prf (K, T3 | data | 0x04)

Cette opération doit s'effectuer tant que nécessaire pour calculer les trois clés dérivées :

Master_d employée pour dériver des nouvelles clés de session.
Master_e employée pour protéger la confidentialité des messages SEP.
Et *Master_a* employée pour authentifier les messages SEP.

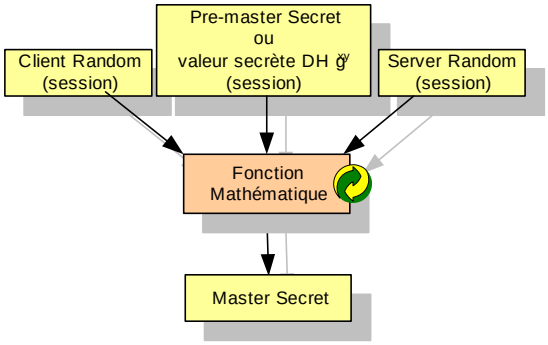


Figure 6-10. Construction du master secret

La clé *Matser_d* est affectée à la première valeur de T1. Cependant, la taille des deux clés *Matser_e* et *Master_a* est celle négociée entre les acteurs SEP. A titre d'exemple, supposons que les clés exigées sont une clé AES de 256 bits et une clé HMAC de 160 bits et que la fonction de PRF produit de 160 bits. La clé d'AES viendra alors de T2 et du début du T3, alors que la clé de HMAC viendra du reste de T3 et du début de T4.

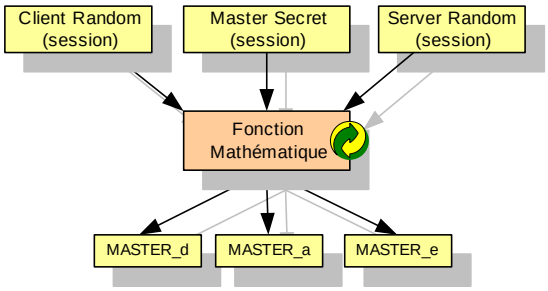


Figure 6-11. Génération des clés dans SEP

Dans un échange SEP abrégé, la valeur *Master_d* permet de générer une nouvelle clé *master_secret*. Ceci s'effectue selon les deux critères suivants :
Si la propriété de PFS est négociée avec un échange des valeurs publiques DH éphémères:

master_secret = PRF+ (*Master_d, «DH KE»*, g^{xy}, *Nonce_NI*, *Nonce_NR*)

Où

g^{xy} est la clé secrète DH
DH KE représente une chaîne de caractère (label)
Nonce_NI et *NONCE_NR* représentent les nouveaux nombres aléatoires négociés durant la deuxième phase.

Si la nouvelle génération des clés est basée sur un chiffrement RSA asymétrique

master_secret = PRF+ (*Master_d,* «*RSA KE*», *Nonce_NI*, *Nonce_NR*)

Où

RSA KE représente une chaîne de caractère (label)
Nonce_NI et *NONCE_NR* représentent les nouveaux nombres aléatoires négociés durant la deuxième phase.

6.11.3 Le protocole de transfert

6.11.3.1 Fonctionnement

Le protocole de transfert (TRANSFER) est la couche basse du protocole SEP. Il assure un service de transfert de données au-dessus des deux protocoles de transport TCP et UDP en utilisant le port 300. Avec TCP, TRANSFER s'inspire du protocole Record de SSL/TLS et de son mécanisme de fragmentation de données. Cependant, il utilise la proposition définie dans [Resc03] [Moda04] pour fournir un service de transfert de données sécurisés au dessus du protocole UDP.
TRANSFER permet alors de fournir les services de confidentialité et d'intégrité des données. Pour assurer la confidentialité des données, TRANSFER chiffre les données utiles avec la clé secrète *Master_e* dérivée durant la phase d'initialisation de SEP. Pour offrir l'intégrité des messages, il utilise une fonction MAC avec une deuxième clé *Master_a* dérivée de la clé maître *master_secret*.

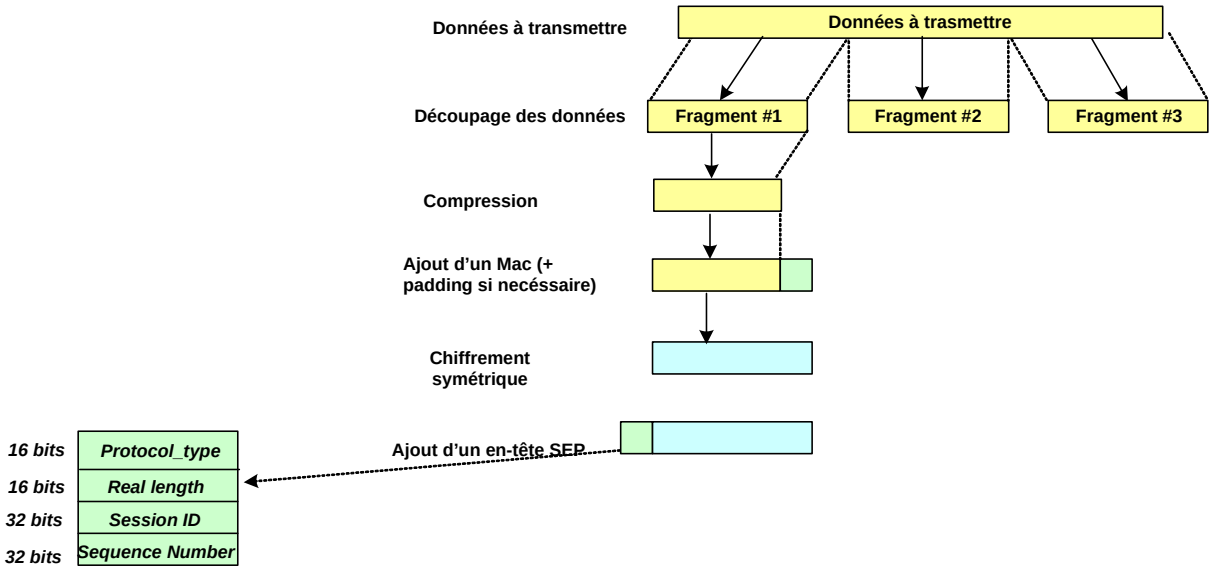


Figure 6-12. Mécanisme d'envoie des données par le protocole de transfert

TRANSFER fragmente un message applicatif à transmettre, puis compresse optionnellement les données, applique un MAC, chiffre le contenu puis ajoute un en-tête SEP (figure 6-12). Ensuite, il transmet l'unité résultante à TCP ou UDP. Les données reçues par le récepteur sont ensuite déchiffrées, vérifiées en intégrité puis délivrées aux applications. Le protocole TRANSFER définit des paramètres de sécurité négociés ou spécifiés durant la phase d'initialisation du protocole.

L'en-tête SEP (figure 6-13) comprend les champs suivants :

- ***Protocol Type (16 bits)***

Protocol Type décrit le service de plus haut niveau utilisé pour traiter les fragments. SEP définie deux types de base: *Alarm et ASNP. Alarm* signifie que le contenu du fragment est un message qui revient au protocole Alarm de SEP. Si la valeur du *Protocol Type* est *ANSP*, les fragments envoyés sont alors des messages *ASNP*.
Ainsi, après la négociation ASNP, les fragments de chaque application négociée aura des types différents tels que *HTTP* pour le flux Web et *FTP* pour le flux FTP.
Ce champ permet de distinguer entre les fragments correspondant à plusieurs applications ou protocoles dans une même session. Le fait que ce champ est sur 16 bits nous permet de couvrir tous les ports TCP ou UDP ainsi que leurs applications.

- ***Length (16 bits)***

Length contient la longueur en octets du fragment du texte chiffré.

- ***Session ID (32 bits)***

Session ID (SID) est un numéro arbitraire sur 32 bits qui identifie d'une manière unique chaque session SEP. SID est formé par la concaténation des deux valeurs : *GSID* (Global Session ID, 16 bits) et *LSID* (Local Session ID, 16 bits).

- ***Data (variable)***

Data contient les données à transférer. Sa taille diffère suivant le protocole de transport utilisé. Par exemple, un datagramme UDP ne doit pas dépasser les 1500 octets. Cependant, avec TCP, la taille des données atteint à 2^{14} octets (65 535 octets) ou moins.

- ***MAC (variable)***

MAC est le champ d'authentification d'un fragment SEP. Ce champ est produit par un algorithme MAC avec une fonction de hachage de type SHA-1 ou MD5 et qui concatène les données avec les champs des en-têtes SEP pour assurer une authentification sur tout le fragment envoyé.

Le calcul est le suivant :

MAC (Master_a, seq_num + S_ID + protocol_type + version + length + data)

Où

Master_a est la clé d'authentification des messages SEP.
seq_num est un numéro séquentiel incrémenté de 1 dans chaque message SEP et qui protège contre le rejeu et la duplication des paquets.
S_ID est l'identificateur session SEP.
protocol_type est le type du protocole utilisé.
version est la version négociée du protocole SEP.
length et la taille du fragment SEP.
data est le contenu du fragment SEP.

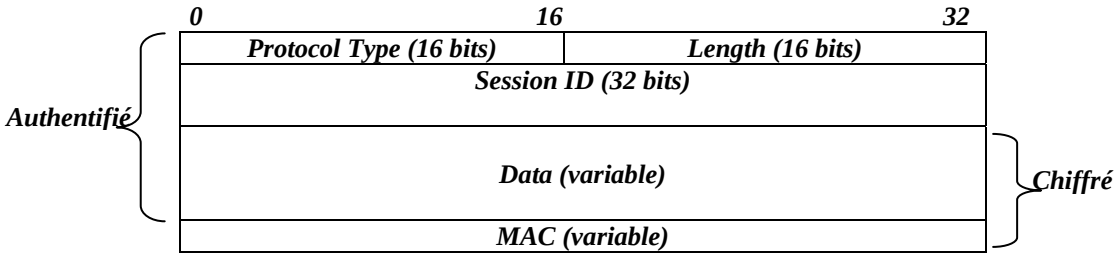


Figure 6-13. En-tête d'un message SEP au-dessus de TCP

Dans le cas de l'utilisation d'un canal non fiable comme UDP le champ suivant est ajouté explicitement à l'en-tête SEP.

- ***Sequence number SN (32 bits)***

Sequence number est un nombre premier qui s'incrémente de *un* pour chaque message envoyé entre les deux entités durant une même session SEP et qui est initié à *zéro* à l'ouverture d'une nouvelle session. A la différence du *seq_num* dans la partie MAC de l'en-tête SEP, le SN permet de protéger contre le rejeu des paquets même dans le cas où les applications n'exigent pas un service d'authentification des messages (voir paragraphe 6.9.7.1 de ce chapitre).

6.11.4 Le Protocole d'Authentification et de Négociation des Services (ASNP)

Le protocole d'authentification et de négociation des services ou ASNP (Authentication and Service Negotiation Protocol) est la phase d'initialisation du protocole SEP. ASNP génère une série de messages échangés entre les entités SEP afin de s'authentifier, de négocier leurs paramètres de sécurité ainsi que les services et les ressources que les clients souhaitent

utiliser. A la différence du protocole TRANSFER, le protocole ASNP opère seulement au dessus du protocole TCP en utilisant le port 301.
Les messages SEP possèdent les champs suivants:

- *content type* : il indique le type du message (8 bits).
- *length* : il indique la longueur du message (16 bits).
- *data* : il indique le contenu du message (> 8 bits).

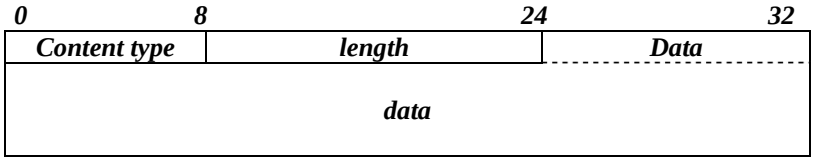


Figure 6-14. Format d'un message ASNP

Les types des messages définis dans SEP sont :

- ***CAN : Client Attribut Negotiation***

  Description :

  Ce message est envoyé par le client à l'ouverture d'une session SEP.

  Paramètres :

  CAN contient les deux paramètres suivants :

  - ***Version*** (8 bits) : le numéro de version du protocole SEP. Actuellement, il contient la valeur 1.0 (1 version majeure et 0 version mineure).
  - ***Nonce*** (16 octets): nombre aléatoire envoyé par le client.

- ***MAN : Médiator Attribut Negotiation***

  Description :

  Message envoyé par la IA à l'ouverture d'une session SEP.

  Paramètres :

  MAN contient les deux paramètres suivants :

  - ***Version*** (8 bits): le numéro de version du protocole SEP. Actuellement, il contient la valeur 1.0 (1 version majeure et 0 version mineure)
  - ***Nonce*** (16 octets) : nombre aléatoire généré par la IA.

- ***SAN : Server Attribut Negotiation***

  Description :

  Ce message est envoyé par le serveur à l'ouverture d'une session SEP.

  Paramètres :

  SAN contient les deux paramètres suivants :

  - ***Version*** (8 bits): le numéro de la version du protocole SEP. Actuellement, il contient la valeur 1.0 (1 version majeure et 0 version mineure).
  - ***NonceS*** (16 octets) : nombre aléatoire généré par le serveur.

- ***SNGreq : Service Negotiation Request***

  Description :

Ce message générique est envoyé par l'un des participants pour la demande d'un nouveau service.

Paramètres :

En plus des trois champs définis dans l'en-tête des messages SEP (*type, length* et *data*), le message *SNGreq* possède les trois champs suivants :

- ***service_type*** (7 bits) : il fournit le type du contenu de ces messages tel que *cipher_suite* ou *cookies.*
- ***flag*** (1 bit) : il signale si le service est critique ou non. Un service est critique si la réjection de ce service par un des communicateurs entraîne une fermeture automatique de la connexion SEP.
- ***data*** : contient des paramètres définis dans §6.11.4.1.

- ***SNGresp : Service Negotiation Response***

Description :

Ce message générique est envoyé par un des participants pour répondre à un service demandé. Les champs additionnels ajoutés dans ce message sont similaires à ceux du message SNGreq.

Paramètres :

Comme le message SNGreq, le message SNGresp possède les trois champs suivants :

- ***service_type*** (7 bits) : fournit le type du contenu de ces messages tels que *cipher_suite* ou *cookies.*
- ***flag*** (1 bit) : Ce champ est toujours égal à 1 dans SNGresp. C'est-à-dire le service est toujours critique.
- ***data*** : contient des données (§6.11.4.1) qui reviennent à chaque type demandé.

- ***KE : Key Exchange***

Description :

Permet l'échange et la dérivation d'un ensemble de clés session à travers :

- Un numéro aléatoire (*pre_master_secret*) chiffré asymétriquement avec la clé publique du serveur ou de la IA.
- Un numéro aléatoire chiffré symétriquement avec la clé de chiffrement *Master_e* générée dans une ancienne session SEP.
- Des valeurs DH éphémères utilisées seulement dans un échange SEP abrégé.

Paramètres :

- Si le récepteur (serveur ou IA) commence par envoyer ce message (échange SEP abrégé), il doit inclure son paramètre DH (p et g) ainsi que sa valeur publique DH éphémère.
- Si l'émetteur (client ou IA) reçoit un message KE, il doit répondre en envoyant sa valeur publique DH éphémère.
- Si l'émetteur ne reçoit pas un message KE du récepteur, il répond par un numéro aléatoire (*pre_master_secret*) crypté avec la clé publique du serveur.
- Si l'émetteur demande l'établissement d'une session SEP en se basant sur un LSID non nul, et le récepteur accepte cette négociation, le récepteur envoie le

message KE contenant un *pre_master_secret.* Le message KE sera chiffré symétriquement avec la clé *Master_e* générée dans la session LSID non nulle.

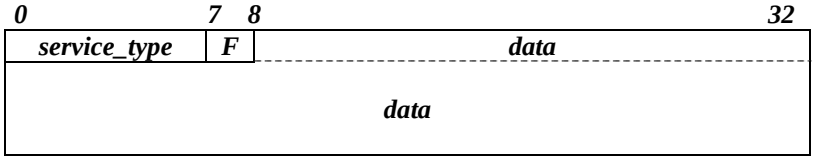


Figure 6-15. Format des messages *SNGreq et SNGresp*

- ***AUTH : Authentication***

  Description :

  Message générique qui permet d'assurer plusieurs méthodes d'authentification. Pour le serveur ou la IA, leur authentification est explicite par des certificats X.509 tandis que pour les clients, l'authentification se fait par plusieurs méthodes telles que les clés partagées, les certificats d'attribut, etc.

  Paramètres :

  En plus des trois champs définis dans l'en-tête des messages SEP (*type, length* et *data*), le message AUTH possède les deux champs suivants:
  - ***authentication_type*** (8 bits) : fournit le type du contenu de ces messages tel que *password, psk, etc.*
  - ***data*** : contient des données (§6.11.4.2) qui reviennent à chaque type demandé.

- ***AUTHSucc : Authentication Succeed***

  Description :

  Signale si l'authentification est acceptée ou rejetée ainsi que le nom de l'application à laquelle elle appartient.

  Paramètres :

  Contient un label (chaîne de caractères) sur 6 octets et le nom de l'application comme définie dans le paragraphe 6.9.4.1. Notons qu'un message AUTHsucc peut contenir plusieurs noms d'application.

  Pour le label, il prend un des valeurs suivantes :

  accept : si l'authentification a réussit ou,
  reject : si l'authentification a échoué.

- ***NGSucc : Negociation Succeed***

  Description :

  Ce message signale la fin du protocole ASNP et le début de l'émission des données protégées avec les paramètres de sécurité négociés.

  Paramètres:

  Contient le hachage (MD5 ou SHA1) de l'ensemble des messages négociés.

6.11.4.1 Les types définies pour SNGreq et SNGresp

Les types définis pour ces messages sont :

- ***Server_application***

  Description :

  Le type *server_application* a une importance particulière par rapport aux autres types définis. Un message *SNGreq* (ou *SNGresp*) de type *server_application* est suivi par zéro ou par plusieurs messages *SNGreq* (ou *SNGresp)* contenant les paramètres de sécurité nécessaires pour ce service.

  Flag : 0x01 (message critique).

  Data: il contient les valeurs suivantes séparée par « : » :
  transport_protocol : application_protocol : server_name : destination_port : number_of_next_security_parameters
  A titre d'exemple, TCP :http :www.enst.fr :80 :n signifie la demande d'une connexion vers un serveur web (http://www.enst.fr) sur le port 80. Le numéro *n* signifie que l'application est suivie par trois autres messages qui lui appartiennent.
  Une entité SEP peut initialiser les paramètres *transport_protocol, application_protocol* à zéro si leurs valeurs sont inconnues ou même s'ils ne fournissent aucune information en plus. Par exemple, un client qui demande le certificat du serveur www.enst.fr:80 peut mettre les autres champs à zéro.

- ***Certificate_request***

  Description :

  Ce message est envoyé pour demander le certificat X.509, le certificat d'attribut ou le statuts OCSP d'un serveur ou d'une IA SEP, ainsi que sa chaîne de certification. Même si ce message n'est pas envoyé, le serveur ou la IA SEP doivent présenter leur identité avec le message AUTH. Ceci permet au client de choisir sa préférence en type d'authentification.

  Flag : 0x01 ou 0x00

  Data: il contient le type du certificat demandé sur 1 octet:
  X.509 Certificate – Signature,
  X.509 Certificate – Attribute,
  ICARE Attribute Certificate,
  Hash and URL of X.509 certificate,
  OCSP Certificate Status.

- ***Certificate***

  Description :

  Ce message contient le certificat X.509 revenant à une application particulière. Pour cela, il diffère du message AUTH qui authentifie un acteur SEP (client, serveur ou IA). Ce message est lié au type server_application.

  Flag : 0x01 ou 0x00

  Data : le contenu diffère suivant la requête de la demande de certificat. (voir annexe C).

- ***Cipher_list***

Description :

C'est la liste des suites de chiffrement choisies par le client, la IA ou le serveur. Un client SEP peut ne pas proposer une suite de chiffrement. Ce cas est généralement utilisé si le client demande l'ouverture d'une session SEP à travers un Proxy SEP.
Key-exchange_Asymetric-alg_Symetric-alg_Hash-fonction

(eg. DH_RSA-1024_3DES_SHA1).

Flag : 0x01 ou 0x00

Data: Chaque suite de chiffrement (2 octets) contient une liste de proposition séparée par « _ ». La suite est de la forme :

SEP_Key-exchange_taille _Symetric-alg_taille_Hash-fonction

A titre d'exemple, SEP_RSA_1024_AES_128_SHA1 signifie l'utilisation de RSA avec des clés sur 1024 bits pour la négociation des clés, l'algorithme AES avec des clés de 128 bits pour le chiffrement asymétrique et SHA1 pour le hachage.

- ***Authentication_request***

Description :

Ce message est envoyé par la IA ou le serveur pour réclamer au client une authentification pour chaque service demandé.

Flag : 0x01 ou 0x00

Data: il contient le type du certificat demandé sur 1 octets :

X.509 Certificate – Signature,
X.509 Certificate – Chain,
X.509 Certificate – Attribute,
ICARE Attribute Certificate,
Hash and URL of X.509 certificate,
Pre Shared Key,
Password.

- ***Compression_list***

Description : Suite des méthodes de compression. Actuellement, le protocole ne définit pas une méthode de compression.

Flag : 0x01 ou 0x00

Data: null

- ***Session_life_time***

Description :

La présence de ce paramètre oblige les acteurs SEP à renouveler leurs clés session à travers un échange SEP abrégé. Notons que la renégociation doit se faire à la demande de l'initiateur de la communication SEP. Le répondeur doit explicitement envoyer une alarme avant l'expiration de la session SEP pour informer l'émetteur de la fermeture automatique de la session si sa durée de vie est terminée.
Si ce champ n'est pas envoyé, les implémentations doivent préciser une durée de vie limitée (par exemple, 86400 s).

Flag : 0x01 (message toujours critique)

Data: le renouvellement d'une session SEP peut dépendre de la quantité de données échangées ou bien d'un temps fixé en secondes. Pour cela, il contient deux champs, le premier identifie le type (seconds ou kilobytes) et le deuxième le contenu (eg. 3500 s ou 1000 kilobytes).

- ***Non_repudiation***

  Description :

  Demande d'un service de non répudiation sur les données de l'application. Si les participants acceptent ce service, les deux bouts de la communication SEP doivent signer tous les messages échangés entre eux suivant le type et le format négociés.

  Flag : 0x01 ou 0x00.

  Data: ce champ contient les mêmes variables définies pour TLS SIGN à savoir le type de la signature (Non_repudiation_with_proof_of_origin, Non_repudiation_with_proof_of_receipt, Non_repudiation_with_double_proof), le format de signature (XML-DSIG, PKCS7) et un booléen pour activer ou désactiver ce service.

- ***Delete_session***

  Description :

  Demande de suppression de l'identificateur session SEP par un des acteurs. Ce message ne doit jamais être envoyé en clair.

  Flag : 0x01 (message toujours critique).

  Data: Contient l'identificateur de la session (*GSID* + *LSID*).

- ***max_fragment_length***

  Description : Ce message contient la longueur maximale que nous souhaitons utiliser pour les fragments SEP. Ce service est important pour des participants à faible bande passante.

  Flag : 0x01 ou 0x00.

  Data: les valeurs définies pour ce champ dans l'annexe C sont : 2^9, 2^10, 2^11 et 2^12 octets.

- ***truncated_hmac***

  Description : afin de permettre de minimiser les en-têtes des fragments SEP suivant les besoins des participants, ces derniers peuvent négocier des HMAC tronqués.

  Flag : 0x01 (message toujours critique).

Data: les valeurs définies pour ce champ dans l'annexe C sont : 80 ou 96 octets. La valeur complète d'un HMAC est calculée, ensuite elle est tronquée en utilisant les 80 ou 96 premiers bits.

6.11.4.2 Les types définis pour AUTH

Les types définis pour ce message sont :

- ***X509_certificate***
  Description :
  Ce message contient un certificat X.509 v3 pour authentifier un acteur SEP. Il est envoyé suite à une demande par *certificate_request* ou directement par un des acteurs concernés. Ce type peut être envoyé si le serveur ou la IA ont changé un certificat après révocation par exemple.

  Data: le certificat X.509 v3 est codé en format DER.

- ***X509_certificate_signature***
  Description :
  Ce message contient une signature sur tous les paramètres négociés. Ce message est envoyé si un service de non répudiation est demandé.

  Data: il contient la signature de tous les messages négociés en format PKCS#7.

- ***X509_certificate_attribute***
  Description :
  Ce message contient un certificat X.509 d'attribut.

  Data: il contient un certificat X.509 d'attribut est codé en format DER.

- ***ICARE_attribute_certificate***
  Description :
  Message contenant un certificat d'attribut défini dans le projet RNRT ICARE.

  Data : il contient un certificat d'attribut ICARE codé en XML.

- ***HASH_URL_X509_certificate***
  Description :
  Ce message contient le hachage et l'URL d'un certificat X.509 stocké dans un annuaire LDAP.

  Data : il contient un hachage (sur 16 ou 20 octets) et l'URL du certificat X.509.

- ***PSK (Pre Shared Key)***
  Description :
  Authentification faible par une clé partagée.

  Data : il contient une clé partagée sur 16 octets

- ***Password***

  Description :

  Authentification faible par un identificateur et un mot de passe.

  Data: contient l'identificateur client (sur 16 octets) et son mot de passe (16 octets).

6.11.4.3 Déroulement de l'échange SEP

1. Phase 1 : Etablissement des paramètres de sécurité et authentification du serveur

Dans cette section, nous décrivons un échange SEP complet qui se déroule directement entre un client et un serveur SEP comme schématisé dans la figure 6-17.

Le client SEP commence cette phase en envoyant dans le message CAN (Client Attribute Negociation), la version du protocole (version 1.0) et un nombre aléatoire *Nonce_C*. Le client envoie aussi un ou plusieurs messages *SNGreq* contenant les services qu'il souhaite utiliser. Chaque service est suivi par une liste qui définit ses paramètres de sécurité tels que par exemple la liste des suites de chiffrement, un service de non répudiation, etc. (figure 6-16)

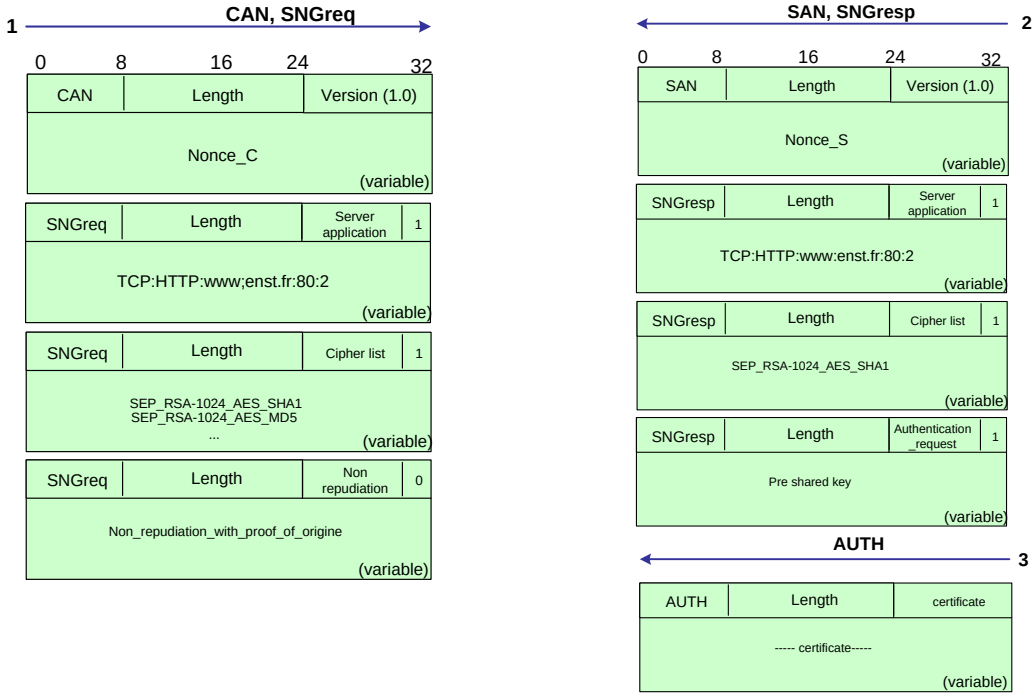


Figure 6-16. Premier échange ASNP entre le client et le serveur SEP

Chaque message ASNP possède un champ qui identifie sa longueur. Ainsi, plusieurs messages peuvent être concaténés sous un même en-tête SEP qui lui aussi, contient la taille de son contenu. Lorsque le client envoie sa demande d'ouverture de session, il génère un *LSID* non nul et fixe la valeur ASNP comme le type du protocole de négociation choisi.

Le serveur génère un nombre aléatoire *Nonce_S.* Il choisit sa version du protocole et envoi ses paramètres dans le message *SAN*. Le serveur doit également vérifier les services demandés par le client ainsi que sa politique d'accès. Il répond aux services demandés avec les messages *SNGresp* et ajoute les méthodes d'authentification (*authentication_request*) et les paramètres de sécurité (*cipher_list,* etc.) pour chaque service. Dans l'en-tête SEP, le serveur génère un *GSID* aléatoire non nul qu'il le concatène avec le *LSID* envoyé par le client.

Le serveur continue la négociation en envoyant son identité. Il envoie dans le message *AUTH* son certificat X.509. A la réception de ces informations par le client, ce dernier vérifie le certificat et enchaîne avec la deuxième phase.

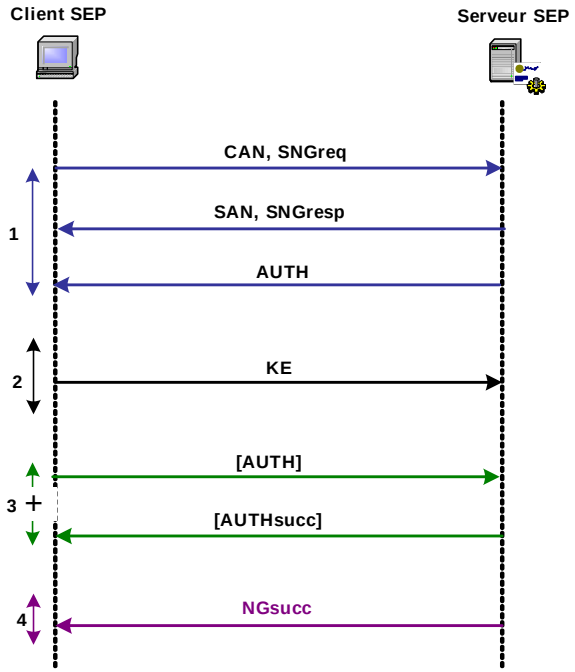


Figure 6-17. Echange complet entre un client et un serveur SEP[11]

2. *Phase 2 : Echange des clés*

A ce stade, les deux entités procèdent à l'échange des clés. Un échange complet SEP possède un seul mécanisme explicite pour l'échange des clés. Ce mécanisme est basé sur le chiffrement asymétrique avec la clé publique du serveur.

Pour cela, le client génère un numéro aléatoire *pre_master_secret* et l'envoie crypté avec la clé publique du serveur correspondant à son certificat X.509 (figure 6-18). À la réception de ce message, le serveur déchiffre *le pre_master_secret* avec sa clé privée.

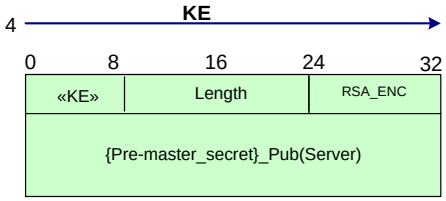


Figure 6-18. Deuxième échange ASNP entre le client et le serveur SEP

Comme expliqué au paragraphe 6.9.2.1, les deux entités peuvent maintenant générer leur clé secrète (*master_secret*) ainsi qu'un ensemble de clés dérivées (*Master_e*, *Master_a* et *Master_d*) servant au chiffrement, à l'authentification et aux dérivations de nouvelles clés. Une session chiffrée est maintenant créée entre les deux communicateurs.

3. *Phase 3 : Authentification du client*

[11] Dans cette figure et dans les cinq autres qui suivent, le caractère + indique un message ou un ensemble de messages qui peuvent se produire zéro ou plusieurs fois en succession. [M] indique que le message M est optionnel.

Une fois que les entités ont terminé les trois phases précédentes, toutes les informations circulant entre les deux entités seront chiffrées. Le client doit essayer maintenant de s'authentifier auprès du serveur SEP suivant l'ordre des services négociés dans la première phase. Le client envoie le premier message *AUTH* contenant sa méthode d'authentification pour le premier service (figure 6-19). A chaque message *AUTH* envoyé par le client et accepté par le serveur, ce dernier doit explicitement répondre en envoyant le message *AUTHsucc* avec la valeur *accept* et le nom de l'application à laquelle revient cette authentification. Si l'authentification échoue le message envoie aussi le même message *AUTHsucc* mais contenant la valeur *reject*.
Ainsi, si le client demande un accès à plusieurs applications, le serveur peut concaténer les authentifications qui se ressemblent. A titre d'exemple, si le client souhaite accéder aux deux serveurs Web et messagerie électronique qui demandent une authentification forte par un certificat d'identité délivré par la même autorité de confiance, le serveur doit depuis la première authentification du client lui envoyer le message AUTHSucc avec le nom des deux applications dedans. Ceci peut éviter une deuxième authentification inutile avec le même certificat.
Dans SEP, si l'authentification du client est explicite (flag égale à 1), l'échec de vérification doit aboutir à une fermeture automatique de la connexion après l'envoi du message NGSucc contenant la valeur *reject* et le nom de l'application dont l'authentification a échoué.
Il est à noter que les méthodes d'authentification qui semblent au premier abord non sécurisées telles que l'authentification par mot de passe ou par une clé partagée, peuvent être utilisées en toute sécurité. Ceci est réalisé grâce à la nature sécurisée de la phase 3 de SEP. Si aucune méthode d'authentification n'est disponible, le serveur passe à la phase suivante.

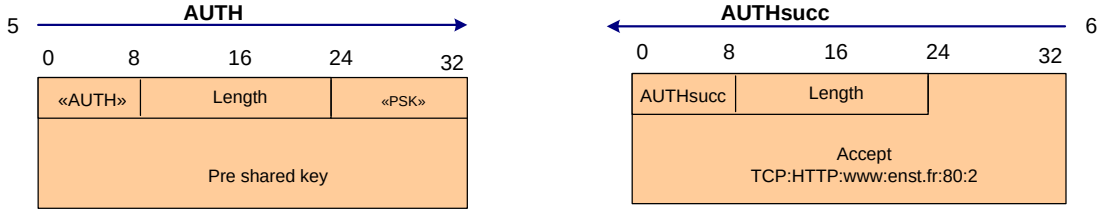


Figure 6-19. Troisième échange ASNP entre le client et le serveur SEP

4. Phase 4 : Fermeture de l'échange ASNP

Afin de terminer l'échange ASNP avec toutes les authentifications nécessaires, le serveur envoie le message *NGsucc* qui permet au deux entités de commencer la phase de protection des données avec les algorithmes et fonctions négociés durant cette phase.

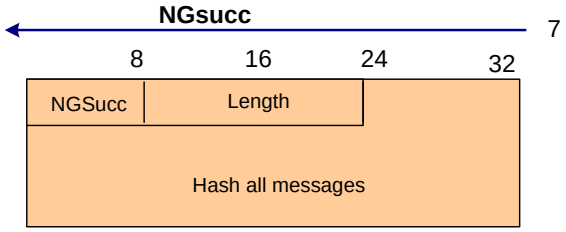


Figure 6-20. Quatrième échange ASNP entre le client et le serveur SEP

6.11.4.4 Déroulement de l'échange SEP abrégé

Récemment, nous avons proposé à l'IETF une extension du protocole TLS qui permet d'authentifier les deux entités client/serveur avec des clés pré partagées [badr04]. Ce travail, soutenu par des partenaires comme NOKIA et CISCO dans le groupe de travail TLS à l'IETF, permet l'utilisation de SSL/TLS dans les équipements réseaux qui possèdent des interfaces

Web pour leur configuration. L'authentification par clé partagée permet un accès sécurisé à ces interfaces, limité à quelques administrateurs réseaux. Une autre proposition [badr04+] qui converge aussi dans le même but, permet aux deux entités SSL/TLS de sauvegarder leurs paramètres sessions et de les réutiliser ultérieurement dans un échange SSL/TLS abrégé.
En se basant sur ces deux propositions, SEP supporte un échange abrégé entre deux acteurs. L'échange SEP abrégé est basé sur le chiffrement symétrique et ne nécessite pas l'utilisation des certificats X.509 pour l'authentification du répondeur (serveur ou IA).
L'intérêt de cet échange est de permettre à des clients qui ne peuvent pas effectuer des opérations cryptographiques lourdes (chiffrement asymétrique, vérification du certificat, etc.) de négocier leurs services et paramètres de sécurité à travers un mécanisme de chiffrement symétrique.
Les variables session (GSID, LSID, algorithmes de chiffrement, fonction de hachage, service(s), et les clés sessions) nécessaires pour cette négociation peuvent être partagées à travers une ancienne session SEP ou même manuellement entre les acteurs.
De plus, l'échange SEP abrégé permet d'assurer un service de PFS (*Perfect Forward Secrecy*) pour le renouvellement des clés, en se basant sur la négociation des valeurs publiques DH éphémères.
Si les deux entités souhaitent un renouvellement des clés sans la propriété de PFS, elles peuvent se baser sur un chiffrement asymétrique du *pre_master_secret* avec la clé publique du répondeur ou même sur un chiffrement symétrique avec la clé de chiffrement *Master_e*. Notons qu'avec SEP, la génération des clés ne dépend pas des services négociés entre les différents acteurs.

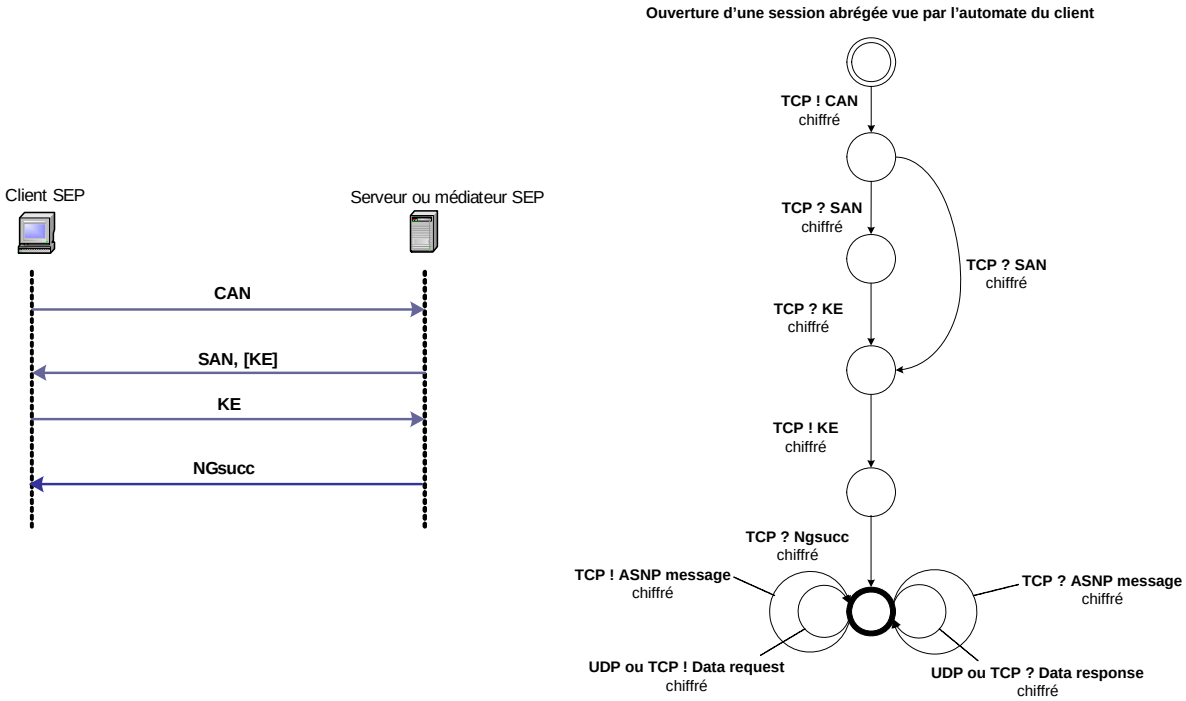


Figure 6-21. Echange SEP abrégé avec l'automate[12] abrégé du client

L'échange SEP abrégé (figure 6-21) est basé sur le chiffrement symétrique à l'aide de la clé de chiffrement calculée dans une session SEP précédente.

[12] Dans cet automate et dans ceux qui suivent, nous utilisons la notation suivante :
TCP ! messageA : Emission du messageA
TCP ? messageB : Réception du messageB

Le client envoie dans le message *CAN* les deux numéros *GSID* et *LSID* non nuls. *GSID* et *LSID* correspondent à une ancienne session SEP. Avec SEP, toutes les sessions sont résumables. Elles dépendent seulement de la durée de vie négociée.
Si la session demandée n'a pas encore expiré, le serveur (ou la IA) envoie le message *SAN* (respectivement *MAN*). Il envoie également le message *KE* contenant une valeur publique DH. Si le serveur n'envoie pas le message KE, le client peut utiliser un mécanisme de chiffrement symétrique ou asymétrique pour la génération des clés session. Ainsi, toutes les clés et les nombres aléatoires sont renouvelés par les deux acteurs. L'échange SEP se termine par un message *NGSucc* qui contient le hachage de tous les messages SEP. A ce point, les deux entités peuvent échanger des données chiffrées ou de nouveaux messages ASNP.

6.11.4.5 Déroulement des autres types d'échanges SEP

Les quatre autres types d'échange se ressemblent par leur mécanisme de négociation aux exceptions suivantes :

- Les services de base ne sont pas toujours assurés de bout en bout.
- Un des trois acteurs (client ou serveur) ne participe pas à la négociation des paramètres de sécurité.

6.11.4.6 Connexion client ----- IA – serveur

La figure 6-22 illustre la négociation décrite à la section 6.4 de ce chapitre. Dans cette négociation, le client envoie les messages *SAN* et *SNGresp* vers un serveur SEP. Ces messages sont interceptés par la IA M. Cette dernière répond en envoyant les messages *MAN* (Mediator Attribute Negotiation) accompagnés par un ou plusieurs messages *SNGresp*. A la réception de ces messages, le client remarque la présence du message *MAN* venant d'une IA tiers inconnue.

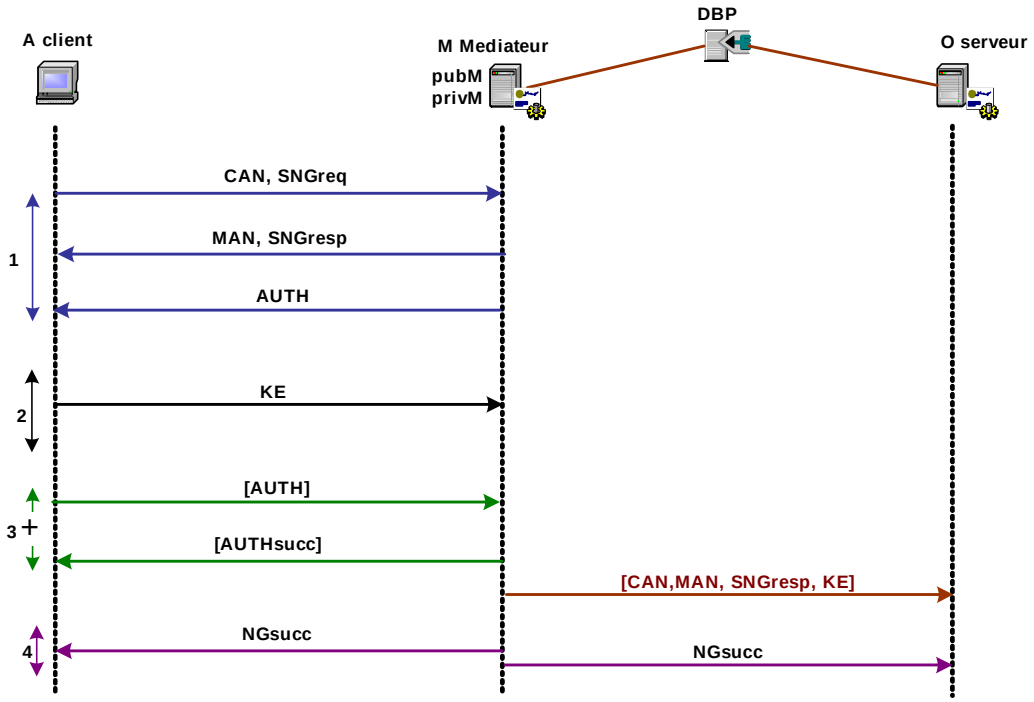
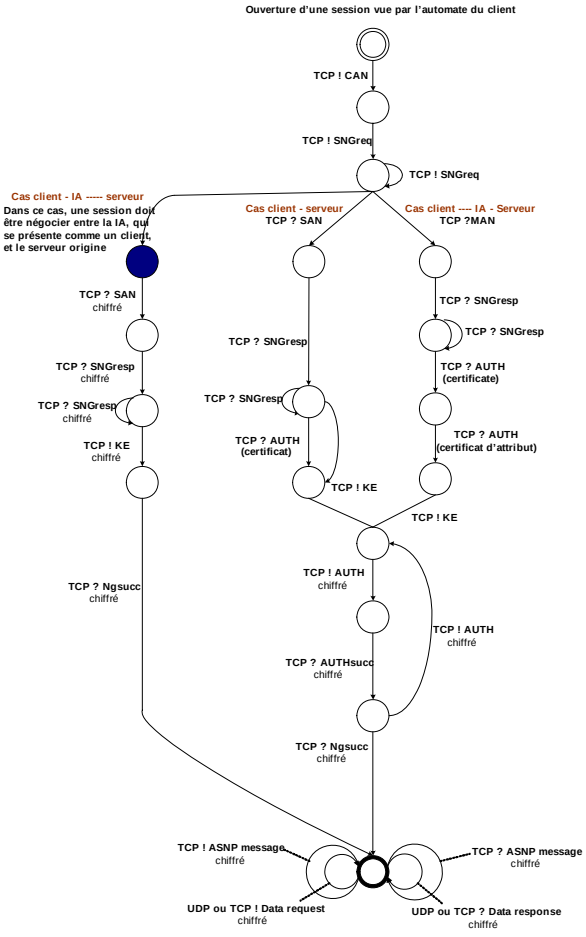
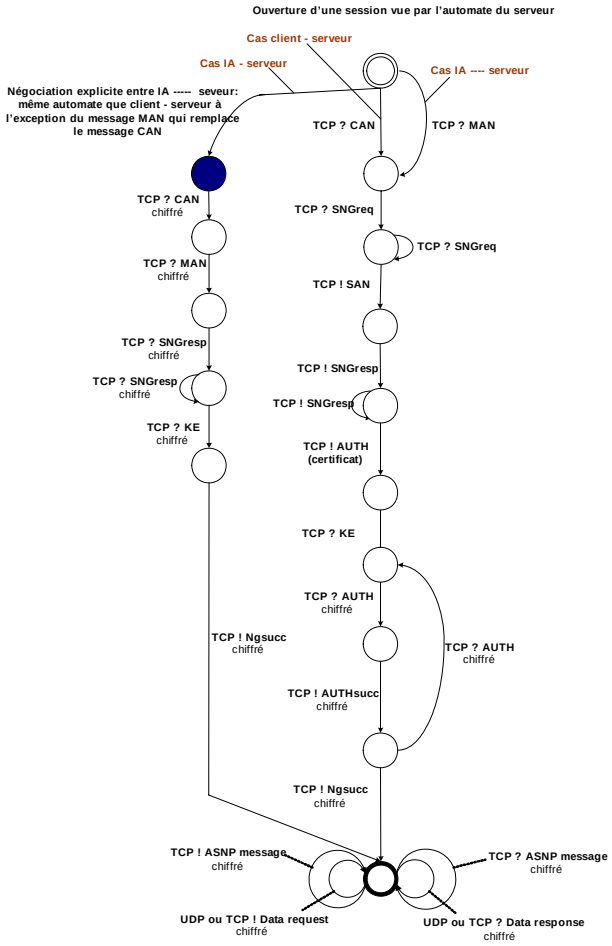


Figure 6-22. Echange SEP par l'intermédiaire d'une IA coté serveur

Pour que la IA prouve son rôle comme intermédiaire de confiance auprès de ce client, elle envoie deux messages *AUTHs*. Le premier message *AUTH* contient un certificat d'identité X.509 (ou autre type défini) et le deuxième contient un certificat d'attribut. Suivant les règles de sécurité définies au niveau de l'IA, et pour assurer une confidentialité de bout en bout, la IA envoie au serveur SEP les nombres aléatoires échangés dans *CAN* et *MAN* ainsi que le *pre_master_secret* chiffré avec la clé publique du serveur ou bien sous un chiffrement symétrique avec une ancienne clé session. Par ailleurs, nous proposons que la IA et le serveur origine se basent sur une clé session SEP déjà distribuée entre eux. Pour signaler au deux bouts de la communication le début de l'échange de données, la IA envoie au client et au serveur le message *NGsucc* contenant le hachage de toutes les informations échangées avec la IA durant cette négociation.



a) Ouverture d'une session vue par l'automate du client



b) Ouverture d'une session vue par l'automate du serveur

Figure 6-23. Automates d'un client et d'un serveur SEP

Les deux automates ci-dessus décrivent l'ouverture d'une session SEP vue par un client (figure 6-23 a) et un serveur SEP (figure 6-23 b).

Le client SEP est l'initialisateur de la connexion SEP vers une IA ou un serveur. La IA peut être un Proxy auquel le client envoie sa demande de connexion. Il attend que la IA établie une session sécurisée avec la IA ou le serveur distant pour lui faire passer tous les paramètres négociés. Un client qui s'adresse directement à un serveur SEP doit prouver son identité pour chaque service qui nécessite une authentification.

Le serveur SEP peut recevoir des connexions venant d'un client ou d'une IA SEP. Le serveur qui reçoit le message CAN sait qu'il doit procéder à une session SEP complète avec authentification et négociation de chaque service demandé. Avec la IA SEP, le serveur peut établir une session SEP semblable à celle établie entre un client et un serveur SEP. Après cette négociation, le serveur peut recevoir d'une IA SEP, d'autres messages ASNP résultats de la demande de connexion établie par des acteurs externes vers une IA proche du serveur.

Notons que les messages *SNGreq*, *SNGresp* et *AUTH* sont toujours en boucle puisqu'ils sont envoyés plusieurs fois durant la même session. Cependant, le message *AUTH* est toujours suivi d'un message *AUTHsucc* qui indique si l'authentification a réussi ou pas.

6.11.4.7 Connexion client – IA ----- serveur

La figure 6-24 montre un scénario symétrique de celui expliqué précédemment. Dans ce scénario, nous proposons que le client se base sur une ancienne session SEP établie avec la IA pour ouvrir une nouvelle connexion vers un serveur SEP externe. Le client met dans le premier message un *LSID* non nul. Si le *LSID* n'a pas expiré et suivant la politique de la IA qui définie le niveau de sécurité nécessaire pour établir des connexions externes, la IA commence une négociation d'une session SEP avec le serveur origine distant. Notons que le message *CAN* envoyé par la IA, doit inclure explicitement les mêmes services demandés par le client mais il contient implicitement leurs paramètres de sécurité.
Le client, qui ne participe pas à la négociation SEP, attend que la IA termine la négociation pour lui envoyer le résultat de la négociation à travers les messages *SAN*, *SNGresp* et *KE*. Le message *NGsucc* envoyé par la IA au client, informe ce dernier qu'une session SEP sécurisée vient d'être établie entre le client et le serveur SEP.

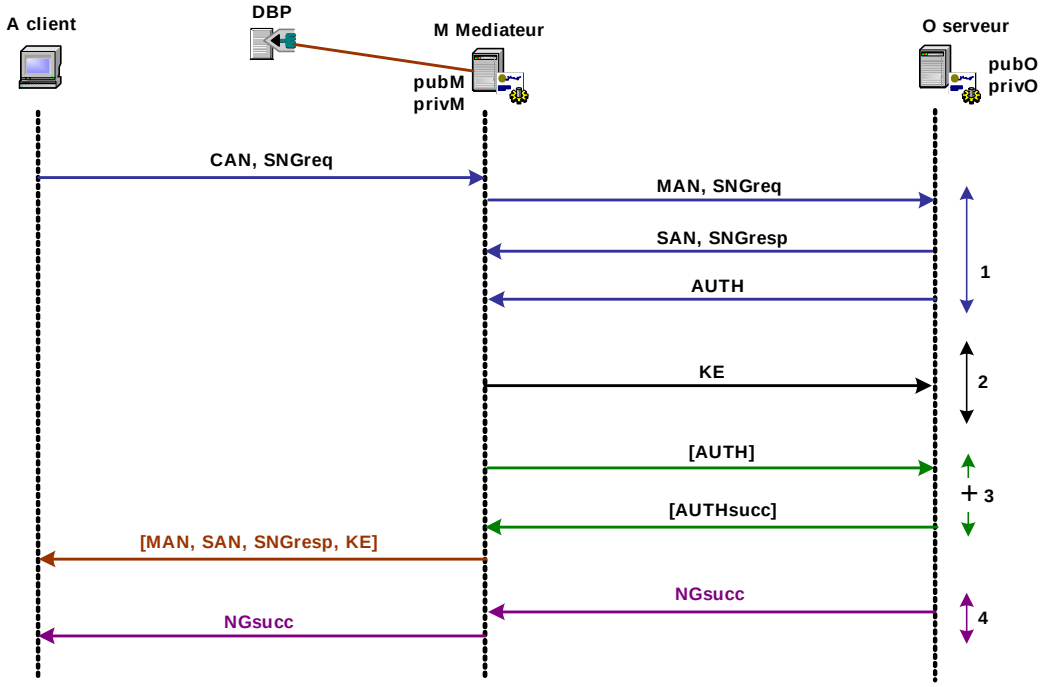


Figure 6-24. Echange SEP par l'intermédiaire d'une IA coté client

Le fait que chaque client génère son propre *LSID* permet à la IA de multiplexer plusieurs clients internes qui souhaitent se connecter vers un même service externe. Ces clients partagent entre eux les valeurs suivants : GSID généré par le serveur, les numéros aléatoires et la clé partagée.

Quelques notes :

- Le client doit mettre un *LSID* non nul pour indiquer l'utilisation d'une ancienne session SEP établit avec la IA.
- Si la IA n'a pas de session déjà établie avec le serveur, elle utilise le numéro aléatoire envoyé par la première demande de connexion envoyée par le client vers le serveur demandé. Dans le deuxième cas, le message CAN est abandonné.
- La IA peut initialiser le *LSID* de son premier échange avec le serveur à zéro. Cependant, une fois la session est négociée, le *LSID* prend la valeur générée par le client.
- Si le client souhaite établir un tunnel non partagé avec d'autres clients, il lui suffit de mettre le *Flag* de son message *SNGresp* à 0x01. Ce Flag correspond à l'extension *cipher_list*. Ainsi, la IA négocie les paramètres de sécurité du client et pas ces propres un.
- Un client qui n'inclut pas l'extension *cipher_list* avec le service demandé, implique un multiplexage automatique du tunnel SEP avec d'autres clients.

6.11.4.8 Connexion client – IA ----- IA - serveur

Dans le dernier scénario (figure 6-25), la négociation se déroule entre deux IAs placées aux deux bouts de la communication. L'avantage de ce scénario est l'établissement d'un tunnel sécurisé entre deux réseaux différents. Ce tunnel peut multiplexer le trafic généré par plusieurs applications d'un client ainsi que le trafic généré par plusieurs clients.

La centralisation des politiques de sécurité au niveau des IAs permet à la IA SEP d'identifier les serveurs d'applications qui proposent au client la même authentification (même clé partagée, même certificat, etc.). Ceci permet d'assurer, au niveau des mécanismes d'authentification des clients, un service de Single Sign On avec une authentification unique partagée entre plusieurs services.

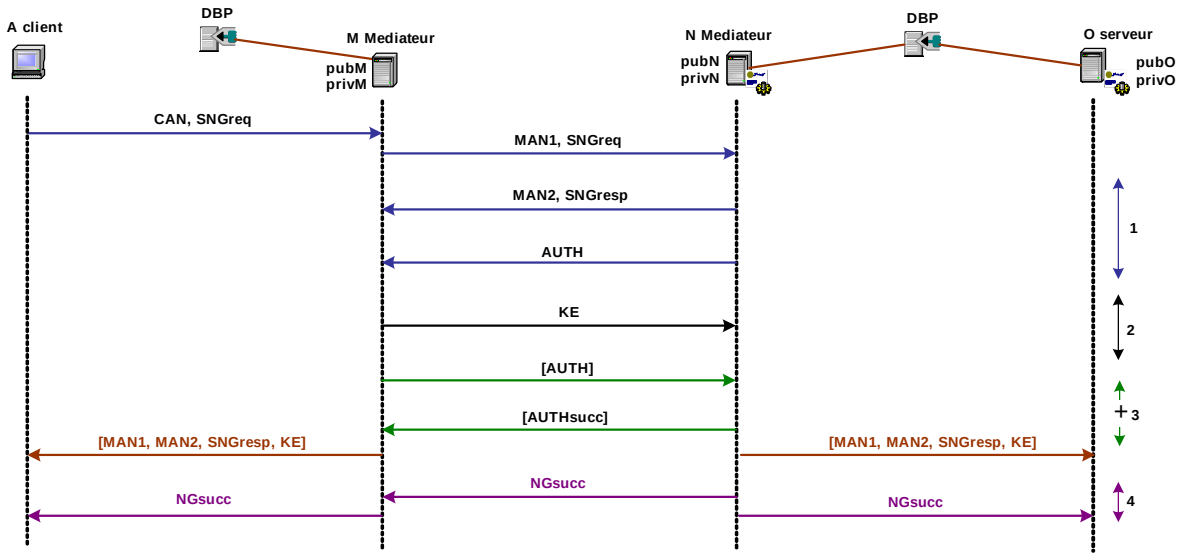


Figure 6-25. Echange SEP par l'intermédiaire de deux IAs

L'automate ci-dessous (figure 6-26) décrit les différents scénarios de communication qui peuvent être établis à travers une IA intermédiaire.

Une IA SEP attend la réception de message CAN pour initialiser l'ouverture d'une session SEP vers un serveur ou une autre IA SEP. Plusieurs scénarios sont possibles. En effet, La IA

peut recevoir des messages SEP venant d'un client ou d'une autre passerelle SEP. Dans le premier cas (connexion avec un client), la IA doit procéder à une double authentification pour prouver son rôle d'authentificateur habilité. Dans le second cas où la connexion est entre deux IAs, la première IA aura le même rôle qu'un simple client SEP tandis que la deuxième joue le rôle d'une réelle IA de confiance. Dans le cas où la communication est entre plusieurs acteurs, la IA SEP doit terminer la négociation avec des messages *NGsucc* envoyés vers les deux extrémités de la communication. Notons que la IA reste capable de recevoir et d'émettre des messages ASNP pour le renouvellement des clés ou l'ouverture de nouvelle session SEP.

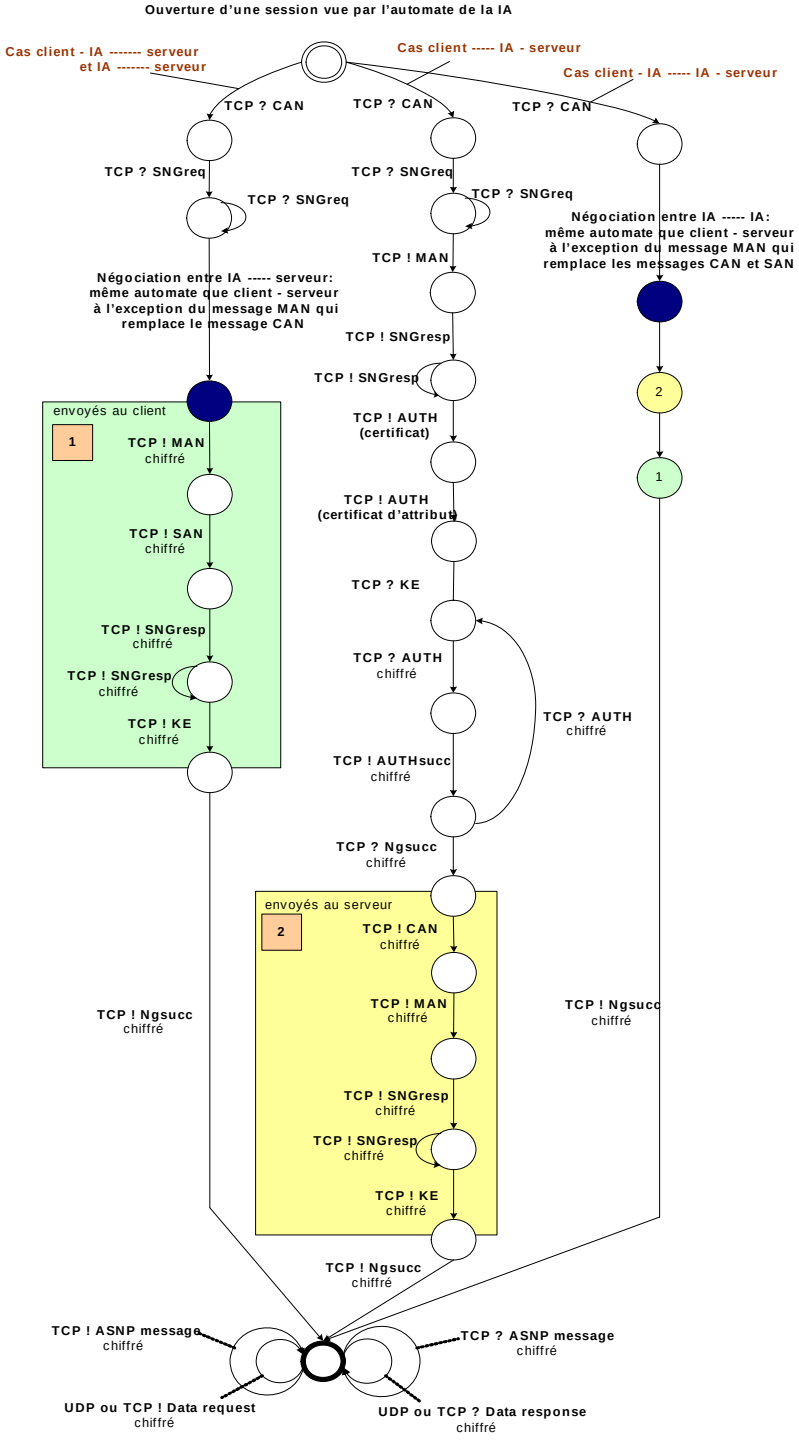


Figure 6-26. Automate d'une session vue par l'automate d'une IA

6.11.5 Le protocole alarme

Le protocole alarme (ALARM) contient une demande d'information, une erreur ou le statut associé à la négociation des sessions SEP. Les messages d'alarmes possèdent le type ALARM défini dans l'en-tête des paquets SEP et l'identificateur session (SessionID) auxquels il appartient. Il opère au dessus de TCP en utilisant le port 302.
Quelques alarmes sont figées dans le protocole SEP afin de garder une interopérabilité entre les différentes versions de SEP. D'autres peuvent être ajoutées par chaque application.
En effet, les messages figés par SEP sont des messages d'erreur qui possèdent un certain niveau de gravité. Selon la gravité de la menace, l'alarme peut constituer un simple avertissement ou déclencher l'abandon de la session. Un message d'avertissement est une simple mise en garde qui n'exige aucune action particulière. Cependant, à la suite d'un message fatal, l'entité émettrice doit immédiatement fermer la connexion avec le récepteur. De son côté, le récepteur quant à lui, ferme la connexion dès l'arrivée du message d'alarme.
Le protocole de gestion des alarmes peut être invoqué :

- par l'application, par exemple pour signaler la fin de la connexion.
- par le protocole ASNP suite à un problème survenu au cours de son déroulement.
- par le protocole de transfert SEP directement, par exemple si l'intégrité d'un message est mise en doute ou s'il y a un problème d'autorisation.
- par un service optionnel suite à un problème dans les messages échangés.

Les messages d'alarmes définis dans le protocole SEP sont les suivants:

Message	Contexte	Type
Service not supported	Demande d'un service inconnu	Fatal
Unknown Server	demande la connexion vers un serveur inconnu	Fatal
Negociation failure	Impossibilité de négocier des paramètres satisfaisants	Fatal
Invalid Version	Echec de la négociation sous la version demandée	Fatal ou avertissement
Record_overflow	Echec dans la taille des fragments SEP	Fatal
Decompression_failure	Echec de la décompression des messages SEP	Fatal
Insufficient_service_parameter	Les paramètres de sécurité d'un service demandé sont insuffisants	Fatal ou avertissement
Unexpected KE Syntax	Arrivée inopportune d'un message KE.	Fatal
Invalid Cert Encoding	Certificat erroné	Fatal
Expired Certificate	Certificat périmé	Fatal
Revoked Certificate	Certificat révoqué	Fatal
Bad _Certificate	Certificat invalide pour d'autres motifs que ceux précisés précédemment.	Fatal
Unknown Certificate Authority	Le certificat de l'autorité de certification est inconnu	Fatal
Invalid Cert Authority	Echec de la vérification du certificat de l'autorité de certification	Fatal
Authentication Failed	Echec de l'authentification	Fatal
Bad_Mac	Echec de la vérification du MAC des messages SEP	Fatal
Invalid Signature	Echec de la vérification d'une signature	Fatal
Invalid Key Enctyption	Echec du déchiffrement d'un message	Fatal
Invalid Authentication	Echec de l'authentification du client. Les implémentations doivent définir le nombre d'essais pour chaque authentification	Avertissement
Insuffisient_security	Signale que le niveau de sécurité demandé (algorithme, clé, etc.) est insuffisant.	Fatal ou avertissement
Expired Session	Echec de la reprise d'une ancienne session par motif d'expiration.	Fatal
Illegal_parameter	Un paramètre échangé au cours du protocole ASNP	Fatal ou avertissement

	dépasse les bornes admises ou ne concorde pas avec les autres paramètres	
Close_notify	Interruption volontaire de la session	Fatal
Session_life_time_alert	Envoyé pour avertir de la fermeture de la connexion.	avertissement

6.11.6 Le gestionnaire de politique d'accès

6.11.6.1 Principe de fonctionnement

La politique de sécurité d'un système informatique permet de spécifier les actions autorisées dans ce système. Pour contrôler l'accès aux informations sensibles d'un système, une politique de sécurité doit identifier les ressources du système contenant les informations sensibles et les opérations permettant d'accéder à ces informations.
Une politique de sécurité repose sur :

– La définition des ensembles de ses ressources et des opérations permettant d'accéder aux ressources.
– La définition de l'ensemble des règles pour le contrôle d'accès aux ressources.

Nous pouvons distinguer entre deux types de sécurité : la sécurité centralisée et la sécurité distribuée. Dans le système centralisé, il existe une unique politique de sécurité prenant en compte l'ensemble des entités du système. La sécurité distribuée se caractérise par la coexistence de nombreuses politiques de sécurité [GGKL89]. L'un des problèmes de ce type de sécurité concerne l'interaction des différents systèmes. Chaque système pouvant avoir ses propres contraintes de sécurité.

SEP propose un mécanisme de fédération des politiques de sécurité au niveau de l'autorité intermédiaire SEP. La fédération des politiques de sécurité permet de :

- *Garantir que la politique résultante préserve les conditions de sécurité associées à chaque politique initiale.*
  Dans SEP, chaque serveur application doit définir sa politique de sécurité. Cependant, une IA SEP qui protège plusieurs serveurs doit assurer que la politique d'une application ne réduit pas celle de l'autre. Ainsi, des systèmes distribués ayant une sécurité centralisée interagissent dans un environnement distribué sans réduire le niveau de sécurité de chacun des systèmes.
- *La politique résultante permet une cohabition avec des politiques plus globales (liées au réseau, à l'entreprise, etc.).*
  Un modèle de fédération permet une cohabitation de la politique de chaque application et une politique de sécurité plus générale correspondant à la sécurité de l'environnement à protéger.

Dans le cas d'un système SEP distribué (plusieurs serveurs origines), une IA SEP doit assurer la centralisation de la base des politiques de sécurité.

6.11.6.2 Plan et procédure de contrôle

La vérification de l'application de la politique de sécurité consiste à définir un contrôle de sécurité sur la base des politiques de sécurité (DBP) définies par les serveurs origine. Si les serveurs origines sont des entités internes reliées par une IA SEP, un deuxième contrôle de sécurité au niveau du réseau serait nécessaire.

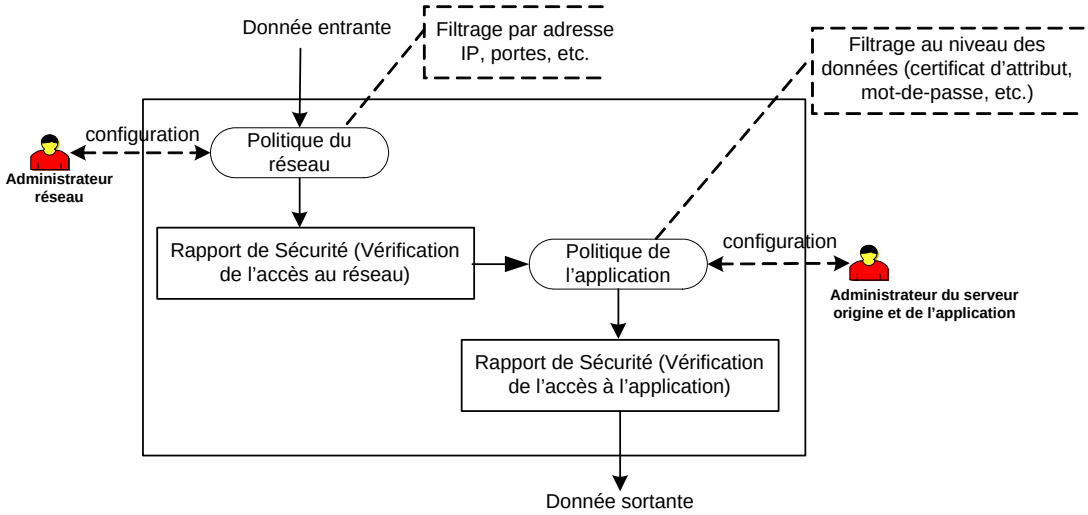


Figure 6-27. Procédure de vérification des configurations des machines SEP

Un ensemble d'étapes doit être réalisé afin d'obtenir le résultat escompté, comme illustré à la figure 6-27.
La première étape pour la mise en œuvre du plan de contrôle de sécurité, consiste à centraliser à l'IA SEP la configuration des serveurs origines. Nous nous intéressons aussi aux besoins des administrateurs qui souhaitent contrôler l'ensemble des flux entrants et sortants sur le réseau. Par rapport au serveur d'application demandé par les clients, l'administrateur réseau effectue son premier filtrage en se basant sur les adresses IP source et destination ainsi que le nom de l'application. Une fois ce contrôle effectué, les acteurs SEP commencent à négocier les paramètres de sécurité de chaque application (algorithme de chiffrement, méthode d'authentification, etc.). Ceci constitue un deuxième contrôle effectué au niveau applicatif et dépend des mécanismes d'autorisation de chaque service.
La figure suivante (figure 6-28) montre le modèle relationnel de la base de données DBP. Certaines parties concernant les détails de cette base de données (type et taille des champs, valeurs, etc.) sont reprotées à la partie 5 de l'annexe C.

Dans la figure 6-28 les syntaxes suivantes sont utilisées :

Table 1 → Table 2 : Signifie que Table 2 possède une ou plusieurs instances de la table 1.
CP : Identifie uniquement une table.
In et CEn : Valeur indexée.

Un administrateur doit commencer la configuration de sa base de données DBP à travers les deux tables Mediator et Server. La table Mediator définit la IA installée avec son adresse IP (*ip_address*), son DNS (*ia_name*) et son certificat de délégation (*delegation_certificate*). Pour chaque IA, l'administrateur doit définir un ou plusieurs serveurs d'application SEP. Chaque serveur possède une adresse IP (*ip_address*) et un nom (*server_name*). Un serveur SEP peut héberger un ou plusieurs services définis dans la table *service*. La table *service* est le centre des relations dans notre base de données. Elle possède un nom (*service_name*) et un flag (*internal*). Flag est égal à true si le service est interne à l'entreprise qui doit être protégée par un ensemble de paramètres de sécurité (méthodes d'authentification, messages alarmes, etc.). Si le flag est égal à false, le service est alors un service défini en dehors des frontières réseau

mais qui demande un contrôle de sécurité explicite au niveau des connexions demandées par les clients internes.

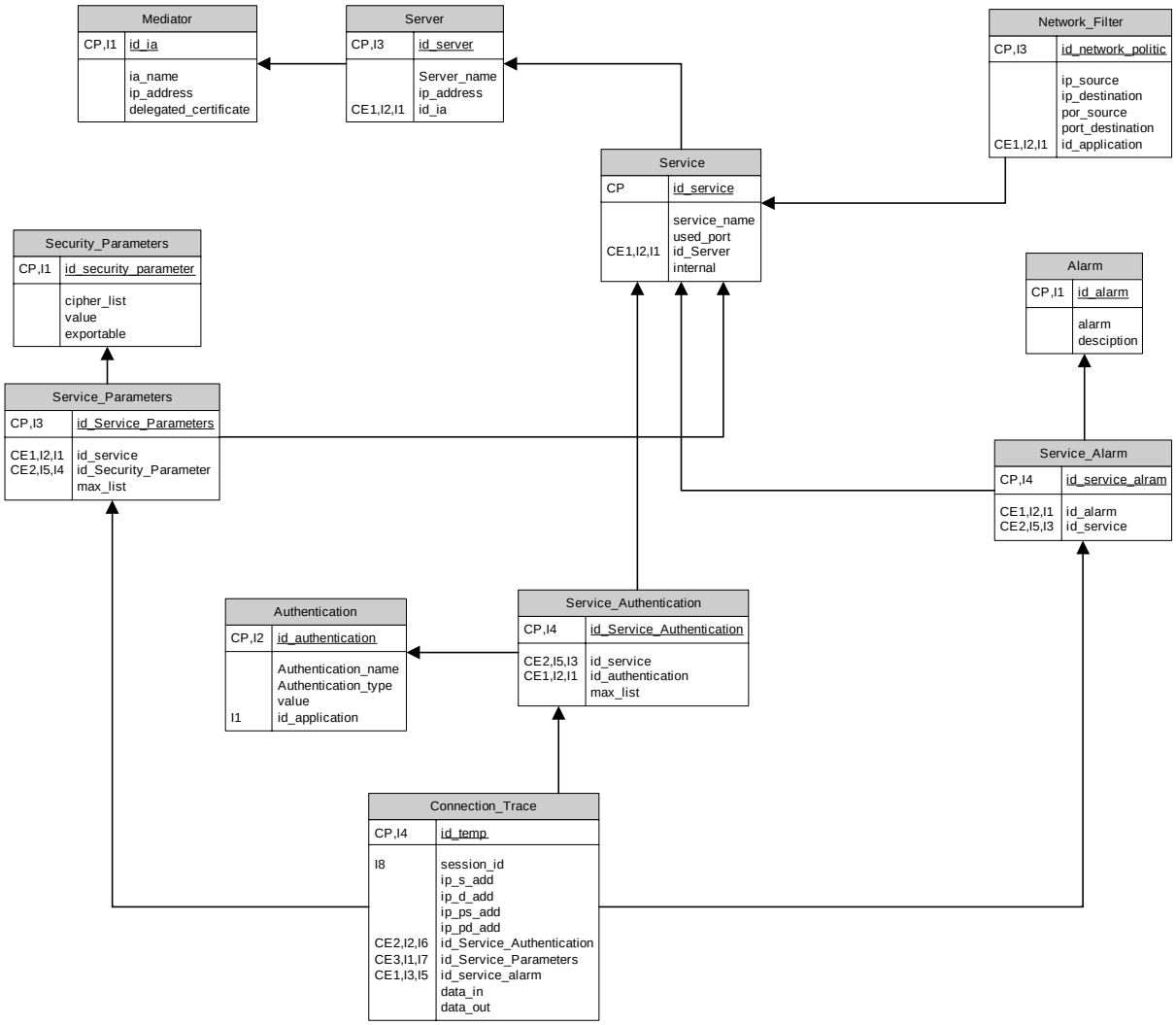


Figure 6-28. Modèle relationnel de la base de données DBP

Un service SEP est relié au trois tables *service_alarm*, *security_parameters* et *authentication methods*. Elles sont initialisées avec leur contenu avant la table *service*. Dans les implémentations, ces trois tables peuvent être figées durant l'installation de SEP. La table *service* communique avec ces trois tables à travers des tables intermédiaires qui définissent les paramètres de sécurité pour chaque service. Les trois tables intermédiaires sont :

- *service_authenication* : définit les méthodes d'authentification qui peuvent être utilisées pour chaque service.
- *service_alarm* : définit les alarmes nécessaires au bon fonctionnement de chaque service.
- *service_parameters* : choisit les paramètres de sécurité pour chaque service.

Finalement, une politique au niveau réseau est définie dans la table *Network_Filter*. A travers la liste des relations déjà établies, cette table peut être utilisée par le *Proxy* ou le serveur origine (*Server*).

DBP permet de stocker aussi les différentes connexions effectuées pour chaque service. En effet, la table *Connection_trace* permet d'identifier les sessions établies vers chaque service, les paramètres de sécurité de chaque service ainsi que l'archivage des données entrante et sortante du système si le service de non répudiation est activé.

6.11.6.3 Délégation à travers les certificats d'attribut

Les certificats d'attributs ont été créés pour résoudre les problématiques des certificats d'identité. En particulier, la notion "de lier une clé à une identité par un certificat" est abandonnée avec les certificats d'attribut. Le rôle d'un certificat est plus général grâce à l'attribution de permissions au possesseur d'une clé. Un certificat d'attribut contient donc un ensemble d'attributs qui donnent des informations sur les privilèges du possesseur du certificat.

Ainsi avec SEP, le serveur origine SEP fournit un certificat d'attribut à la IA SEP pour qu'elle puisse effectuer en son nom des opérations pendant une durée déterminée. Cette opération est plus précisément l'authentification des utilisateurs.

Le format de certificat d'attribut que nous utilisons avec SEP est celui développé dans le projet ICARE [ICARE]. Les certificats ICARE utilisent le format XML-DSIG qui permet d'ajouter une ou plusieurs signatures au même certificat d'attribut [Bern02] [Deme04]. Pour cela, les IAs SEP possèdent un seul certificat d'attribut qui contient la signature de plusieurs serveurs origines avec les rôles délégués par chaque serveur.

En général, le certificat d'attribut est composé de l'identificateur de l'émetteur (serveur origine), de l'identificateur du propriétaire (la IA SEP), des droits à donner (attributs), du temps de validité du certificat et de la capacité de déléguer à un tiers. L'émetteur et le propriétaire sont identifiés par leurs adresses IP, leurs noms de domaine et leurs clés publiques.

6.11.7 SEP et le protocole de transport UDP

6.11.7.1 Le protocole de transfert

6.11.7.1.1 Anti-rejeu

Avec UDP, l'en-tête des messages SEP contient un numéro de séquence pour fournir une protection contre le rejeu des datagrammes. Dans SEP, la vérification du numéro séquentiel ressemble à celle utilisée dans la section 3.4.3 du [RFC2402].

Chez le récepteur, le compteur de fragments reçus dans une session SEP, doit être initialisé à zéro quand la session est établie. Pour chaque fragment SEP reçu, le récepteur doit vérifier que le fragment contient un numéro de séquence qui ne reproduit pas le numéro de séquence d'un fragment SEP reçu pendant cette session. Ceci devrait être le premier contrôle appliqué aux fragments après qu'ils ont été affectés à une session SEP.

Les duplications sont rejetées par l'utilisation de la technique des fenêtres glissantes (sliding window). Une taille minimale de 32 paquets pour la fenêtre glissante doit être soutenue mais une taille de 64 paquets est préférable [Resc03].

Puisque cette technique est implémentée localement sur les machines, les entités communicantes n'échangent pas la taille des fenêtres et peuvent, par conséquence, avoir des fenêtres de taille différente.

La limite (bord) droite de la fenêtre glissante représente la valeur valide la plus élevée du numéro de séquence reçu dans la session en question. Les fragments SEP qui contiennent des numéros de séquence inférieurs à ceux du bord gauche de la fenêtre sont rejetés. Afin

d'exécuter ce contrôle, des moyens efficaces basés sur l'utilisation du bitmask sont misent en place. Ils sont présentés dans l'annexe C du [RFC2401].
Si le fragment SEP reçu faisant parti de la fenêtre est nouveau, ou si le paquet est à droite de la fenêtre, alors le récepteur procède à la mise à jour de sa fenêtre glissante.

6.11.7.1.2 PMTU Discovery

Dans un protocole de sécurité, les messages doivent être transmis et reçus dans un ordre défini afin de faciliter la vérification cryptographique des messages. En revanche, l'utilisation d'UDP comme protocole de transport non fiable, nous pousse à fournir des solutions aux deux problèmes suivants:

- La probabilité non nulle de perte des paquets (étudiée au paragraphe suivant).
- La fragmentation des données supérieures à la taille d'un fragment UDP.

Dans le cas du protocole UDP, on sait que la taille maximale d'un datagramme UDP est de 1500 octets. Pour les données chiffrées et transférées par le protocole SEP, la fragmentation pose deux importants problèmes. D'une part, un datagramme partiel ne peut pas être compris par le récepteur. En effet, les champs SEP se trouvent au début et à la fin. Le scellement nécessaire à l'authentification se trouve également à la fin du message SEP.
D'autre part, avec certains algorithmes, le déchiffrement est chaîné. Autrement dit, le déchiffrement d'un bloc de bits dépend du déchiffrement des blocs précédents.

Pour cela, nous proposons que chaque message SEP soit de taille inférieure ou égale à 1500 octets afin d'éviter une coûteuse fragmentation des données.
Pour calculer la taille maximale qui peut être envoyée par chaque entité SEP, nous utilisons dans SEP la méthode adaptée par IPSec ESP pour déterminer le MTU (Maximum Transmission Unit) est la taille maximale d'un paquet sur un lien) [RFC1191] accessible.
Pour effectuer le découvert du MTU, l'expéditeur peut donc activer le bit DF du paquet IP (*Don't Fragment*) [RFC2460]. Un routeur qui ne peut acheminer un datagramme *non-fragmentable* envoie à son émetteur un message ICMP « *size too big* » avec la valeur de la MTU.
L'idéal est donc de déterminer au préalable la taille maximale du datagramme qui entraînera son acheminement jusqu'au destinataire. On parle de découverte de PMTU (Path MTU). La PMTU est le minimum des MTU des liens sur le chemin vers le destinataire.
Une méthode pour calculer cette PMTU s'appelle *PMTU Discovery*. Elle consiste à envoyer des messages relativement grands afin de recevoir des retours ICMP *size too big*. Par diminution successive, on arrive à déterminer la PMTU. Le problème vient justement des messages ICMP. Ils offrent une faille permettant aux personnes mal intentionnées de faire du déni de service en envoyant de faux messages ICMP avec une taille trop faible.
A cette méthode, plusieurs alternatives sont proposées :

- On peut par exemple fixer une « valeur plancher » (576 octets typiquement) et considérer qu'en dessous, il s'agit d'une attaque. Si la MTU contenue dans le message ICMP est supérieure à cette valeur planchées, on la traite normalement (on adapte). Sinon, on néglige le message.
- On peut se renseigner « manuellement » sur les valeurs des MTU de tous les liens du réseau à traverser et négliger tous les messages *size too big*. Cette stratégie est utilisable

lorsque le réseau est maîtrisé mais elle n'a aucun intérêt lorsque le réseau à traverser est Internet.

Une implémentation SEP permet aux applications de négocier et de paramétrer le découvert PMTU comme un service optionnel négocié durant la phase d'initialisation de SEP. SEP peut permettre à l'application de ré-effectuer de temps en temps la découverte de PMTU. Ceci remettra à zéro le PMTU. L'exécution de telles demandes devrait être limitée, à une chaque deux secondes, par exemple.

6.12 Conclusion

SEP est un protocole de sécurité jeune dans le domaine de la sécurisation des échanges. Avec une architecture flexible et modulaire, notre protocole à pu répondre aux nombreux besoins en sécurité tels que la protection d'identité des usagers, le mise en œuvre de VPN, la délégation des rôles et le contrôle d'accès.
L'utilisation future de ce protocole pour la sécurisation des applications implique plusieurs avantages :

1. *Le client peut exiger la sécurité*
   À la différence des autres solutions comme SSL/TLS et grâce à l'approche de négociation par politique, chaque entité SEP peut imposer son niveau de sécurité. Chaque service négocié possède un flag qui indique si les paramètres demandés par cette entité sont critiques ou non.

2. *Le protocole SEP s'adapte à plusieurs types de contextes*
   Avec quatre architectures différentes, et plusieurs scénarios de communication, notre protocole a pu s'adapter à plusieurs environnements fixes ou mobiles. En effet, l'architecture « client – IA ----- serveur » envisage de simples opérations cryptographiques du côté des clients. Par ailleurs, si la IA est du côté des serveurs, ces derniers seront complètement protégés avec un service de SSO qui réduit les coûts des différentes authentifications faites par les clients. D'autres entités qui souhaitent un niveau de sécurité assez élevé, peuvent toujours utiliser un échange DH avec un service de PFS.

3. *Extensible: rajout de nouveaux protocoles et services*
   Grâce à sa nature modulaire, de nouveaux protocoles et services peuvent être ajoutés à SEP sans effectuer aucun changement au niveau de sa structure. Les nouveaux protocoles peuvent être ajoutés au dessus du protocole TRANFER en utilisant des ports spécifiques définis pour chaque protocole. Ces protocoles doivent être définis dans la base de données DBP puis négociés grâce aux messages génériques du protocole ASNP.

4. *Plusieurs services sont négociables*
   Une caractéristique à SEP est son intégration des services de non répudiation et d'archivage de données. Ces deux services optionnels sont négociés ensemble à travers les messages SNGreq et SNGresp. L'archivage de données se fait à travers la base de données PDB.

Dans le chapitre suivant, nous validons cryptographiquement notre protocole SEP grâce à l'outil Hermes d'EVA [EVA]. Hermes est un outil de vérification automatique de la propriété d'un secret parfait. La technique d'EVA montre que tous les scénarios développés dans ce

chapitre répond aux exigences cryptographiques nécessaires pour un nouveau protocole de sécurité comme SEP.

Chapitre 7

7 Validation formelle du protocole SEP

7.1 Résumé

Dans ce chapitre, nous validons le protocole SEP avec une technique de vérification automatique basée sur le langage EVA (LEVA) et son outil Hermes. L'outil Hermes d'EVA a fait ses preuves et a permis de détecter de nombreuses attaques sur des protocoles cryptographiques [Domi01] [Blan01].
Nous avons utilisé cette technique pour la vérification d'une extension du protocole IKEv2 [Huss05] ainsi qu'un nouveau protocole de sécurité, similaire à SEP, mais qui opère dans les réseaux à domicile [Medd04].

7.2 Introduction

Parmi les différents aspects de la sécurité informatique, la cryptologie a pris une importance considérable. En effet, la cryptographie est un moyen datant de l'antiquité permettant de mettre des règles d'échange entre des entités communicantes pour assurer la confidentialité et l'authenticité des messages échangés.
Aujourd'hui, les distributeurs de billets, les machines ATM, les abonnements aux chaînes de télévision payantes et le commerce électronique, exigent un haut niveau de sécurité portant sur des propriétés de secret et d'authenticité, mais aussi sur de nombreuses autres propriétés comme la non duplication (des factures), la non révocation, l'anonymat, etc.
Différents protocoles cryptographiques ont été mis en œuvre pour répondre aux exigences demandées, mais la difficulté de la conception des protocoles vient du fait que les messages échangés peuvent être écoutés par une tierce personne, interceptés ou modifiés.

Afin d'assurer que ces protocoles sont protégés contre de pareilles attaques, un nouveau domaine de sécurité est apparu. C'est la modélisation et la vérification des protocoles cryptographiques.
Depuis 1990, plusieurs techniques ont été présentées [Blan01] [Möd03] [Denk98]. Nous présentons au cours de ce chapitre la technique EVA [EVA] [Securify-TR7]. Cette technique a fait ses preuves et a permis de détecter notamment les attaques sur les protocoles IKEv1 et Schröder [Blan01]. Enfin, nous vérifions le protocole SEP en utilisant cette technique.

Les objectifs que nous visons dans ce chapitre sont:

- Réécriture du protocole SEP ainsi que quelques autres protocoles dans un format cryptographique simple.

- Vérification automatique de ces propriétés au moyen d'un ou de plusieurs outils de vérification.
- Validation des différentes activités d'échanges (échanges entre deux ou trois entités)
- Explication claire des preuves de sécurité.

7.3 Technique de vérification

7.3.1 Propriétés de sécurité

Nous allons donner dans ce paragraphe, un aperçu des objectifs que les modèles définis dans la section suivante, essaient de vérifier.

7.3.1.1 Secret

On dit qu'une donnée S est secrète si la session qui contient la donnée S est indistinguable de toute session contenant une donnée S' en lieu et place de S [Cort03]. D'autre part, parmi les propriétés du secret, on distingue les propriétés de secret globales et locales. La donnée peut être secrète « tout le temps » ou bien peut être secrète « tant que la session correspondante n'est pas terminée ». La première notion est plus facile à modéliser puisqu'il suffit d'exprimer que l'intrus ne peut jamais déduire le secret. La seconde propriété demande un modèle plus précis qui permet d'exprimer les débuts et les fins de sessions.

7.3.1.2 Authentification

Comme dans la plupart des outils de vérification des protocoles cryptographiques, la vérification de la propriété d'authentification est liée à la vérification du secret. Ainsi, au lieu de prouver qu'un protocole de sécurité satisfait la propriété d'authentification pour laquelle il a été conçu, on vérifie souvent une propriété de secret dont on pense qu'elle est équivalente. B. Blanchet [Bla02] a fait un premier pas vers un lien formel entre les deux propriétés en associant à une propriété d'authentification particulière, une propriété de secret telle que prouver le secret suffit à assurer l'authentification [Cort03].

7.3.1.3 Anonymat

L'anonymat est le fait de ne pas être identifiable parmi un ensemble de personnes. Un protocole qui fournit cette propriété doit donc assurer qu'une personne extérieure ne peut pas lier les messages émis à l'identité de la personne qui émet le message ou à l'identité du transmetteur. Ainsi V. Shatikov et D.J.D Hughes [Cort03] proposent une définition reposant sur l'équivalence observationnelle. En effet, intuitivement, un protocole préserve l'anonymat de l'identité du premier participant (par exemple) si, quel que soit l'agent qui joue le rôle du premier participant, les protocoles obtenus sont indistinguables.

7.3.2 Difficultés de la vérification

La vérification des protocoles cryptographiques est un cas particulier de model-checking où les systèmes considérés sont des protocoles cryptographiques dans un réseau hostile et les propriétés à vérifier sont celles énoncées aux paragraphes précédents (secret, authentification, anonymat). Le problème de la vérification automatique des protocoles cryptographiques vient du caractère non borné des paramètres du système à vérifier :

- Le nombre de sessions d'un protocole (en parallèle ou en séquence) n'est pas borné.
- Le nombre de participants n'est pas borné.
- Les capacités mémoire d'un attaquant ne sont pas supposées bornées.

- La taille des messages que peut fabriquer un attaquant n'est pas a priori bornée.
- L'intrus peut générer un nombre arbitraire de nouvelles clés et de nouveaux nonces.

7.3.3 La modélisation

La représentation ordinaire des protocoles, appelée représentation intuitive, décrit uniquement le déroulement normal du protocole (sans attaques). Donc pour le vérifier, cette représentation est ambiguë.
L'étude des protocoles cryptographiques commence par une première étape de modélisation. Depuis cinq années environ, de nombreux modèles dédiés aux protocoles se sont développés [Blan01] [MR00] [pau98]. Dans le paragraphe suivant, nous présentons le modèle LEVA que nous utilisons pour la validation cryptographique de SEP. Cette technique permet de vérifier les propriétés de sécurité et ainsi de détecter les attaques possibles sur le protocole. Cette technique a fait ses preuves et a permis de détecter notamment les attaques sur les protocoles IKEv1 et Schroeder [Blan01].

7.4 Choix du modèle LEVA pour la validation du protocole SEP

7.4.1 Introduction

Le projet EVA vient pour corriger quelques défauts dans le modèle Dolev et Yao [Dela03], notamment l'aspect multisessions, la théorie équationnelle, l'authentification et les propriétés de sécurité et du commerce électronique.
Le commerce électronique pause des défis supplémentaires à relever. Des propriétés plus complexes que l'authentification sont à établir : non duplication des messages (factures) et non révocation des messages (commandes).
Un autre aspect plus complexe du commerce électronique est qu'il inclue typiquement, non pas deux entités de confiance, mais trois entités (l'acheteur, le vendeur et la banque) tels que tout sous-ensemble de deux entités se méfie du troisième. Il est donc plus délicat de séparer le monde en deux zones : les entités de confiance et l'intrus, toute entité étant considérée comme un intrus pour les deux autres.
Les processus LEVA se traduisent en des ensembles de clauses de Horn (en programmes Prolog.).
En principe, il suffit de lancer un interprète Prolog pour vérifier les protocoles cryptographiques EVA avec quelques modifications.

7.4.2 Le projet EVA

La vérification des protocoles cryptographiques se heurte à un premier problème : une trop grande diversité des modèles. En effet, chaque auteur a son modèle favori qui utilise souvent des constructions très particulières. Ces différents modèles sont le plus souvent incomparables du point de vue de l'expressivité : certains permettent les clés composées, d'autres non. D'autres encore permettent le chiffrement asymétrique, etc. De plus, les propriétés exprimées sur les protocoles sont dépendantes du modèle, voire même du protocole à vérifier.

EVA (Explication et Vérification Automatique) est un projet du Réseau National des Technologies Logicielles (RNTL) qui regroupe plusieurs partenaires industriels et académiques (Trusted Logic S. A, Laboratoire Spécification et Vérification, et Laboratoire Verimag) afin de développer des outils à base d'abstraction et de model-cheking pour la vérification cryptographiques des protocoles de sécurité. Ceci dans le cadre des Critères

Communs de sécurité qui constituent une norme internationale en matière de certification de composants logiciels.

EVA définit un ensemble de syntaxes abstraites (qui ont la forme de programmes définissant les actions des participants au protocole) pour lesquelles est définie une sémantique opérationnelle simple et une correspondance (c.à.d une procédure de traduction) entre les syntaxes concrètes et les syntaxes abstraites. Dans LEVA, la syntaxe concrète est destinée aux utilisateurs où la description des protocoles est assez simple. La syntaxe abstraite est destinée aux outils où les transactions sont définies pour chaque participant du protocole.

7.4.3 Outils de démonstration automatique

Plusieurs outils de démonstration automatique ont été mis au point ces dernières années. Ils se sont développés un peu partout en France, en Europe et dans le monde. Citons tout particulièrement les outils de preuve automatique CASPER [Low98], CASPEL [Mill] et EVA.
Mais ces outils ne vérifient actuellement qu'une seule propriété de sécurité : le secret. Autrement dit, tous les modèles de protocoles cryptographiques utilisés par les démonstrateurs, font l'hypothèse du chiffrement parfait [Dela03]. Sans avoir la clé de déchiffrement, on ne peut obtenir aucune information sur un texte chiffré.

EVA propose trois outils CPV [Goub00], HERMES [Bozg03] et SECURIFY [Securify-TR13] de démonstration automatique. Dans ce paragraphe, nous ne présentons que SECURIFY et HEMES.

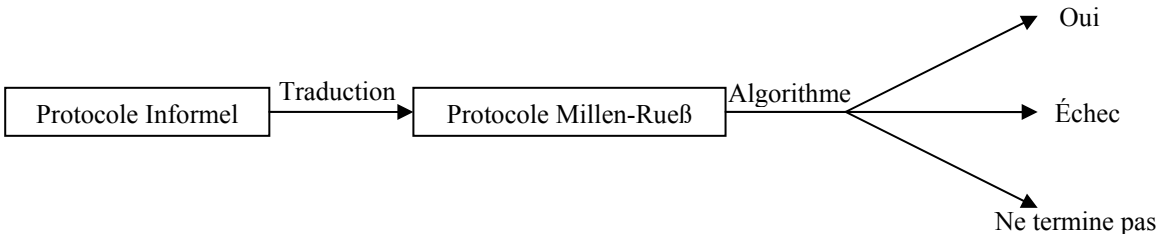
7.4.3.1 Securify

Securify [Securify-TR7] [Securify-TR13] permet une vérification automatique de la propriété de secret en se basant sur le modèle de Millen-Rueß présenté brièvement ci-dessous. La procédure de preuves est constituée de trois conditions suffisantes et une procédure de recherche en arrière.

« Si aucun de nos lemmes 'de base' ne permet de conclure au secret, on recherche alors toutes les instances des règles qui ont pu engendrer une partie des prémisses de la règle déjà considérée, et ainsi de suite. »

Cet outil a été entièrement programmé en Ocaml puis il a été intégré au langage LEVA. Le comportement de l'outil est sous trois formes :

- L'outil donne une réponse « oui » lorsque la propriété du secret à protéger est garantie dans le modèle Millen-Rueß.
- L'outil donne une réponse « echec » si la méthode de preuve échoue, on ne peut pas conclure si le protocole est sûr ou nom. Cependant, l'arbre d'échec de la preuve peut assez souvent permettre de construire une attaque et montrer ainsi que le protocole n'est pas correct.
- Si l'outil ne donne aucune réponse, alors l'algorithme ne termine pas.



Securify prend en entrée un fichier .cpl, dérivé par le parseur LEVA d'un fichier .eva dans lequel est décrit le protocole ainsi que les propriétés qu'il doit vérifier. Il permet également à l'utilisateur d'obtenir les graphes successifs des preuves (ou de l'echec des preuves) qui ont permis d'obtenir la réponse oui ou « don't know ». Ces graphes permettent de comprendre pourquoi la preuve du secret a réussit ou a échoué.

7.4.3.2 Hermes

Hermes [Bozg03] permet de prouver la propriété de secret de manière complètement automatique, pour un nombre non borné de sessions et de participants. Pour vérifier le protocole (figure 7-1), cet outil commence à extraire la propriété du *secret* et la spécification du protocole (module 1). Ensuite, il calcule une abstraction de la propriété et du protocole (module 2) puis complète l'ensemble des secrets en générant des contraintes qui définissent les conditions dans lesquelles le protocole peut être utilisé sans risque d'invalider les propriétés de secret (module 3). Un calcul symbolique des messages accessibles est ensuite effectué.

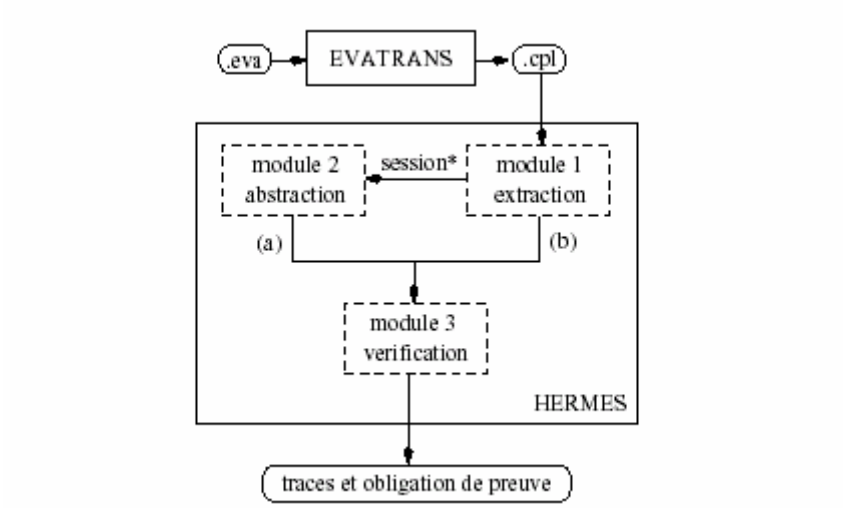


Figure 7-1. L'architecture de l'outil Hermes et la connexion au traducteur EVATRANS

L'outil produit également des traces qui justifient l'ajout de chaque nouveau secret et de chaque nouvelle contrainte. Ces traces correspondent à des tentatives d'attaques.
Le principe de vérification implanté dans HERMES est décrit en détail dans [Bozg02], nous en donnons ici une présentation succincte nécessaire à l'interprétation des résultats. Les contraintes calculées par le module 3 d'Hermes s'interprètent comme des limites imposées sur les connaissances de l'intrus au moment de débuter une session du protocole. La section 8.2 présente un exemple de contraintes. Ces contraintes sont établies de manière à garantir que les messages échangés par des participants honnêtes au cours d'une session du protocole jouée en parallèle avec un nombre arbitraire d'autres sessions, ne permettent pas à l'intrus de découvrir les secrets échangés par les participants honnêtes.

La méthode de calcul des contraintes du module 3 s'applique aussi bien à un nombre arbitraire de sessions parallèles (entrée a) qu'à un nombre fini de sessions parallèles (entrée b). Dans le premier cas, le module 2 d'Hermes construit une abstraction qui satisfait la propriété suivante. Si la propriété abstraite est vérifiée pour le protocole abstrait alors la propriété de secret est satisfaite par le protocole concret pour un nombre arbitraire de sessions. Ces dernières sont exécutées en parallèle par un nombre arbitraire de participants générant un nombre arbitraire de données fraîches (nonces et clés).

7.4.3.3 Principe de Hermes

HERMES calcule des conditions suffisantes pour garantir que les messages échangés au cours d'une session du protocole ne peuvent être exploités pour déduire des secrets. L'outil est basé sur la notion de messages capables de garder un secret, appelés messages protecteurs. Il s'agit des messages de la forme $\{X\}_K$, c'est-à-dire cryptés au moyen d'une clé dont l'inverse n'est pas connu de l'Intrus. Les messages protecteurs se déduisent des clés déclarées secrètes dans les hypothèses du protocole.

Étant donné un protocole, un ensemble *S* de secrets et un ensemble d'hypothèses sur les clés secrètes, HERMES calcule, d'une part, un ensemble de messages qui protègent les secrets échangés au cours du protocole ; et d'autre part, il ajoute aux secrets les messages qui permettent de construire une attaque. Il s'agit en particulier des messages dans lesquels les secrets ne sont pas protégés. À l'issue du calcul dont la terminaison est assurée par un opérateur de convergence forcée (widening), HERMES retourne un ensemble de secrets *S'* qui contient les secrets de départ et un ensemble de messages protecteurs *H'*. Les résultats *H' et S'* définissent les conditions dans lesquelles le protocole peut être utilisé sans risque. La propriété de secret est garantie si, dans les messages initialement connus de l'Intrus, les secrets *S'* sont protégés par les messages *H'*.

7.4.4 Terminologie de LEVA

7.4.4.1 Exemple du protocole SEP

Dans ce paragraphe, nous présentons la terminologie et les notations usuelles des protocoles cryptographiques, à travers le protocole SEP limité à deux acteurs (client – serveur).

Dans le protocole SEP, les deux entités A (client SEP) et B (un serveur SEP) essaient de partager une clé secrète Ka afin de chiffrer tous les flux de données entre elles. Quelques contraintes et notations sont à respecter:

- A et B sont appelées deux principaux.
- La notation : Message i. A -> B : M signifie que le message M est envoyé de A à B dans l'ordre i.
- La notation {M}_SK(B) représente la signature de la donnée M (ou un challenge) avec la clé privée de B, SK(B).
- La notation {M}_PK(B) représente le chiffrement asymétrique de la donnée M (ou un challenge) avec la clé publique de B, PK(B).
- La clé session Ka est générée par A puis envoyée vers B sous un chiffrement asymétrique avec la clé publique de B. La notation {Ka}_PK(B) représente cette étape.
- La notation {M}_Ka représente le chiffrement symétrique de la donnée M avec la clé secrète Ka.

- Avec SEP, seule l'authentification de B (le répondeur) est obligatoire (avec une certificat de signature).

Afin de préciser l'algorithme de chiffrement asymétrique que nous souhaitons, nous pouvons écrire : {Ka}_PK(B)^algo pour le chiffrement asymétrique avec l'algorithme RSA et {M}_SK(B)^algo pour la signature du message M en utilisant l'algorithme asymétrique (algo) RSA.

Si l'algorithme de chiffrement est suffisamment sûr (clé >1024 bits), la seule façon pour un intrus I de récupérer Ka, à partir de {Ka}_PK(B), est de connaître la clé privée de B (SK(B)) et la déchiffrer avec {Ka}_SK(B). Pour cela, nous supposons que tant que les deux entités gardent leurs clés privées secrètes, Ka ne peut pas être compromise par un attaqueur. Ka doit être renouvelée régulièrement durant chaque nouvelle session.

Même si le mécanisme ci-dessus semble être sûr, grâce à l'utilisation des méthodes de chiffrement asymétrique, il n'en est rien. Si la clé session a été utilisée depuis un temps suffisamment long, elle peut rentrer en la possession d'un intrus. L'intrus a eu tout le temps de trouver la clé K0 à partir des différents messages chiffrés {M}_K0 échangés entre A et B. L'intrus peut donc passer {K0}_PK(B) à B, sachant que B va confronter l'ancienne clé K0 avec la clé attendue Ka. Tout message M que B chiffrera (symétriquement) avec cette clé sera envoyé sous la forme {M}_K0 et non {M}_Ka, ce qui permettra à l'intrus de les déchiffrer et de récupérer M.

Pour empêcher cette attaque, on peut ajouter des données qui permettent à B de vérifier que les messages reçus sont récents. Une solution simple est l'utilisation d'un nombre aléatoire (*Nonces*). Puisque A et B ne se connaissent pas en avant, les deux principaux doivent échanger leur identité. En effet le protocole devient:

- B initie la session et tire un nombre aléatoire Na (le nonce).
- A envoie en clair Na et son identité A à B.
- B génère le nombre aléatoire Nb.
- B envoie les deux nombres aléatoires Na, Nb et sa signature sur ces valeurs afin de prouver son identité.
- A crée la clé Ka et envoie le message {Ka}_PK (B)^alg à B.

Ainsi, on note en abrégé ce protocole :

01	Message 1.	A -> B : Na
02	Message 2.	B -> A : Na, Nb, B, {Nb,Na}_SK (B)^alg
03	Message 3.	A -> B : {Ka}_PK (B)^alg

Ce protocole n'est pas encore sûr. En effet, un intrus I malhonnête peut jouer le rôle de B en signant avec sa clé privée SK(I) le deuxième message envoyé de B à A. Ainsi, A qui ne connaît pas la vraie identité de B va penser qu'elle communique avec B. A envoie donc la clé Ka chiffrée avec la clé publique de I. I déchiffre la clé et l'envoie à B chiffrée avec la clé publique de ce B.
Afin d'empêcher cette attaque, les clés publiques des communicateurs doivent être certifiées avec une autorité de confiance. Dans SEP, nous supposons toujours que les entités réceptrices (IA ou serveur origine) possèdent des certificats X.509.

En plus, puisque la relation (identité – certificat) est ambiguë. H. Krawczyk [Kraw03] [Canet01] propose d'ajouter les identités des deux communicateurs dans les messages signés, soit dans un nouveau message chiffré avec un HMAC calculé avec une clé dérivée de la clé Ka. Cette technique est utilisée dans le protocole IKEv1 qui offre aussi la protection des identités des communicateurs. Notre protocole SEP devient donc:

01	Message 1.	A -> B : Na, A
02	Message 2.	B -> A : Nb,B,{Nb,Na,B,A}_SK (B)^alg
03	Message 3.	A -> B : {Ka}_PK (B)^alg, {Nb,Na,A,B}_Ka
04	Message 4.	B -> A : {Nb,Na,A,B}_Ka

Finalement, comme expliqué dans les chapitres précédents, SEP fournit un service de protection d'identité pour les usagers seulement. Pour cela, le protocole devient :

01	Message 1.	A -> B : Na
02	Message 2.	B -> A : Nb,B,{Nb,Na,B}_SK (B)^alg
03	Message 3.	A -> B : {Ka}_PK (B)^alg, {Nb,Na,A,B}_Ka
04	Message 4.	B -> A : {Nb,Na,A,B}_Ka

7.4.5 Validation du protocole SEP avec LEVA et Hermes

Les règles décrivant le protocole SEP sont :

7.4.5.1 Règles générales

alg : asym_algo
everybody knows alg

Lorsqu'un protocole de sécurité utilise un algorithme de chiffrement asymétrique, nous devons définir cet algorithme d'une manière générique avec (alg : asym_algo). Nous supposons que cet algorithme est connu aussi bien par les intrus que par les participants. Nous notons cela par '*everybody knows alg*'.

Dans ce qui suit, nous définissons les participants à la communication. Dans une communication SEP avec deux entités, A désigne le client SEP, B le serveur et CA une autorité de certification. Dans une communication à trois entités, A désigne le client SEP, B la IA, C le serveur origine et CA l'autorité de certification. Avec SEP, les participants sont définis de la manière suivante :

A, B, C, CA: principal

Nous définissons les nombres aléatoires et les clés secrètes comme des clés de type Key. Les messages sont définis comme des variables de type *number*.

Basetype key
Na, Nc, Ka : key
O1, O2, P : number

Les clés publiques des participants sont représentées par PK(principal) et les clés privées par SK(principal).

keypair^alg SK, PK (principal)

Dans un échange à trois acteurs, nous supposons que les transactions entre le serveur origine et la IA sont protégées grâce à une clé (Kbc) générée dans une ancienne session SEP. La clé Kbc est définie comme une clé partagée entre les deux principales B et C.

shr (principal,principal) : number secret
alias Kbc = shr(B,C)

Les certificats des serveurs ou des IAs SEP contiennent principalement l'identité du participant, sa clé publique et la clé publique de l'autorité de certification. Ces informations sont signées avec la clé privée du CA. Pour cela, elles sont définies de la forme suivante :

alias certB = { CA, B, PK(B) }_SK(CA)^alg
alias certC = { CA, C, PK(C) }_SK(CA)^alg

Les certificats d'attribut sont aussi définis par un *alias*. Un certificat d'attribut ne contient pas la clé publique du détenteur mais son identité (nom, numéro série de son certificat d'identité, etc.). Il contient aussi l'identité de l'émetteur de ce certificat (généralement le serveur origine).

alias Acert = {C,B}_SK(C)^alg

Ici, nous définissons les variables session qui sont : Na, Nb, Ka.

s. session {Na,Nb,Ka} A=A, B=B*

Avec le symbole *, Hermes considère un nombre de session parallèle non borné. Nous pouvons limiter ce nombre pour des raisons de déboguage ou de limitation mémoire.

7.4.5.2 Connaissances initiales

Les connaissances initiales des participants représentent un ensemble de paramètres que chaque participant possède avant le début des sessions.

A knows A, CA,, PK(CA)
B knows B,CA, SK(B), PK(B), PK(CA),certB

Les deux connaissances ci-dessus sont pour un client (A) SEP et un serveur (B) SEP. Les deux principaux partagent, en avant, la connaissance d'un tiers de confiance CA, ainsi que sa clé publique. Chacun possède un certificat et une paire de clés publique/privée.
Finalement, nous supposons à travers l'utilisation de la syntaxe *assume* que les clés secrètes SK(B), SK(C), SK(A), SK(CA) sont connues seulement par leur propriétaire. En plus, la novelle clé session Ka et la clé de chiffrement Kbc (cas de 3 entités) utilisée entre le médiateur et le serveur sont des secrets.

assume secret(SK(B)@s.B),
secret(SK(C)@s.C),
secret(SK(CA)@s.CA),
secret(Ka@s.A),
secret(Ka@s.B),
secret(Kbc@s.B),
secret(Kbc@s.C)

Voici le programme décrivant le protocole SEP avec 2 entités. Ici le client A reste anonyme tandis que le serveur présente son certificat d'identité.

Début du fichier sep-2-entités.eva

```
01                                                    SEP_specification_in_Securify
02  alg : asym_algo
03  everybody knows alg
04
05    A, B,CA: principal
06    basetype key
07    Ka: key
08    Na,Nb,P, SNGreq, SNGresp: number
09
10    keypair^alg SK, PK (principal)
11
12    h(number) : number
13
14    alias certB = {CA, B, PK(B)}_SK(CA)^alg
15
16    A knows A, CA, B, PK(CA)
17    B knows B,CA, SK(B), PK(B), PK(CA), certB
18    CA knows CA, PK(CA), SK(CA),B, PK(B)
19
20     {
21      1. CA -> B:  certB
22      2. A -> B : Na
23      3. B -> A : Nb, certB
24      4. A -> B : {Ka}_(PK (B))^alg
25      5. B -> A : {Nb,Na,B,certB}_Ka
26
27     }
28
29     s. session* {Ka,Na,Nb} A=A, B=B, CA=CA
30       assume secret (SK(B)@s.B),
31              secret (SK(CA)@s.CA),
32              secret(Ka@s.A),
33              secret(Ka@s.B)
```

Fin du fichier sep-2-entités.eva

Notons que le premier échange (numéro 1) ASNP dans les deux scénarios de SEP commence avec une distribution du certificat envoyé du CA vers le serveur ou le médiateur. En effet, Hermes ne permet pas de faire une corrélation entre l'identité d'un participant et sa clé publique. C.à.d. si le premier message n'est pas présent dans l'échange, Hermes va trouver une attaque de type homme de milieu qui possède la même identité du serveur (B) mais avec

une autre clé K(I). De ce fait, l'intrus (I) peut envoyer au client un certificat de la forme suivante :

$$CertI = \{CA, h, PK(I)\}\_SK(CA)\text{^}alg.$$

Voici le programme décrivant le protocole SEP avec 3 entités. Ici, nous supposons que le client s'authentifie avec un certificat X.509. Pour le médiateur (B), il présente un certificat d'attribut (Acert) et son certificat d'identité (certB).

Début du fichier sep-3-entités.eva

```
01  SEP_specification_in_Securify
02
03  alg : asym_algo
04  everybody knows alg
05
06    A, B,C,CA: principal
07    basetype key
08    Ka: key
09    Na,Nc,P,accept: number
10    //certB,certC,Acert: number
11    keypair^alg SK, PK (principal)
12
13    h(number) : number
14    shr (principal,principal) : number secret
15    alias Kbc = shr(B,C)
16
17    alias certA = { CA, A, PK(A) }_SK(CA)^alg
18    alias certB = { CA, B, PK(B) }_SK(CA)^alg
19    alias certC = { CA, C, PK(C) }_SK(CA)^alg
20    alias Acert = {C,B}_SK(C)^alg  // certificat d'attribut: délégation d'un rôle d'habilitation de C à B
21
22
23    A knows A, B, C, CA, SK(A), PK(A), PK(CA), certA
24    B knows B, C, CA, SK(B), PK(B), PK(C), PK(CA), certB, certC, Acert, Kbc
25    C knows B, C, CA, SK(C), PK(C), PK(B), PK(CA), certC, certB, Acert, Kbc
26   CA knows CA, PK(CA), SK(CA), B, certB, C, certC, A, certA
27
28     {
29      1. CA -> B: certB
30      2. CA -> A: certA
31      3. A -> B : Na
32      4. B -> A : Nc,B,C,Acert,certB
33      5. A -> B : {Ka}_(PK (B))^alg
34      6. A -> B : {A,certA,{A,certA,Na,Nc,Acert,B,C}_(SK(A))^alg}_Ka
35      7. B -> A : {accept}_Ka
36      8. B -> A : {Nc,Na,C,B,A}_Ka
37      9. B -> C : {Nc,Na,A,Ka}_Kbc
38     }
39
40    //s. session* {Kap,Na,Nc} A=A, B=B, C=C, CA=CA
41      s. session A=A, B=B, C=C, CA=CA
42
43    assume secret (SK(B)@s.B),
```

```
44      secret (SK(A)@s.A),
45      secret (SK(C)@s.C),
46      secret (SK(CA)@s.CA),
47      secret(Kbc@s.B),
48      secret(Kbc@s.C)
```

Fin du fichier sep-3-entités.eva

En employant EVATRANS, la première spécification de SEP est compilée en un ensemble de règles qui définissent une session SEP générique. A partir de cette session, les principaux A et B sont remplacées par un seul participant honnête appelé H qui joue le rôle de tous les principaux. Un intrus I va essayer de forger les vrais messages par des valeurs Xi afin de reconstruire les anciens messages SEP envoyés dans d'autre session. Par conséquent, toutes les sessions dans lesquelles H n'est pas impliqué, sont capturées avec le modèle de Dolev et de Yao. Pour ce qui concerne les autres sessions où H est impliqué, nous identifions :

- toutes les sessions où H joue tous les rôles et qui sont différentes de la session fixe,
- toutes les sessions où H joue le premier rôle tandis que le deuxième rôle est joué par un autre principal et (3) toutes les sessions où H joue le deuxième rôle tandis que le premier rôle est joué par un autre principal.

La figure suivante montre un exemple des quatre premières règles du protocole SEP à deux entités. Un intrus I remplace le nombre aléatoire Nahh envoyé entre deux honnêtes H par NahI et qui forge la session par des messages x2. Il sera alors identifié par les participants honnête lors de la présentation des certificats d'identité avec (I,({(CA,(I,K(I)))}_SK(CA).

```
      h
r1 : ------------------------------
       Nahh
       h
r2 : ------------------------------
       NahI

       (x2,(h,({(CA,(h,PK(h)))}_SK(CA),{(x2,(Nahh,h))}_SK(h))))
r3 : ------------------------------
       ({Ka(hh)}_PK(h),{(x2,(Nahh,(h,h)))}_Ka(hh))

       (x2,(I,({(CA,(I,K(I)))}_SK(CA),{(x2,(NahI,I))}_K(I))))
r4 : ------------------------------
       ({K(I)}_K(I),{(x2,(NahI,(h,I)))}_K(I))
```

Les règles génériques utilisées par Hermes

7.4.6 Contraintes retournées par Hermes

7.4.6.1 Résultats récupérés avec Hermes

7.4.6.1.1 Résultat avec deux acteurs

Secrets : SK(h); Ka	
Good Patterns : {xs}_PK(h); {xs}_Ka	**Bad Patterns :** vide

7.4.6.1.2 Résultat avec trois acteurs

Secrets : SK(h); Ka; shr(hh);	
Good Patterns : {xs}_PK(h); {xs}_Kap; {xs}_shr(hh)	**Bad Patterns :** vide

7.4.6.2 Interprétation des résultats

L'ensembles des secrets S' retournés par HERMES ne contient pas de nouveaux secrets. L'ensemble H' des messages protecteurs retourné par HERMES est représenté comme la différence entre les instances de *Good patterns* et celles des *Bad Patterns* [Bozg02]. Les messages protecteurs de secrets sont ceux qui coïncident avec les *Good Patterns* privés des messages qui coïncident avec les *Bad Patterns*. Ainsi, dans le cas présent, les patterns indiquent que tout message chiffré avec l'une des clés Ka, shr ou PK, peut contenir un secret sans danger.

7.4.6.3 Obligation de preuve retournée par Hermes

Pour valider la propriété de secret, il reste à prouver que sous les hypothèses de la partie «assume...», les secrets S' sont protégés par l'ensemble H' des messages retournés par HERMES. Les arguments qui permettent de satisfaire l'obligation de preuve sont les suivants:

- Le secret Ka est généré durant le protocole. Il est donc frais et par définition, il ne peut apparaître dans aucun message préexistant. Il est donc trivialement protégé par n'importe quel message protecteur.
- D'après les hypothèses, les secrets SK(h) et shr(h) ne sont jamais envoyés (cela n'interdit pas que ces clés aient été utilisées pour crypter un message). Ces deux secrets sont donc trivialement protégés par n'importe quel message protecteur.

On peut alors conclure que la propriété de secret est vérifiée. Notons que les obligations de preuve produites par HERMES, se résolvent en général par des raisonnements similaires aux précédents qui font appel à des arguments de fraîcheur ou bien directement aux hypothèses. Cependant, cet outil se concentre sur un seul modèle d'attaquant, celui de Dolev-Yao [Dela03].

7.5 Conclusion

Divers travaux et études ont été réalisés pour éliminer les faiblesses de certains outils et pour résoudre les difficultés auxquelles ils font face. En effet, ceci devient de plus en plus indispensable vu que les systèmes informatiques sont de plus en plus souvent la cible d'actions malveillantes, de tentatives d'intrusion et d'attaques diverses.
L'utilisation d'EVA et de l'outil Hermes dans la vérification cryptographique de SEP montre que notre protocole est cryptographiquement sécurisé. Par ailleurs, nous envisageons dans un second temps de continuer la vérification de notre protocole avec d'autres outils [Bla02]. En effet, l'outil Hermes fut l'un des premiers à prouver des propriétés de secret dans le cadre d'un nombre de sessions non bornées et il nous a permis de vérifier de nombreux protocoles cryptographiques [Medd04]. Cependant, cet outil repose sur l'hypothèse du chiffrement parfait et il nécessite que les protocoles soient suffisamment typés.

Chapitre 8

8 Conclusion générale et perspectives

8.1 Conclusion générale

De nombreux mécanismes de sécurité ont été proposés pour les réseaux fixes et mobiles. Bien que ces mécanismes aient pu répondre à un ensemble d'exigences de sécurité, ils demeurent uniquement efficaces dans un contexte spécifique lié aux hypothèses et aux exigences restrictives qui ont été émises lors de la conception.

Nous pouvons diviser notre travail dans cette thèse en trois grandes parties :

1. L'analyse des solutions de sécurité existantes.
2. L'extension de quelques solutions de sécurité.
3. La conception et la validation d'un nouveau protocole de sécurité.

Dans la première partie (chapitres 2 et 3), nous avions pour but de définir une liste d'exigences en sécurité qui permet d'analyser les trois solutions de sécurité les plus déployées, à savoir les protocoles SSL/TLS, IPSec et SSH. Ceci nous a permis de comparer ces protocoles et de pouvoir arborer d'une part, leurs avantages et inconvénients et d'autre part, les exigences qu'ils ne peuvent pas satisfaire.

Dans la deuxième partie (chapitres 4 et 5), nous nous somme intéressés à étendre le protocole SSL/TLS. Grâce à sa simplicité de déploiement, SSL/TLS est actuellement le protocole d'authentification et de sécurisation des échanges le plus déployé. Nous avons proposé plusieurs modifications à ce protocole afin d'améliorer ses performances.

La première modification (chapitre 4) consiste à changer la phase d'initialisation du protocole SSL/TLS. Nous avons proposé de remplacer le protocole Handshake de SSL/TLS par le protocole de gestion des clés ISAKMP utilisé actuellement avec le protocole IPSec. L'intérêt de ce travail est de fournir, entre autres, l'unification des associations de sécurité et la mise en épreuve du protocole ISAKMP avec un nouveau protocole de sécurité. Il permet aussi à SSL/TLS de fournir de nouveaux services tels que la protection d'identité et l'authentification par des clés partagées.

La deuxième modification (chapitre 5) consiste à étendre les standards SSL/TLS (RFC 2246 et RFC 3546) existants avec de nouvelles fonctionnalités. Nous avons montré l'intérêt de l'intégration d'un module générique de non répudiation dans le protocole SSL/TLS. Nous

nous sommes basés sur le standard TLS Extension qui nous permet de garder toute interopérabilité avec la version standard de ce protocole.
Grâce aux besoins réels définis dans le chapitre 2 tels que la protection d'identité, l'extensibilité du protocole et la sûreté cryptographique, nous continuons actuellement à étendre le protocole SSL/TLS avec des nouveaux services. Nous avons proposé au sein de l'IETF un nouveau mécanisme d'authentification pour SSL/TLS basé sur les clés pré partagées [Bad04]. Ce travail converge actuellement, avec d'autres propositions [Eron04] [Gutm04], vers un standard commun qui regroupe plusieurs partenaires industriels, à savoir CISCO et NOKIA. L'intérêt de cette proposition réside dans l'utilisation de SSL/TLS dans la configuration à distance des équipements réseau (i.e. routeurs) qui ont un accès limité. Cette proposition possède trois avantages par rapport aux deux autres solutions ([Eron04] et [Gutm04]) : la protection de l'identité du client, le service de PFS (Perfect Forward Secrecy) et la resistance contre les attaques actives et par dictionnaire.

Cependant, ces propositions ne répondent pas à un certain nombre d'exigences que nous avons définies dans le premier chapitre, à savoir, la mise en œuvre d'un VPN au niveau de données, l'autorisation, le contrôle d'accès, etc. Ainsi, il a fallu concevoir un nouveau protocole de sécurité qui ne prend pas en compte ni les anciennes versions de chaque protocole ni les choix d'implémentation et de conception qui ont été faits. Pour cela, nous avons proposé et validé dans la troisième partie de cette thèse (chapitres 6 et 7) un nouveau protocole de sécurité nommé SEP (Secure and Extensible Protocol) qui intègre nativement l'évolution des protocoles de sécurité des échanges et des réseaux d'une manière performante.

Notre protocole SEP est basé sur SSL/TLS afin de fournir les services de sécurité de base tels que la confidentialité, l'intégrité et le partage d'un secret. En plus, Il permet l'établissement d'une session sécurisée par l'intermédiaire d'une autorité de confiance (IA) et la mise en œuvre d'un VPN au niveau de données et l'autorisation. Nous avons validé ce protocole en utilisant l'outil Hermes d'EVA.

Toutefois, d'autres travaux sont nécessaires avant que SEP soit réellement exploitable dans un environnement à fortes contraintes de sécurité. En effet, notre protocole n'est pas pour autant exempté de toute reproche et les inconvénients suivants peuvent être cités :

- Le protocole SEP ne protège pas contre l'analyse du traffic. Un intrus peut en analysant le premier échange SEP, déduire vers quel service le client souhaite se connecter ainsi que les paramètres de sécurité de chaque service.
- Le protocole ne définit pas un mécanisme de sécurisation vers la base de données des politiques de sécurité DBP.
- Le protocole n'intègre pas actuellement un mécanisme de découverte de services. Par ailleurs, nous proposons pour ce sujet d'utiliser les messages *SNGreq* et *SNGresp*. Ces deux messages peuvent être utilisés avec le protocole ALARM pour fournir un simple échange de découverte de services indépendamment du protocole ASNP.
- Le protocole se rapproche beaucoup du protocole SSL/TLS. Il possède les inconvénients de ce dernier tels que les attaques liées aux protocoles de transports utilisés.
- Notre vérification formelle est limitée à une spécification particulière du protocole SEP et ne vérifie pas le protocole lui-même tel qu'il sera implémenté.

8.2 Perspectives et travaux futurs

Les perspectives de nos travaux sont surtout basées sur le protocole SEP et sur le protocole SSL/TLS. Nous nous intéressons les axes suivants :

- *Intégration d'un système de contrôle d'accès au protocole SSL/TLS :*
  Actuellement, le protocole SSL/TLS n'offre pas un mécanisme de contrôle d'accès ou d'autorisation. Notre perspective consiste à intégrer un système de contrôle d'accès tels que le système 3A (Authentification, Autorisation, Audit) dans le protocole SSL/TLS. Ceci permet d'offrir une solution de sécurité indépendante des plates-formes matérielles et plus robuste aux violations de droit dans la mesure où elle autorise un accès restreint à certains systèmes ou à certaines applications spécifiques.

- *Utilisation de SEP dans les réseaux sans fils:*
  Un des problèmes spécifiques aux réseaux mobiles est la corrélation entre l'utilisateur nomade et le lieu où celle-ci a été transmise. SEP offre un service d'anonymat vis-à-vis de toute personne tierce non autorisée. Par conséquent, il possède un avantage majeur par rapport à d'autres solutions d'authentification pour les environnements mobiles tels que le protocole EAP-TLS.

- *Utilisation de SEP dans le domaine de diffusion multimédia :*
  Un point important qui donne une nouvelle dimension à SEP et sur lequel nous focaliserons nos travaux est la future utilisation de SEP dans des systèmes de diffusion multimédia pour le groupe [MMUSIC] à l'IETF. Les méthodes souples d'authentification proposées par SEP, son mécanisme de distribution de données basé sur les IAs intermédiaires, son utilisation du protocole UDP ainsi que son mécanisme de délégation des rôles permettent à notre protocole d'être la solution idéale pour une distribution large et sécurisée des données audio ou vidéo sur Internet.

- *Utilisation de SEP avec les CDNs :*
  L'apporche CDNs (Content Delivery Network) [Dele04] est une technique de partage des informations (surtout sur les sites Web) sur plusieurs passerelles afin d'alléger le besoins de bande passante des sites Internet à fort trafic et des fournisseurs d'accès. Le principe consiste à installer des serveurs chez ces derniers répliquant le contenu des sites.
  Pour plusieurs raisons citées dans cette thèse, cette technologie ne permet pas de dupliquer des sites de commerces électroniques basés sur SSL/TLS ou IPSec. Avec le mécanisme de délégation attribuée à la IA SEP, cette dernière pourrait jouer le rôle d'une passerelle CDN installée à proximité ou même éloignée des serveurs origines SEP. Cette proposition fournit des tunnels sécurisés et minimise l'utilisation de la bande passante vers le serveur origine SEP.

Bibliographie

[3DES] W. Tuchman. Hellman Presents No Shortcut Solutions To DES. IEEE Spectrum, v. 16, n. 7, July 1979, pp 40-41.

[Arkk02] J. Arkko, and R. Blom. The MAP Security Domain of Interpretation for ISAKMP. IETF Internet Draft, draft-arkko-map-doi-05.txt, February 2002

[Alle00] J. Allen, A. Christie, W. Fithen, J. Mackhugh, and J. Pickel. State of Practice of Intrusion Detection Technologies. Stoner Carnegie Mellon University, Networked Systems Survivability Program, Technical Report, CMU/SEI-99-TR-028, 2000.

[Apos99] G. Apostolopoulos, V. Peris, and D. Saha. Transport Layer Security: How much does it really cost?. In the Proceedings of the IEEE INFOCOM, 1999.

[Badr04] M. Badra, O. Sherkaoui, I. Hajjeh et A. Serhrouchni. Pre-Shared-Key key Exchange methods for TLS, IETF Internet Draft, draft-badra-tls-key-exchange-00.txt (work in progress), July 2004.

[Badr04+] M. Badra, A. Serhrouchni, P. Urien. TLS Express. IETF Internet Draft,draft-badra-tls-express-00.txt (work in progress), June 2004.

[Bert03] M. Bertrand, Redondance de sites et VPN SSL dans le cadre d'un réseau GPRS. Master's thesis, Département Informatique et réseaux, ENST Paris, Juillet 2003.

[Bern02] P.A. Frausto Bernal, C. Antoine. Controlling digital multisignature with attribute certificate. Annual Computer Security Applications Conference, Las Vegas, Nevada, USA, pp, 09-13 décembre 2002

[Bla02] B. Blanchet. From secrecy to authenticity in security protocols. In 9th International Static Analysis Symposium (SAS'02), volume 2477 of Lecture Notes in Computer Science, pages 242–259, Madrid, September 2002. Springer-Verlag.

[Bozg03] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. In Tools and Agorithms for the Construction and Analysis of Sytems (TACAS'03), 2003.

[Bozg02] L. Bozga, Y. Lakhnech, M. Périn. L'outil de vérification Hermes. Rapport technique EVA N.6, Mai 2002.

[Blan01] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In the 14th IEEE Computer Security Foundations Workshop (CSFW'01), Cape Breton, Novia Scotia, Canada, June 2001.

[MMUSIC] Multiparty Multimedia Session Control (MMUSIC) Home Page. http://www.ietf.org/html.charters/mmusic-charter.html.

[Canv03] Brice Canvel, Alain Hiltgen, Serge Vaudenay, Martin Vuagnoux. Password Interception in an SSL/TLS session. CRYPT'03, Lecture Notes in Computer Science, No.2729, pp. 583-599, Springer-Verlag, 2003.

[Canet01] R. Canetti, and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. Eurocrypt'2001. http://eprint.iacr.org/2001/040.

[Coud04] T. Piette-Coudol. Les contraintes imposes par le droit à la signature électronique. In the 3rd Conference on Security and Network Architectures (SAR'04), France, Juin 2004.

[Can02] R. Canetti, and H. Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. Cryptology ePrint Archive, Report 2002/120, 2003.

[Cort03] V. Cortier. Vérification automatique des protocoles cryptographiques. Thèse de doctorat, Ecole Normale Supérieure de Cachan, mars 2003.

[CST96] Sécurité des réseaux : Analyse et mise en œuvre. Gouvernement du Canada, service CST, janvier 1996.

[Cant01] A. Canteaut, F. Lévy-dit-Véhel. La cryptologie moderne. Paru dans la Revue Armemen, 73:76-83, Mars 2001.

[CA-96.21] CERT Advisory CA-96.21. TCP SYN flooding and IP spoofing, November 2000. http://www.cert.org/advisories/CA-1996-21.html.

[Denk98] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude, 1998.

[Diff76] W. Diffie, M.E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, vol IT-22, n 6, pp. 644-654, November1976.

[Dela03] S. Delaune. Vérification de protocoles de sécurité dans un modèle de l'intrus étendu, Mémoire de DEA. Laboratoire Spécification et Vérification. Ecole Normale Supérieure de Cachan. Paris, Septembre 2003.

[Dele04] C. Deleuze. La distribution de contenu dans l'Internet (CDN), cours magistral à l'ENST Paris, Janvier 2004. http://christophe.deleuze.free.fr

[DES] ANSI X3.106. American National Standard for Information Systems-Data Link Encryption. American National Standards Institute, 1983.

[DSS] NIST FIPS PUB 186. Digital Signature Standard. National Institute of Standards and Technology, U.S. Department of Commerce, May 18, 1994.

[Dbar02] D. Barrett, R. Silverman. SSH, le Shell sécurisé : la référence. N° ISBN: 2-84177-147-4, O'Reilly, Paris, 2002

[Deme04] J. Demerjian, A. Serhrouchni. EPMI : Une extension de l'infrastructure de gestion des privilèges. IEEE International Conference Sciences of Electronic, Technology of Information and Telecommunications SETIT'2004, ISBN 9973-41-902-2, pp.205, Sousse, Tunisie, 15-20 mars 2004.

[Domi01] Dominique Bolignano, Francesca Fiorenza, Florent Jacquemard, Daniel Le Métayer. EVA test base, Rapport Technique EVA No 4, novembre 2001.

[Eron04] P. Eronen, and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). IETF Internet Draft, draft-eronen-tls-psk-00.txt,August 2004.

[EVA] Projet RNTL-EVA. Réseau National des Technologies Logicielles, Explication et Vérification Automatique de protocoles cryptographiques. http://www-eva.imag.fr

[Ford94] W. Ford, M. Baum. Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption. ISBN 0-13-476342-4, 1994.

[FIPS1402] FIPS 140. Security Requirements for Cryptographic Modules, NIST. Publié en 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[FRZ02] Official Site of Fortezza. http://www.armadillo.huntsville.al.us.

[Ferg00] N. Ferguson, B. Schneier. A cryptographic evaluation of IPSec. Technical report, Counterpane Internet Security, 95128, USA, 2000.

[Goub00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In International Workshop on Formal Methods for Parallel Programming, Theory and Applications. volume 1800 of LNCS, 2000.

[Gutm04] P. Gutmann. Use of Shared Keys in the TLS Protocol. IETF Internet Draft, draft-ietf-tls-sharedkeys-02.txt, April 2004.

[GNUTLS] The GNUTLS project. http://www.gnu.org/software/gnutls

[Hajj03a] I. Hajjeh, A. Serhrouchni and F. Tastet. ISAKMP Handshake for SSL/TLS. In the proceedings the IEEE Global Communications conference (GLOBECOM'03). San Francisco (California, USA), Vol. 3, pp. 1481-1485, décembre 2003.

[Hajj03b] I. Hajjeh, A. Serhrouchni and F. Tastet. New Key Management Protocol for SSL/TLS. In the proceedings of the IEEE-IFIP Network Control and Engineering for QoS, Security and Mobility (NetCon'03). Muscat (Oman), Vol. 1, N. 19, pp. 251-262, octobre 2003.

[Hajj03c] I. Hajjeh, A. Serhrouchni and F. Tastet. Vers l'intégration de nouveaux services dans SSL/TLS. In the proceedings of the IEEE International Conference Sciences of Electronic, Technology of Information and Telecommunications (SETIT 2003). Sousse (Tunisie), Vol. 1, N. 106, pp. 52, mars 2003.

[Hajj03d] I. Hajjeh, A. Serhrouchni et F. Tastet. A new Perspective for ebusiness with SSL/TLS. In the proceedings of the Fifth International Conference on Advances in Infrastructure for eBusiness, eEducation, eScience, on the Internet (SSGRR2003w). L'aquila (Italie), Vol. 1, N. 84, pp. 56, janvier 2003.

[Hajj03e] I. Hajjeh, A. Serhrouchni et F. Tastet. Une nouvelle perspective pour SSL/TLS avec ISAKMP. 2ème Rencontre francophone sur Sécurité et Architecture Réseaux (SAR 2003). Nancy (France), pp. 55-63, juin 2003.

[Hajj04a] I. Hajjeh, A. Serhrouchni. Integrating a signature module in SSL/TLS. In the proceedings of the First ACM/IEEE International Conference on E-Business and Telecommunication Networks, ICETE'04 . Setúbal (Portugal), August 2004.

[Hajj04b] I. Hajjeh, A. Serhrouchni. Génération d'une preuve de non répudiation dans SSL/TLS. 3ème Rencontre francophone sur Sécurité et Architecture Réseaux (SAR 2004). La Londe, Cote d'Azur (France), Juin 2004

[Hall00] N. Hallqvist, A. Keromytis. Implementing Internet Key Exchange (IKE). In Proceedings of the Annual USENIX Technical Conference, June 2000.

[Han99] H. Handschuh. Cryptanalyse et Sécurité des algorithmes à Clé Secrète. Thèse de doctorat. Ecole Nationale Supérieure des Télécommunications, 1999.

[Huss05] M. Hussain, I.Hajjeh, H. Afific et D. Sereta: «Tri-party IKEv2 in Home Networks». Soumis à la conférence ICC'2005, 40th annual IEEE International Conference on Communications, Seoul (Korea), Mai 2005.

[Hoff02] P. Hoffman. Features of Proposed Successors to IKE. IETF Internet Draft, draft-ietf-ipsec-soi-features-01.txt, May 2002.

[GFD.17] M. Thompson, D. Olson, R. Cowles, S. Mullen, M. Helm. CA-based Trust Issues for Grid Authentication and Identity Delegation, Grid Forge Document No. 17.

[ibmVPN] Using IPsec to Construct Secure Virtual Private Networks. IBM Corporation. http://www-4.ibm.com/software/network/library/whitepapers/vpn

[IKEv2] Harkins, D., Kaufman, C., and Perlman, R. The Internet Key Exchange (IKE) Protocol; IETF Internet Draft draft-ietf-ipsec-ikev2-14.txt, May 2004.

[ISO-9594] ITU-T Recommendation X.509 | ISO/IEC 9594-8: Information Technology – Open Systems Interconnection – The Directory: Public-Key And Attribute Certificate Frameworks" ITU-T, 03/2000.

[ISO9798] ISO/IEC 9798-2: Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms, 1999.

[ICARE] Projet RNRT I-CARE. Infrastructure de Confiance sur des Architectures de Réseaux Internet & Mobiles, 2000. http://www.cert-i-care.org.

[Jaff00] Y. Jaffarain. IP Sécurité : de la théorie à la pratique. Mémoire de diplôme d'ingénieurs, ENST Paris, 2000.

[Jaco03] J. Jacobson. Trust negotiation in session-layer protocols. Master of Science, Department of Computer Science, Brigham Young University, July 2003.

[Jack01] K. Jackson, S. Tuecke, D. Engert. TLS Delegation Protocol. IETF Internet draft, draft-ietf-tls-delegation-01.txt, February 2001

[Rizc00] M. Rizcallah. Construire un annuaire d'entreprise avec LDAP. Editions Eyrolles, Paris, 2000

[Kivi04] T. Kivinen, T. Kivinen, B. Swander, and V. Volpe. Negotiation of NAT-Traversal in the IKE. IETF Internet Draft. February 2004.

[JFK] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis,A.D. Keromytis, and O. Reingold. Just Fast Keying (JFK). IETF Internet Draft (draft-ietf-ipsec-jfk-00.txt), May 2003

[Kraw01] H. Krawczyk. On JFK design and a comparison with SIGMA. Technion University, 2001. http://www.ee.technion.ac.il/~hugo/compare-jfk-sigma.txt

[Kraw96] H. Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. From IEEE Proceedings of the 1996 Symposium on Network and Distributed Systems Security.

[Kraw03] H. Krawczyk. The 'SIGn-and-Mac' Approach to Authenticated Diffie-Hellman and its Use in the IKE protocols. In the proceedings of the 23rd Annual International Cryptology Conference (CRYPTO'03), California, USA, 2003. IEEE Computer Society Press.

[Klima03] V. Klima, O. Pokorny and T. Rosa, Attacking RSA-based Sessions in SSL/TLS, http://eprint.iacr.org/2003/052/

[Kols02] O. Kolesnikov, B. Hatch. Construire un VPN sous IP avec Linux. Edition CampussPress, Paris, 2002

[Llor03] C. Llorens, L. Levier. Tableaux de bord de la sécurité réseau. Edition Eyrolles. ISBN : 2-212-11273-4, 2003

[Labo00] G. Labouret. IPSEC: Présentation technique. Hervé Schauer Consultants (HSC), Paris, France

[Low98] G. Lowe. Casper: A compiler for the analysis of security protocols. In the proceedings of the 10th Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press. Also in Journal of Computer Security, Volume 6, pages 53-84, 1998.

[Moda04] N. Modadugu, and E. Rescorla. The Design and Implementation of Datagram TLS. In the 11th Annual Network and Distributed System Security Symposium, San Diego, USA, February 2004.

[MR00] J. Millen and H. Rueß. Protocol-Independent secrecy. In the proceedings of the 21st Symposium on Research in Security and Privacy (RSP'00). IEEE Computer Society Press, 2000.

[MG-2] Guide de la gestion des risques d'atteinte à la sécurité des technologies de l'information (MG-2). Gouvernement du Canada, service CST, janvier 1996

[Möd03] Sebastian Mödersheim. OFMC: An On-the-Fly Model-Checker for Security Protocol Analysis. World Wide Web, http://www2.inf.ethz.ch/~moederss/research/, 2003

[Mill] J. Millen. Common authentication protocol specification language. http://www.csl.sri.com/millen/caspel.

[MG-15a] Infrastructure à clé publique du gouvernement du Canada. Livre blanc. Gouvernement du Canada, centre de la sécurité des télécommunications (CST), février 1998

[Medd04] A. Meddahi, K. Masmoudi, H. Afifi, A. M'Hamed, I. Hajjeh. Enabling Secure Third Party Control on Wireless Home Network. In the 4th IEEE Workshop on Applications and Services in Wireless Networks ASWN'2004. Boston (USA), August 2004.

[Mark97] T. Markham. Internet Security Protocol. Dr. Bobb's Journal, juin 1997.

[MD2] B. Kaliski. The MD2 Message Digest Algorithm. IETF RFC No. 1319, April 1992.

[MISC02] Le Magazine MISC (Multi-system and Internet Security CookBook), Edition, N. 2, Juin 2002.

[OpenSSL] The OpenSSL Projet.http://www.openssl.org.

[PFS] H. Krawczyk. Perfect Forward Secrecy, Internet Report. Available at: http://www.win.tue.nl/~henkvt/PerfForwSec (Krawczyk).pdf

[PKIX-WG] Groupe de travail Public-Key Infrastructure (X.509) (PKIX), IETF, Http://www.ietf.org/html.charters/pkix-charter.html

[Perl01] R. Perlman and C. Kaufman. Analysis of the IPSec Key Exchange Standard. In the proceedings of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2001.

[Resc03] E. Rescorla, N. Modadugu, Datagram Transport Layer Security, IETF Internet draft, (draft-rescorla-dtls-00.txt), November 2003.

[Resc02] Eric Rescorla. SSL and TLS: Designing and Building Secure System. ISBN: 0-201-61598-3.

[RFC1191] J. Mogul. Path MTU Discovery. IETF RFC, No. 1191, November 1990.

[RFC1832] R. Srinivansan. XDR: External Data Representation Standard, IETF RFC No. 1832, August 1995.

[RFC2326] H. Schulzrinne et al. Real Time Streaming Protocol (RTSP), IETF RFC No. 2326, April 1998.

[RFC2521] P. Karn and W. Simpson. Photuris: Session-Key Management Protocol. IETF RFC No. 2521, March 1999.

[RFC3275] Signature Syntax and Processing. W3C and IETF Sandard. IETF RFC No. 3275, March 2002.

[RFC2828] R. Shirey. Internet Security Glossary. IETF RFC No. 2828, May 2000

[RFC2818] E. Rescorla. HTTP Over TLS. IETF RFC No. 2818, May 2000

[RFC2315] B. Kalishi. PKCS#7: Cryptographic Message Syntax Version 1.5. IETF RFC No. 2315, March 1998

[RFC3369] R. Housley. Cryptographic Message Syntax (CMS). IETF RFC No. 3369, August 2002

[RFC2560] M.Myers et.al. Internet X.509 Public Key Infrastructure: Online Certificate Status Protocol – OCSP. IETF RFC No.2560, June 1999.

[RFC2311] S. Dusse, P. Hoffman et al. S/MIME Version 2 Message Specification, IETF RFC No. 2311, March 1998.

[RFC3232] J. Reynolds. Assigned Number: RFC 1700 is Replaced by an On-line Database. IETF RFC No. 3232, January 2002.

[RFC2246] T. Dierks, C.Aallen. The TLS Protocol Version 1.0. IETF RFC No. 2246, January 1999.

[RFC3546] S. Blake-Wilson, M. Nystrom et al. Transport Layer Security (TLS) Extensions. IETF RFC No. 3546, June 2003

[RFC3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, and M. H. E. Schooler. SIP: Session Initiation Protocol. IETF RFC No. 3261, June 2002.

[RFC3550] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF RFC No. 3550, July 2003.

[RFC3435] F. Andreasen and B. Foster. Media Gateway Control Protocol (MGCP). IETF RFC No. 3435, January 2003.

[RFC2104] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication, IETF RFC No. 2104, February 1997.

[RFC2817] R. Khare and S. Lawrence. Upgrading to TLS Within HTTP/1.1, IETF RFC No. 2817, May 2000.

[RFC1034] P. Mockapetris. Domain Names - Concepts and Facilities. STD 13, IETF RFC No. 1034, November 1987.

[RFC1035] P. Mockapetris. Domain Names - Implementation and Specification. STD 13, IETF RFC No. 1035, November 1987.

[RFC2252] M. Wahl, A. Coulbeck, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions. IETF RFC No. 2252, December 1997.

[RFC2253] S. Kille, M Wahl and T. Howes. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. IETF RFC No. 2253, December 1997.

[RFC2460] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. IETF RFC No. 2460, December 1998.

[RFC959] J. Postel and J. Reynolds. File Transfer Protocol. STD 9, IETF RFC No. 959, October 1985.

[RFC791] J. Postel. Internet Protocol - DARPA Internet Program Protocol Specification. IETF RFC No. 791, USC/Information Sciences Institute, September 1981.

[RFC761] J. Postel. Transmission Control Protocol. IETF RFC No. 761, USC/Information Sciences Institute, January 1980.

[RFC768] J. Postel. User Datagram Protocol. IETF RFC No. 768, August 1980

[RFC2522] P. Karn and W. Simpson. Photuris: Session-Key Management Protocol. IETF RFC No. 2522, March 1999.

[RFC1321] R. Rivest, The MD5 Message Digest Algorithm, IETF RFC No. 1321, April 1992.

[RFC854] J. Postel and J. Reynolds. Telnet Protocol Specifications, STD 8, IETF RFC No. 854, May 1993.

[RFC2407] D. Piper. The Internet IP Security Domain of Interpretation for ISAKMP. IETF RFC No. 2407, November 1998.

[RFC2402] S. Kent and R. Atkinson. IP Authentication Header (AH). IETF RFC No. 2402, November 1998.

[RFC2412] H. Orman. The OAKLEY Key Determination Protocol. IETF RFC No. 2412, November 1998.

[RFC2406] S. Kent, and R. Atkinson. IP Encapsulating Security Payload (ESP). IETF RFC No. 2406, November 1998.

[RFC1510] J. Khol and C. Neuman. The Kerberos network authentication service (v.5). IETF RFC No. 1510, September 1993.

[RFC2401] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. IETF RFC No. 2401, November 1998

[RFC2409] D. Harkins et. al. The Internet Key Exchange (IKE). IETF RFC No. 2409, November 1997.

[RFC3547] M. Baugher, T. Hardjono, H. Harney, and B. Weis. The Group Domain of Interpretation. IETF RFC No. 3547, July 2003.

[RFC791] DARPA Internet program protocol specification, IETF RFC No. 791, September 1981.

[RFC2408] D. maughan. Internet Security Association and Key Management Protocol (ISAKMP). IETF RFC No. 2408, November 1998.

[RFC2663] Srisuresh, P. and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. IETF RFC No. 2663, August 1999.

[RFC1928] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones. SOCKS Protocol Version 5, IETF RFC No. 1928, March 1996.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk. Hypertext Transfer Protocol -- HTTP/1.0. IETF RFC No. 1945, May 1996.

[RFC2719] B. Aboba, D. Simon. PPP EAP TLS Authentication Protocol. IETF RFC No. 2716, October 1999

[Rive78] R.L. Rivest, A. Shamir and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, vol. 21, n 2, p. 120-126, February 1978.

[RSA03] Factorization of RSA-576. RSA Security, url:http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa576.html, 2003.

[Securify-TR7] Rapport Technique EVA No 7, L'outil de vérification SECURIFY, Véronique Cortier, mai 2002. http://www-eva.imag.fr/bib/EVA-TR7.pdf

[Securify-TR13] Rapport Technique EVA No 13, A guide for SECURIFY, Véronique Cortier, décembre 2003. http://www-eva.imag.fr/bib/EVA-TR13.pdf

[SAML] Security Assertions Markup Language, Verisign, February 2001, http://www.oasis-open.org/committees/security/docs

[SonIKE] Son-of-IKE Performance. IPSec IETF Mailing List. http://www.vpnc.org/ietf-ipsec/01.ipsec/msg02223.html

[SSL3] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. November 18, 1996. http://home.netscape.com/eng/ssl3/ssl-toc.html.

[SSH-Arch] T.Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH protocol architecture. IETF Internet Draft, draft-ietf-secsh-architecture-13.txt, September 2002.

[SSH-Auth] T.Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH authentication protocol. IETF Internet Draft, draft-ietf-secsh-userauth-18.txt, September 2002

[SSH-Trans] T.Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH transport layer protocol. IETF Internet Draft, draft-ietf-secsh-transport-17.txt, October 2003

[SSH-Conn] T.Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH connection protocol. IETF Internet Draft, draft-ietf-secsh-connect-18.txt, October 2003.

[Snell02] J. Snell, D. Tidwell and P. Kulchenko. Programming Web Services with SOAP. O'Reilly Editions, January 2002.

[Shac02] H. Shacham and D. Boneh. Fast-Track Session Establishment for TLS. In the Proceedings of Internet Society's, 2002.

[Shac01] H. Shacham and D. Boneh. Improving SSL Handshake Performance via Batching. In D. Naccache, ed., Proceedings of RSA, 2001.

[Sierr02] J.M Sierra, J.C Hernandez and A. Ribagorda. Narayana Jayaram Migration of Internet Security Protocols to the IPSEC framework. In the Proceedings of the 36th IEEE Carnahan Conference on Security Technology, IEEE Press, EEUU, 2002.

[SIGMA] H. Krawczyk. The IKE-SIGMA Protocol. Available at: http://www.ee.technion.ac.il/~hugo/draft-krawczyk-ipsec-ike-sigma-00.txt

[Stunnel] The STUNNEL Projet; http://www.stunnel.org

[SHA] NIST FIPS PUB 180-1. Secure Hash Standard. Nationa Institute of Standards and Technology, U.S. Department of Commerce, Work in Progress, May 31, 1994.

[Schn95] Bruce Schneier. Applied Cryptography : Protocols, Algorithms, and Source Code in C, 2nd edition. Published by John Wiley & Sons, 1995.

[Sprin02] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In Proceedings of ACM SIGCOMM'2002, Pittsburgh, PA, August 2002.

[Strei02] W.W. Streilein, R.K. Cunningham, and S.E. Webster. Improved Detection of Low-Profile Probe and Novel Denial-of Service Attacks. In the proceedings of the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, Baltimore, MD, 2002.

[Shc02] H. Shcham, D. Boneh, and E. Rescorla. Client Side Caching for TLS. In the Proceedings of Internet Society's 2002 Symosium on Network and Distributed System Security (NDSS), Page 195-202, 2002.

[Stal02] W. Stallings. Sécurité des réseaux, Applications et standard, Editions Vuibert Informatique, Paris, 2002. ISBN 2-7117-8653-6

[Samf96] D. Samfat, Architecture de sécurité pour réseaux mobiles, Thèse de Doctorat, ENST Paris, 1996.

[Sher92] M. Shérif, A. Serhrouchni. La monnaie électronique, Editions Eyrolles, Paris, 2000. ISBN 2-212-09082-X.

[Tayl04] D. Taylor, T. Wu and T. Perrin. Using SRP for TLS Authentication, IETF Internet Draft, draft-ietf-tls-srp-06, Jan 2004.

[Tsch03] H. Tschofenig, H. Schulzrinne. RSVP Domain of Interpretation for ISAKMP, IETF Internet Draft, draft-tschofenig-rsvp-doi-01.txt, October 2003.

[Verd03] T. Verdet. Interfaçage de OpenSSH avec une PKI. Master's thesis, Département Informatique et réseaux, ENST Paris, Octobre 2003

[W3C-98a] The Extensible Mark-up Language (XML) the base specifications XML 1.0. W3C Rec 02/1998.

[Wrig94] G. R. Wright and W. R. Stevens. TCP/IP Illustrated, Volume 2. Addison-Wesley Publishing Company, 1994.

[WAP] WAP Forum, Wireless Transport Layer Security Specification Version 1.1, 11.2.1999

[Wadj02] Fouzia Wadjinny. Contexte d'utilisation des certificats d'attributs, Mémoire de DEA, Département Informatiques et Réseaux, ENST Paris, Septembre 2002.

[Wich99] M. Wichert, D. Ingham et al. Non-repudiation Evidence Generation for CORBA using XML. In the proceedings of the ACSACs conference, Phoenix, Arizona, December 1999.

[Wag96] D. Wagner, B. Schneier. Analyse of the SSL 3.0 protocol. University of California, Couterspane Systems, 1996.

[W3Cdtd] Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1. Retrieved from: http://www.w3.org/XML/1998/06/xmlspec-report.

[X.208] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1998

[Xiao01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. 10th USENIX Security Symposium, 2001.

[Ylon95] T. Ylonen. The SSH (Secure Shell) Remote Login Protocol, Helsinki University of Technology, November 1995.

Annexe A : Liste des acronymes

AA	Attribute Authority
AC	Attribute Certificate
ACL	Access Control List
AH	Authentication Header
AES	Advanced Encryption Standard
ASN. 1	Abstract Syntax Notation One
ASNP	Authentication and Service Negotiation Protocol
DES	Data Encryption Standard
DH	Diffie and Hellman Algorithm
DOS	Denial of Service
Draft	Projet de RFC Internet.
DTD	Document Type Declaration
DSA	Digital Signature Algorithm
ACL	Access control list
API	Application Program Interface
CA	Certificate authority
CRL	Certificate Revocation List
ESP	Encapsulating Security Payload
FTP	File transfer protocol
FW	FireWall
HMAC	Hash-Based Message Authentication Code
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IA	Intermediate Authority
IANA	Internet Assigned Numbers Authority
I-CARE	Infrastructure de Confiance sur des Architectures de RésEaux Internet & Mobile
ICP	Infrastructure à Clé Publique
IEEE	Institute of Electrical and Electronics Engineers
IETF	International Engineering Task Force
IKE	Internet Key Exchange
IP	Internet protocol
IPSec	Internet protocol security
ISAKMP	Internet Security Association and Key Management Protocol
ISO	International Organisation for Standardization
ITU-T	International Telecommunication Union - Telecommunication
JFK	Just Fast Keying
LDAP	Lightweight Directory Access Protocol
MIME	Multipurpose Internet Mail Extensions
MAC	Message Authentication Code
NIST	National Institue of standards and Technology
OCSP	On-line Certificate Status Protocol
OSI	Open Systems Interconnection
PEM	Privacy Enhanced Mail
PC	Personal Computer

PDA	Personal Data Assistant
PMI	Privilege Management Infrastructure
PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastructure
PRF	Pseudo Random Function
RSA	Rivest Shamir Adleman
RFC	Request for comment
RNRT	Réseau National de la Recherche en Télécommunication
SHA	Secure Hash Algorithm
S/MIME	Secure Multi-purpose Internet Mail Extension
SDL	Specification and Description Language
SDK	Software Development Kit
SEP	Secure and Extensible Protocol
SIGMA	Sign and Mac Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign On
SKIP	Simple Key for IP
TBD	To Be Defined
TELNET	TErminaL NETwork protocol
TLS	Transport Layer Security
TCP	Transport Control protocol
UDP	Universal Datagramme Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	WWW Consortium
WWW	World Wide Web
XML	eXtensible Markup Language
XDR	EXternal Data Representation standard
X.500	Norme proposé par l'ITU - T, ISO (annuaires).
X.509	Norme du certificat numérique propose par l'ITU-T, ISO.

Annexe B : Spécification en LEVA de quelques solutions de sécurité

L'objectif de cette annexe est de présenter les spécifications des protocoles mentionnés dans le chapitre trois. Nous donnons les spécifications en *eva* des trois protocoles, SSL/TLS, SSH et IKE. La vérification en Hermes des trois protocoles montre qu'ils sont cryptographiquement sécurisés

Description du protocole SSL/TLS en LEVA

La figure suivante montre une spécification en LEVA du protocole SSL/TLS avec authentification du client en se basant sur le chiffrement asymétrique avec la clé publique du serveur pour la génération d'une clé partagée.

```
01  SSL_specification_in_LEVA
02
03  alg : asym_algo
04  everybody knows alg
05
06    A, B,CA: principal
07    basetype key
08    Ka: key
09    Na,Nb,P: number
10    keypair^alg SK, PK (principal)
11
12
13    alias certB = { CA, B, PK(B) }_SK(CA)^alg
14    alias certA = { CA, A, PK(A) }_SK(CA)^alg
15
16    A knows A, B, SK(A), PK(A), PK(B), PK(CA), certA, certB
17    B knows B, CA, SK(B), PK(B), PK(CA),certB
18
19     {
20      1. A -> B : Na
21      2. B -> A : Nb,B,certB
22      4. A -> B : {Ka}_(PK (B))^alg
23      3. A -> B : certA, A, {Nb,Na,B,A}_(SK (A))^alg
24      6. A -> B : {Nb,Na,B,A}_Ka
25      7. B -> A : {Nb,Na,B,A}_Ka
26     }
27
28     s. session* {Ka,Na,Nb} A=A, B=B
29
30     assume secret (SK(B)@s.B),
31              secret (SK(A)@s.A),
32           secret(Ka@s.A),
33           secret(Ka@s.B)
34
35  ---------------------------------------------------------------------------------------------------
36  Secrets: SK(h); Ka
37  GoodPatterns: {xs}_PK(h); {xs}_Ka
    BadPatterns: vide
```

La figure suivante présente le Handshake de SSL/TLS sans authentification du client se basant sur le chiffrement asymétrique avec la clé publique du serveur pour la génération d'une clé partagée :

```
01                                  SSL_specification_in_LEVA
02
03  alg : asym_algo
04  everybody knows alg
05
06    A, B,CA: principal
07    basetype key
08    Ka: key
09    Na,Nb,P: number
10    keypair^alg SK, PK (principal)
11
12
13    alias certB = { CA, B, PK(B) }_SK(CA)^alg
14
15
16    A knows A, B, PK(B), PK(CA), certB
17    B knows B, CA, SK(B), PK(B), PK(CA),certB
18
19     {
20      1. A -> B : Na
21      2. B -> A : Nb,B,certB, {Nb,Na,B,PK(B)}_(SK (B))^alg
22      3. A -> B : {Ka}_(PK (B))^alg
23      6. A -> B : {Nb,Na,B,A}_Ka
24      7. B -> A : {Nb,Na,B,A}_Ka
25     }
26
27     s. session* {Ka,Na,Nb} A=A, B=B
28
29     assume secret (SK(B)@s.B),
30           secret(Ka@s.A),
31           secret(Ka@s.B)
32  ---------------------------------------------------------------------------------------------------------
33
34  Secrets:  SK(h); Ka
35  GoodPatterns: {xs}_PK(h); {xs}_Ka
36  BadPatterns: vide
```

Description du protocole SSH en LEVA

Ci-dessous, une spécification en LEVA du protocole SSH avec authentification du serveur par une clé privée et se basant sur un échange DH pour la génération d'une clé partagée.

```
01                                  SSH_specification_in_LEVA
02
03  alg : asym_algo
04  everybody knows alg
05
06    A, B: principal
07    basetype key
08    keypair^alg SK, PK (principal)
09
```

```
10    Na,Nb,P: number          // nombre aléatoire
11
12    S1: number               // demande d\'un service SSH
13    Pa, Pb: number  // clé publique DH éphémère
14    G: number                // groupe DH négocié
15    Kas: key                 // clé générée par l\'échange DH
16
17    A knows A, B, PK(B), Kas
18    B knows B, SK(B), PK(B), Kas
19
20     {
21      1. A -> B : Na
22      2. B -> A : Nb
23      3. A -> B : Pa, G
24      4. B -> A : Pb, G
25      5. B -> A : B, PK(B), {Na, Nb, Pa, Pb, G}_(SK(B))^alg
26      6. A -> B : {S1}_Kas
27      7. B -> A : {S1}_Kas
28     }
29
30    s. session* {Kas,Na,Nb,Pa,Pb,G} A=A, B=B
31
32    assume secret (SK(B)@s.B),
33          secret(Kas@s.A),
34          secret(Kas@s.B)
35  -----
36  Secrets:
37  SK(h);
38  Kas
39
40  GoodPatterns:
41  {xs}_PK(h);
42  {xs}_Kas
```

Description du protocole IKEv1 en LEVA

Voici une spécification en LEVA du protocole IKEv1 (Echange de protection d'identité avec authentification forte par des certificats X.509 des deux entités).

```
01                        IKEv1_Identity_Protection_Signature
02
03  alg : asym_algo
04  everybody knows alg
05
06    A, B,CA: principal
07    basetype key
08    keypair^alg SK, PK (principal)
09    SAi,SAr,Ci,Cr,Ni,Nr: number
10
11    //Ks(number, number, number) : number
12    p, g, Xa, Xb: number  // valeur publique DH éphémère
13    Ks: key             // clé dérivée des valeurs DH et des autres paramètres.
14
15    alias certB = { CA, B, PK(B) }_SK(CA)^alg
16    alias certA = { CA, A, PK(A) }_SK(CA)^alg
17
```

```
18   A knows A, SK(A), PK(A), PK(CA), certA, Ks
19   B knows B, CA, SK(B), PK(B), PK(CA),certB, Ks
20
21    {
22     1. A -> B : Ci,SAi
23     2. B -> A : Cr,SAr
24     3. A -> B : Ci, Cr, p,g,Xa, Ni
25     4. B -> A : Ci, Cr,p,g,Xb, Nr
26     5. A -> B : Ci,Cr, {{A,certA,Ni,Nr,p,g,Xa,Xb}_(SK(A))^alg}_Ks
27     6. B -> A : Ci,Cr, {{B,certB,Nr,Ni,p,g,Xa,Xb}_(SK(B))^alg}_Ks
28    }
29
30    s. session* {Ci,Cr,Ni,Nr,p,g,Xa,Xb,Ks} A=A, B=B
31
32    assume secret (SK(B)@s.B),
33          secret (SK(A)@s.A),
34          secret(Ks@s.A),
35          secret(Ks@s.B)
36   ------------------------------
37   Secrets: SK(h); Ks
38   GoodPatterns: {xs}_PK(h); {xs}_Ks
39   BadPatterns: vide
```

Annexe C : Spécifications détaillées des différents services SEP

Syntaxes XDR

Pour décrire le protocole SEP, nous utilisons les syntaxes définies dans le RFC 2246 du standard TLS. La syntaxe utilisée est très légère et ressemble au langage de programmation C et à la syntaxe de XDR [RFC1832]. Le but de ce langage est de rendre la documentation du protocole SEP plus claire et plus formelle.

Présentation des constantes SEP

1. Les paramètres de sécurité

```
enum {server, client, proxy} ConnectionEnd;
enum {des3, aes} BulkCipherAlgorithm;
enum {block} CipherType;
enum {md5, sha}MACAlgorithm;
struct {
        ConnectionEnd entity;
        BulkCipherAlgorithm bulk_cipher_algorithm;
        CipherType cipher_type;
        unit8 keysize;
        unit8  key_material_length;
        MACAlgorithm mac_algorithm;
        unit8 hash_size;
        opaque master_secret[48];
        opaque client_random[32];
        opaque server_random[32];
        opaque mediator_random[32];

} SecurityParameters;

enum {rsa, dsa} SignatureAlgorithm;
   select (SignatureAlgorithm)
   {   case rsa:
         digitally-signed struct {

            opaque md5_hash[16];
            opaque sha_hash[20];
         };
      case dsa:
         digitally-signed struct {
            opaque sha_hash[20];
         };
   } Signature;
```

2. Le protocole TRANSFER

Ce qui suit est la syntaxe complète de la partie transfert.

```
struct {
          unit8      major,minor;
        }ProtocolVersion;
```

```
enum {
          asnp(0), alarm(1), http(2), ftp(3), rtsp(4),(255)
          }ProtocolType;

enum {tcp(0), udp(1), (65535);
        } TransferProtocol;

struct {
          select (TransferProtocol) {
             case tcp :
                  ProtocolType        protocol;
                  SessionID           session;
                  unit16              length;
                  opaque              fragment [SEPPlaintext.length];
            case udp :
                  ProtocolType        protocol;
                  SessionID           session;
                  unit48              sequence_number;
                  unit16              length;
                  opaque              fragment[SEPPlaintext.length];
             };
}SEPPlainText;


struct {
          select (TransferProtocol) {
             case tcp :
                  ProtocolType        protocol;
                   SessionID           session;
                  unit16               length;
                 GenericBlockCipher  Block-cipher;
            case udp :
                  ProtocolType         protocol;
                   SessionID           session;
                  Uint48               sequence_number;
                  unit16               length;
                 GenericBlockCipher  Block-cipher;
          }fragment;
}SEPCipherText;

   struct {
          select (TransferProtocol) {
             case tcp :
                  ProtocolType        protocol;
                  SessionID           session;
                  opaque              fragment[SEPCompressed.length];
             case udp:
                  ProtocolType        protocol;
                  SessionID           session;
                  Uint48              sequence_number;
                  opaque              fragment[SEPCompressed.length];
             };
   } SEPCompressed;


struct {
           opaque IV[CipherSpec.hash_size];
```

```
            opaque content [SEPPlainText.length];
            opaque MAC[CipherSpec.hash_size];
            uint8 padding[GenericBlockCipher.padding_length];
            unit8 padding_length;
}GenericBlockCipher ;

struct {
          opaque GSID[2];
         opaque LSID[2];
          } sessionID ;
```

3. Le protocole ALARM

```
enum { warning(1), fatal(2), (255) } AlarmLevel;
       enum {
          Service not supported(0),
         Unknown Server(1),
         Negociation failure(2),
         Invalid Version(3),
         Record_overflow(4),
         Decompression_failure(5),
         Insufficient_service_parameter(6),
         Unexpected KE Syntax(7),
         Invalid Cert Encoding(8),
        Expired Certificate(9),
        Revoked Certificate(10),
        Bad _Certificate(11),
        Unknown Certificate Authority(12),
        Invalid Cert Authority(13),
        Authentication Failed(14),
        Bad_Mac(15),
        Invalid Signature(16),
        Invalid Key Enctyption(17),
        Invalid Authentication(18),
        Insuffisient_security(19),
       Expired Session(20),
       Illegal_parameter(21),
       Close_notify(22),
       Session_life_time_alert(23),
           (255)
       } AlarmDescription;
    struct {
       SessionID session;
       AlarmLevel level;
       AlarmDescription description;
    } Alarm;
```

4. Le protocole ASNP

```
enum {
       ClientAttributeNegociation(0), ServerAttributeNegociation (1), MediatorAttributeNegociation (2),
Key_Exchange(3) , Service_Negociation_Request(4), Service_Negociation_Response (5),  Authentication(6),
Authentication _Succeed(7), Negociation_Succeed(8), (255)
    } ContenType;

ProtocolVersion version = {1,0} ;
```

```
enum {
  client(0), server(0), mediator(2)
      } ConnectionEnd;
   struct {
      ContenType content_type;
      uint16 length;
      select (ContentType) {
         case ClientAttributeNegociation:       CAN;
         case ServerAttributeNegociation:       SAN;
         case MediatorAttributeNegociation:   MAN;
         case Key_Exchange:        KE;
         case Service_Negociation_Request: SNGreq;
         case Service_Negociation_Response: SNGresp;
         case Authentication:   AUTH;
         case Authentication _Succeed:  AUTHsucc;
         case Negociation_Succeed: NGsucc;
      } body;
   } ASNP;

   struct {
      uint32 gmt_unix_time;
      opaque random_bytes[28];
   } Random;

/*----------------------Begin CAN ----------------------------------------------*/
   struct {
      ProtocolVersion client_version;
      Random random;
   } CAN;
/*-------------------------End CAN ----------------------------------------------*/

/*----------------------Begin SAN ----------------------------------------------*/
   struct {
      ProtocolVersion server_version;
      Random random;
   } SAN;
/*-------------------------End SAN ----------------------------------------------*/

/*----------------------Begin MAN ----------------------------------------------*/
   struct {
      ProtocolVersion mediator_version;
      Random random;
   } MAN;
/*-------------------------End MAN ----------------------------------------------*/

/*----------------------------Begin KE----------------------------------------------*/

   enum { rsa, diffie_hellman, aes } KeyExchangeAlgorithm;
   struct {
       select (ConnectionEnd) {
       case client : EncryptedMasterSecret;
      case mediator: EncryptedMasterSecret;
   } RSAParams;

   struct {
       select (ConnectionEnd) {
       case client :
```

```
            opaque DH_Y<1..2^16-1>;
        case server :
            opaque DH_p<1..2^16-1>;
            opaque DH_g<1..2^16-1>;
            opaque DH_Y<1..2^16-1>;
        case mediator :
            opaque DH_p<1..2^16-1>;
            opaque DH_g<1..2^16-1>;
            opaque DH_Y<1..2^16-1>;
        };
  } DHParams;


  struct {
     select (KeyExchangeAlgorithm) {
        case diffie_hellman:
           ServerDHParams params;
           Signature signed_params;
        case rsa:
           ServerRSAParams params;
           Signature signed_params;
        case aes:
           opaque iv[16];
           opaque encrypted_key<0.. 2^16-1> ;
     };
  } KeyExchange;
 enum { rsa, dsa } SignatureAlgorithm;

  select (SignatureAlgorithm)
  {   case anonymous: struct { };
     case rsa:
        digitally-signed struct {
           opaque md5_hash[16];
           opaque sha_hash[20];
        };
     case dsa:
        digitally-signed struct {
           opaque sha_hash[20];
        };
  } Signature;

  struct {
    select (ConnectionEnd) {
       case client:
          ProtocolVersion client_version;
          opaque random[46];
       case mediator:
          ProtocolVersion mediator_version;
          opaque random[46];
    };
  } PreMasterSecret;

  struct {
     PreMasterSecret pre_master_secret;
  } EncryptedPreMasterSecret;

/*----------------------------- End KE-----------------------------------------*/
```

```
/*-----------------------------Begin Service_Negociation_Request -----------*/

     struct {
        ServiceType service_type;
        Falg flag ;
        opaque content_data<0..2^16-1>;
     } SNGreq;

enum {
         server_application(0), certificate_request(1),  cipher_list(2), authentication_request(3),
         compression_list(4), truncated_hmac(5), fragment_length (6), session_life_time(7), non_repudiation(8),
         delete_session(9), (255)
         } ServiceType;

enum {false, true} Flag;

enum {
         TransferProtocol    protocol;
         ProtocolType        type ;
         Servername          server ;
          ParamNumber        param ;
        }server_application ;

opaque server_name<2..2^16-1>;
opaque ParamNumber[2];

      enum {
            certificate_sign(1), certificate_attribute(2), icare_certificate_attribute(3), url_certificate_sign(4),
              ocsp_status(5), (255)
      } CertificateType;

      opaque DistinguishedName<1..2^16-1>;

      struct {
         CertificateType certificate_types<1..2^8-1>;
         DistinguishedName certificate_authorities<3..2^16-1>;
      } Certificate_request;

   uint8 CipherSuite[2];

   Cipher_list SEP_RSA_WITH_NULL_MD5                = { 0x00,0x01 };
   Cipher_list SEP_RSA_WITH_NULL_SHA                = { 0x00,0x02 };
   Cipher_list SEP_RSA_WITH_IDEA_CBC_SHA            = { 0x00,0x07 };
   Cipher_list SEP_RSA_WITH_DES_CBC_SHA             = { 0x00,0x09 };
   Cipher_list SEP_RSA_WITH_3DES_EDE_CBC_SHA  = { 0x00,0x0A };


      enum {
           certificate(0), certificate _sign(1), certificate_attribute(2), icare_certificate_attribute(3),
           url_certificate_sign(4), psk_key(5), id_password(6), (255)
      } AuthenticationType;

   struct {
      ASN.1Cert certificate_list<1..2^24-1>;
   } Certificate;

   struct {
      Signature signature;
```

```
   } CertificateVerify;

      opaque DistinguishedName<1..2^16-1>;

      struct {
         Select (AuthenticationType) {
            Case certificate:
               DistinguishedName certificate_authorities<3..2^16-1>;
            Case certificate_sign:
               DistinguishedName certificate_authorities<3..2^16-1>;
            Case certificate_attribute:
               DistinguishedName certificate_authorities<3..2^16-1>;
            Case  icare_certificate_attribute:
               DistinguishedName certificate_authorities<3..2^16-1>;
             Case  url_certificate_sign:
               DistinguishedName certificate_authorities<3..2^16-1>;
             Case  psk:  psk;
             Case  id_password: id_password;
         };
      } authentication_request;

struct {
          compression comp;
         }compression_list ;

   enum { null(0), (255) } compression;

enum{80(1), 96(1),(255)}truncated_hmac;

enum{ 2^9(1), 2^10(2), 2^11(3), 2^12(4), (255) } fragment_length;

struct {
           select(unit) {
              Seconds second;
              KiloBytes Kbytes;
         } session_life_time;

opaque seconds<1..32>;
opaque KiloBytes<1..32>;


enum {
           pkcs7_1.5(0), smime (2), xmldsig(255);
        }ContentFormat;

struct {
          ContentFormat             content_format;
          SigMethod                 sig_meth;
          Boolean                   bool;
          Signature_type            sign_type<1..2^16-1>;
          third_party               url_third;
        } signature_request;

enum {
              x509cert(0), x509cert_url(1), (255);
        } SigMethod;

enum {
```

```
          false(0), true(1);
        } Boolean;

opaque third<1..22^16-1> ;

Signature_type  Non_repudiation_with_proof_origine = { 0x00,0x01 };
Signature_type  Non_repudiation_with_proof_receipt = { 0x00,0x02 };
Signature_type  Non_repudiation_with_double_proof = { 0x00,0x03 };


struct {
           SessionID session;
         }delete_session;


/*------------------------------End  Service_Negociation_Request ---------*/

/*------------------------------Begin Service_Negociation_Response -------*/
     struct {
        ContentType content_type;
        Falg flag ;
        opaque content_data<0..2^16-1>;
     } SNGresp;

/*------------------------------End Service_Negociation_Response ---------*/

/*------------------------------Begin Authentication ---------------------------*/
     struct {
        AuthenticationType authentication_type;
        opaque content_data<0..2^16-1>;
     } SNGresp;

   opaque ASN.1Cert<2^24-1>;

   struct {
      ASN.1Cert certificate_list<1..2^24-1>;
   } Certificate;

   struct {
      ASN.1Cert certificate_list<1..2^24-1>;
      CertificateVerify verify;
   } Certificate_sign;

   struct {
      Signature signature;
   } CertificateVerify;

     enum {
        individual_certs(0), pkipath(1), (255)
     } CertChainType;

     enum {
        false(0), true(1)
     } Boolean;

     struct {
        CertChainType type;
        URLAndOptionalHash url_and_hash_list<1..2^16-1>;
```

```
    } url_certificate_sign;

    struct {
       opaque url<1..2^16-1>;
       Boolean hash_present;
       select (hash_present) {
          case false: struct {};
          case true: SHA1Hash;
       } hash;
    } URLAndOptionalHash;

    opaque SHA1Hash[20];

   opaque ASN.1Cert<2^24-1>;

      struct {
         opaque ASN.1Acert<<2^24-1>;
      } certificate_attribute

      struct {
              opaque XML_AC<2^24-1>;
      } icare_certificate_attribute

struct {
    opaque identity<1.. 2^16-1>;
    opaque shared_key<1.. 2^16-1>;
} psk;

struct {
    opaque identity<1.. 2^16-1>;
    opaque password<1.. 2^16-1>;
} id_password;

    struct {
       CertificateStatusType status_type;
       select (status_type) {
          case ocsp: OCSPResponse;
       } response;
    } ocsp_status;

    opaque OCSPResponse<1..2^24-1>;

/*-----------------------------End Authentication ----------------------------------------------------*/

/*-----------------------------Begin Authentication Succeed-------------------------------------------*/

/*-----------------------------End Authentication Succeed ---------------------------------------------*/
enum {accept(0), reject(1)} succeed;

struct {
            succeed auth_succeed;
            server_application  server_app;
       }Authentication_Succeed;

/*-----------------------------Begin Negociation Succeed----------------------------------------------*/
     struct {
        opaque verify_data[12];
     } NegocitionSucceed;
```

```
verify_data
 /**   PRF(master_secret, negocition_succeed_label, MD5(handshake_messages) +
   SHA-1(handshake_messages)) [0..11]; // negocition_succeed_label witll take the value
“server_ng_succeed” if this message is sent by the server and “mediator_ng_succeed” if the message is send
from the mediator. */
/*-----------------------------End Negociation Succeed ------------------------------------------*/
```

Représentation en XML/DTD de la base de donnée DBP (DataBase Policy)

```
<database name="politique_sep.mdb">
<table name="Alarm" />

<table name="Authentication" />
<table name="Connection_Trace" />
<table name="Mediator" />
<table name="Network_Filter" />
<table name="Security_Parameters" />
<table name="Server" />
<table name="Service" />
<table name="Service_Alarm" />
<table name="Service_Authentication" />
<table name="Service_Parameters" />
</database>


<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE DTDSchema [
<!ELEMENT database ((Alarm, Authentication, Connection_Trace, Mediator, Network_Filter, Security_Parameters, Server, Service,
Service_Alarm, Service_Authentication, Service_Parameters)*)>
<!ELEMENT  Alarm.table (Alarm)+ >
<!ELEMENT  Alarm  (id_alarm, alarm, desciption) >
<!ELEMENT  id_alarm  (#PCDATA) >
<!ATTLIST id_alarm
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  alarm  (#PCDATA) >
<!ATTLIST alarm
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  desciption  (#PCDATA) >
<!ATTLIST desciption
            Type CDATA  "text"
            Size CDATA  "50">
<!ELEMENT  Authentication.table (Authentication)+ >
<!ELEMENT  Authentication  (id_authentication, Authentication_name, Authentication_type, value, id_application) >
<!ELEMENT  id_authentication  (#PCDATA) >
<!ATTLIST id_authentication
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  Authentication_name  (#PCDATA) >
<!ATTLIST Authentication_name
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  Authentication_type  (#PCDATA) >
<!ATTLIST Authentication_type
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  value  (#PCDATA) >
<!ATTLIST value
            Type CDATA  "text"
            Size CDATA  "50">
```

```
<!ELEMENT id_application (#PCDATA) >
<!ATTLIST id_application
        Type CDATA "number"
        Size CDATA "4">
<!ELEMENT Connection_Trace.table (Connection_Trace)+ >
<!ELEMENT Connection_Trace (id_temp, session_id, ip_s_add, ip_d_add, ip_ps_add, ip_pd_add, id_Service_Authentication, id_Service_Parameters, id_service_alarm, data_in, data_out) >
<!ELEMENT id_temp (#PCDATA) >
<!ATTLIST id_temp
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT session_id (#PCDATA) >
<!ATTLIST session_id
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT ip_s_add (#PCDATA) >
<!ATTLIST ip_s_add
        Type CDATA "text"
        Size CDATA "50">

<!ELEMENT ip_d_add (#PCDATA) >
<!ATTLIST ip_d_add
        Type CDATA "text"
        Size CDATA "50">

<!ELEMENT ip_ps_add (#PCDATA) >
<!ATTLIST ip_ps_add
        Type CDATA "text"
        Size CDATA "50">

<!ELEMENT ip_pd_add (#PCDATA) >
<!ATTLIST ip_pd_add
        Type CDATA "text"
        Size CDATA "50">

<!ELEMENT id_Service_Authentication (#PCDATA) >
<!ATTLIST id_Service_Authentication
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT id_Service_Parameters (#PCDATA) >
<!ATTLIST id_Service_Parameters
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT id_service_alarm (#PCDATA) >
<!ATTLIST id_service_alarm
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT data_in (#PCDATA) >
<!ATTLIST data_in
        Type CDATA "text"
        Size CDATA "536870910">

<!ELEMENT data_out (#PCDATA) >
<!ATTLIST data_out
        Type CDATA "text"
        Size CDATA "536870910">
<!ELEMENT Mediator.table (Mediator)+ >
<!ELEMENT Mediator (id_ia, ia_name, ip_address, delegated_certificate) >
<!ELEMENT id_ia (#PCDATA) >
<!ATTLIST id_ia
        Type CDATA "number"
        Size CDATA "4">

<!ELEMENT ia_name (#PCDATA) >
<!ATTLIST ia_name
        Type CDATA "text"
        Size CDATA "50">
```

```
<!ELEMENT  ip_address  (#PCDATA) >
<!ATTLIST ip_address
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  delegated_certificate  (#PCDATA) >
<!ATTLIST delegated_certificate
            Type CDATA  "text"
            Size CDATA  "536870910">
<!ELEMENT  Network_Filter.table (Network_Filter)+ >
<!ELEMENT  Network_Filter  (id_network_politic, ip_source, ip_destination, por_source, port_destination, id_application) >
<!ELEMENT  id_network_politic  (#PCDATA) >
<!ATTLIST id_network_politic
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  ip_source  (#PCDATA) >
<!ATTLIST ip_source
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  ip_destination  (#PCDATA) >
<!ATTLIST ip_destination
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  por_source  (#PCDATA) >
<!ATTLIST por_source
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  port_destination  (#PCDATA) >
<!ATTLIST port_destination
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  id_application  (#PCDATA) >
<!ATTLIST id_application
            Type CDATA  "number"
            Size CDATA  "4">
<!ELEMENT  Security_Parameters.table (Security_Parameters)+ >
<!ELEMENT  Security_Parameters  (id_security_parameter, cipher_list, value, exportable) >
<!ELEMENT  id_security_parameter  (#PCDATA) >
<!ATTLIST id_security_parameter
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  cipher_list  (#PCDATA) >
<!ATTLIST cipher_list
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  value  (#PCDATA) >
<!ATTLIST value
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  exportable  (#PCDATA) >
<!ATTLIST exportable
            Type CDATA  "logical"
            Size CDATA  "2">
<!ELEMENT  Server.table (Server)+ >
<!ELEMENT  Server  (id_server, Server_name, ip_address, id_ia) >
<!ELEMENT  id_server  (#PCDATA) >
<!ATTLIST id_server
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  Server_name  (#PCDATA) >
<!ATTLIST Server_name
            Type CDATA  "text"
```

```
            Size CDATA  "50">

<!ELEMENT  ip_address  (#PCDATA) >
<!ATTLIST ip_address
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  id_ia  (#PCDATA) >
<!ATTLIST id_ia
            Type CDATA  "number"
            Size CDATA  "4">
<!ELEMENT  Service.table (Service)+ >
<!ELEMENT  Service  (id_service, service_name, used_port, id_Server, internal) >
<!ELEMENT  id_service  (#PCDATA) >
<!ATTLIST id_service
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  service_name  (#PCDATA) >
<!ATTLIST service_name
            Type CDATA  "text"
            Size CDATA  "50">

<!ELEMENT  used_port  (#PCDATA) >
<!ATTLIST used_port
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_Server  (#PCDATA) >
<!ATTLIST id_Server
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  internal  (#PCDATA) >
<!ATTLIST internal
            Type CDATA  "logical"
            Size CDATA  "2">
<!ELEMENT  Service_Alarm.table (Service_Alarm)+ >
<!ELEMENT  Service_Alarm  (id_service_alram, id_alarm, id_service) >
<!ELEMENT  id_service_alram  (#PCDATA) >
<!ATTLIST id_service_alram
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_alarm  (#PCDATA) >
<!ATTLIST id_alarm
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_service  (#PCDATA) >
<!ATTLIST id_service
            Type CDATA  "number"
            Size CDATA  "4">
<!ELEMENT  Service_Authentication.table (Service_Authentication)+ >
<!ELEMENT  Service_Authentication  (id_Service_Authentication, id_service, id_authentication, max_list) >
<!ELEMENT  id_Service_Authentication  (#PCDATA) >
<!ATTLIST id_Service_Authentication
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_service  (#PCDATA) >
<!ATTLIST id_service
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_authentication  (#PCDATA) >
<!ATTLIST id_authentication
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  max_list  (#PCDATA) >
<!ATTLIST max_list
```

```
            Type CDATA  "number"
            Size CDATA  "4">
<!ELEMENT  Service_Parameters.table (Service_Parameters)+ >
<!ELEMENT  Service_Parameters  (id_Service_Parameters, id_service, id_Security_Parameter, max_list) >
<!ELEMENT  id_Service_Parameters  (#PCDATA) >
<!ATTLIST id_Service_Parameters
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_service  (#PCDATA) >
<!ATTLIST id_service
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  id_Security_Parameter  (#PCDATA) >
<!ATTLIST id_Security_Parameter
            Type CDATA  "number"
            Size CDATA  "4">

<!ELEMENT  max_list  (#PCDATA) >
<!ATTLIST max_list
            Type CDATA  "number"
            Size CDATA  "4">
]>
```

Annexe D : Publications associées à cette thèse

Livre

1. Co-auteur avec Marilyne Maknavicius de la chapitre sécurité du livre "IPv6 : Théorie et pratique ", Editions O'Reilly, Paris (edition 4)

Revue

2. I. Hajjeh, A. Serhrouchni et R. Naja. Fourniture d'un service de non répudiation avec SSL/TLS pour le commerce électronique. A paraitre dans "l'Annales des Télécommunications".

Contribution à l'IETF

Status : Proposed Standard

3. P. Eronen (editeur), H. Tschofenig (editeur), M. Badra (auteur), O. Cherkaoui (auteur), I. Hajjeh (auteur) et A. Serhrouchni (auteur). Pre-Shared Key Ciphersuites for Transport Layer Security (TLS), IETF Internet Draft, draft-ietf-tls-psk-04.txt, (Proposed Standard) September 2004.

Status : Work in progress

4. I. Hajjeh, A. Serhrouchni, M. Badra et O. Cherkaoui. TLS-SIGN, IETF Internet Draft, draft-hajjeh-tls-sign-00.txt, January 2005 (work in progress).

5. M. Badra, O. Cherkaoui, I. Hajjeh et A. Serhrouchni. Pre-Shared-Key key Exchange methods for TLS. IETF Internet Draft, draft-badra-tls-key-exchange-00.txt (work in progress), July 2004.

Communications internationales avec comité de lecture

6. I. Hajjeh, M. Badra, A. Serhrouchni. Building a Secure and Extensible Protocol for wired and wirless environments. "VTC'2005 Spring, 61st Vehicular Technology Conference Spring 2005 Wireless Access", Stockholm, Sweeden. May 2005.

7. M. Hussain, I.Hajjeh, H. Afific et D. Sereta. Tri-party IKEv2 in Home Networks. "IEEE ICACT'2005, 7th International Conference on Advanced Communication Technology", Gangwon-Do (Korea), February 2005.

8. I. Hajjeh et A. Serhrouchni. Integrating a signature module in SSL/TLS. "ACM-IEEE ICETE'2004, First International Conference on E-Business and Telecommunication Networks". Setúbal (Portugal), August 2004.

9. Meddahi, K. Masmoudi, H. Afifi, A. M'Hamed et I.Hajjeh. Enabling Secure Third Party Control on Wireless Home Network. "IEEE ASWN'2004, 4th Workshop on Applications and Services in Wireless Networks", Boston, (USA), August 2004.

10. I. Hajjeh, A. Serhrouchni et F. Tastet. ISAKMP Handshake for SSL/TLS. "IEEE GLOBECOM'2003, Global Communications conference". San Francisco (California, USA), Vol. 3, pp. 1481-1485, December 2003.

11. I. Hajjeh, A. Serhrouchni et F. Tastet. New Key Management Protocol for SSL/TLS. "IEEE-IFIP NETCON'2003, Network Control and Engineering for QoS, Security and Mobility". Muscat (Oman), Vol. 1, N. 19, pp. 251-262, octobre 2003.

12. I. Hajjeh, A. Serhrouchni et F. Tastet. Vers l'intégration de nouveaux services dans SSL/TLS. "IEEE SETIT'2003, International Conference Sciences of Electronic, Technology of Information and Telecommunications". Sousse (Tunisie), Vol. 1, N. 106, pp. 52, Mars 2003.

13. I. Hajjeh, A. Serhrouchni et F. Tastet. A new Perspective for ebusiness with SSL/TLS. "SSGRR'2003w, Fifth International Conference on Advances in Infrastructure for eBusiness, eEducation, eScience, on the Internet". L'aquila (Italy), Vol. 1, N. 84, pp. 56, January 2003.

Communications nationales avec comité de lecture

14. I. Hajjeh et A. Serhrouchni. Génération d'une preuve de non répudiation dans SSL/TLS. "SAR'2004, 3ème Rencontre francophone sur Sécurité et Architecture Réseaux". La Londe, Cote d'Azur (France), 21-25 Juin 2004 (Selectionné comme meilleur papier).

15. I. Hajjeh, A. Serhrouchni et F. Tastet. Une nouvelle perspective pour SSL/TLS avec ISAKMP. "SAR'2003, 2ème Rencontre francophone sur Sécurité et Architecture Réseaux". Nancy (France), pp. 55-63, Juin 2003.