Sécurité pour les réseaux du futur. Gestion sécurisée des identités

Hassane Aissaoui Mehrez

Université Pierre et Marie Curie
Institut Mines-Télécom / Télécom Paris-Tech
École doctorale
Laboratoire d'informatique de Paris 6, UPMC
Équipe de recherche PHARE.

Sécurité pour les réseaux du futur
Gestion sécurisée des identités

Hassane AISSAOUI MEHREZ

Thèse de doctorat
Doctorat Paris 6 Informatique, télécommunications et électronique de Paris

Dirigée par Pascal URIEN & Guy PUJOLLE

Présentée et soutenue publiquement le 10/07/2015

Devant un jury composé de :

SAUVERON Damien, Maître de Conférences & HDR XLIM, Université de Limoges	Rapporteur
CHAUMETTE Serge, Professeur, LaBRI, Université de Bordeaux	Rapporteur
BOUABDALLAH Abdelmadjid, Professeur, Heudiasyc, UTC de Compiègne	Examinateur
PARADINAS Pierre, Professeur, CEDRIC, CNAM, PARIS	Examinateur
SIEGELIN Christoph, Directeur, Semiconductor Purchasing & Supply, Gemalto à Meudon	Examinateur
URIEN Pascal, Professeur, INFRES, IMT - Télécom Paris Tech, à Paris	Directeur de thèse
PUJOLLE Guy, Professeur, PHARE, LIP6, UPMC, à Paris	Co-directeur de thèse

Dédicace

« … Mais l'homme ne doit jamais s'avouer vaincu (…). Un homme, ça peut être détruit, mais pas vaincu… » Ernest Hemingway. Extrait du livre : Le vieil homme et la mer.

Je dédie ce modeste travail et ma profonde gratitude à tous ceux qui me sont les plus chers :

- À ma mère (1933, août 2013) qui m'a transmis la vie, l'amour et mon père (1930, juin 2006) le courage et la persévérance. À toi ma chère mère et à toi mon cher père toutes mes joies, mon amour et ma reconnaissance, pour l'éducation que vous nous avez prodiguée, le sens du devoir et les valeurs universelles que vous nous avez inculquées. Ce travail est le fruit de tous vos sacrifices consentis à notre égard et au prix de toutes les privations depuis notre enfance. Aucune dédicace ne suffit guère pour exprimer l'attachement, l'amour, l'estime et le respect que j'ai pour vous. Vous êtes présents à chaque instant dans mon esprit.

- À ma femme Latifa : ma fidèle compagne (Jahmouma) et mon ange gardien dans les moments les plus délicats, pour son soutien et sa patience, sans son aide et ses encouragements ce travail ne s'achèvera ! Ce travail est un témoignage de ma reconnaissance de mon amour sincère et fidèle le long d'un chemin commun et serein.

- À mes enfants Ayman et Mayssa-Nahelle, qui ont supporté mes absences durant ces dernières années.

- À mes deux sœurs (Fatima et Rahma) et mes deux frères (Moustapha et Yahya), j'adresse mon témoignage d'attachement et d'affection que je porte pour vous. Malgré la distance qui nous sépare, vous êtes toujours dans mon cœur. Je vous remercie pour votre hospitalité sincère et sans égale. Ce modeste travail exprime mon affection profonde pour votre soutien inconditionnel, ainsi qu'à mes belles sœurs mes beaux-frères, mes nombreux et merveilleux cousins, cousines, nièces et neveux. Vous étiez toujours présents pour les bons conseils. Votre amour et votre soutien m'ont été d'un grand secours.

- À mes beaux-parents et en particulier mon beau-frère Omar Agzour et Monsieur David Peover qui ont pris part à ce travail en m'aidant à corriger mes livrables et mes publications. Ce modeste travail exprime ma reconnaissance pour vos efforts.

- Enfin, à mes ami(e)s et à toute personne qui m'a soutenue, guidée et aidée à finaliser cette thèse.

Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui ont rendu cette thèse possible par leurs aides et leurs contributions.

Mes premiers remerciements sont adressés tout d'abord au Professeur Pascal URIEN (IMT-Télécom Paris Tech, ex-ENST) mon directeur de thèse pour avoir accepté de m'impliquer dans le projet SecFuNet. Je tiens à lui exprimer mes sincères respects pour m'avoir encadré au cours de ma thèse. Mes sincères remerciements vont également au Professeur Guy PUJOLLE (UPMC) mon co-directeur de thèse pour avoir accepté de m'encadrer et de m'intégrer dans son équipe. Je tiens à exprimer, ma profonde reconnaissance à ma collègue et amie Madame Noémie Simoni "le catalyseur de cette thèse" pour m'avoir poussé à m'inscrire en doctorat et encouragé à aller jusqu'au bout pour poursuivre mes recherches qui sans elle, n'auraient jamais eu lieu. Elle a non seulement entrepris la lourde tâche de relecture, mais aussi m'a fait part de ses remarques très précieuses dont j'ai tenu le plus grand compte. Je n'oublierai jamais la confiance qu'ils m'ont accordée et leurs qualités scientifiques qui ont contribué énormément à la progression de mes travaux de recherche. Je les remercie d'être à la fois une référence scientifique pour mes recherches et un capital inestimable pour leurs qualités humaines.

Je tiens également à remercier les membres de mon jury de thèse. Je suis reconnaissant envers Monsieur Damien SAUVERON, Maître de Conférences à l'Université de Limoges et chercheur au laboratoire XLIM, et Monsieur Serge CHAUMETTE, Professeur à l'Université de Bordeaux et chercheur au laboratoire LaBRI, pour avoir bien voulu rapporter cette thèse, et pour votre lecture consciencieuse, ainsi que vos remarques et commentaires pertinents. Mes remerciements vont également aux examinateurs de ma thèse qui ont eu l'amabilité d'examiner ma thèse. Je remercie Monsieur Le Président du jury Pierre PARADINAS, Professeur au CNAM à Paris. Je tiens à remercier également : Monsieur Abdelmadjid BOUABDALLAH, Professeur et Directeur du département Informatique à l'UTC de Compiègne et Monsieur Christoph SIEGELIN, Directeur, Semiconductor Purchasing & Supply, Gemalto à Meudon.

Mes remerciements chaleureux vont à toutes les personnes avec qui j'ai été amené à discuter et à valider mes travaux de recherche. Mes pensées les plus profondes et chaleureuses à mes ami(e)s, et à tous les habitants de mon village (TAOURIRT), que j'ai côtoyés.

Résumé

Aujourd'hui, l'Internet change radicalement nos habitudes, avec l'arrivée massive du nomadisme, l'internet des objets, l'utilisation croissante de l'informatique en grille, les services Web, les réseaux sociaux et l'émergence de nouvelles approches ces dernières années.

La virtualisation des infrastructures informatiques et le Cloud Computing ont particulièrement permis de définir de nouveaux paradigmes, appelés X as a Service (XaaS), introduisant ainsi une rupture assez franche avec les modèles traditionnels, qui sont perçus comme une étape préparatoire vers l'Internet du Futur.

En effet, la mise en œuvre de ces paradigmes permet de mutualiser et de réorganiser le système informatique de manière différente, de dématérialiser les infrastructures physiques, de déporter les systèmes ou les applications sur des conteneurs virtuels distants. Par conséquent, l'architecture globale de l'Internet doit évoluer, en s'appuyant fortement sur ces nouvelles approches, en particulier, le Cloud Computing et la virtualisation.

Malheureusement, comme toute technologie nouvelle, elle crée de nouveaux risques, qui viennent se greffer aux problèmes traditionnels : la séparation des privilèges, la gestion des accès, la gestion de l'identité, les failles des logiciels de virtualisation, l'isolation des machines virtuelles (VM), la protection des données personnelles, la vie privée, la réversibilité pendant l'externalisation (Outsourcing), etc.

Les services basés sur le Cloud requièrent des fonctions de collaboration interfonctionnelles sécurisées ainsi que des systèmes de protection contre l'utilisation abusive des ressources. Ces systèmes doivent être équilibrés de façon raisonnable avec les besoins de confidentialité, d'intégrité et de protection de la vie privée des utilisateurs. Ils doivent permettre l'authentification des utilisateurs sans révéler d'informations sur leur identité.

Ainsi, une offre de services personnalisés dans un environnement virtuel et/ou transorganisationnel, en utilisant des mécanismes de sécurité adaptés à des infrastructures traditionnelles, peut prendre une dimension très complexe dans le modèle Cloud Computing, et peut constituer des défis à lever pour les fournisseurs de ces services.

Parmi ces défis à résoudre, la gestion des identités des ressources, qui constitue un élément crucial pour authentifier les services à consommer, minimiser le risque d'accès frauduleux à des données personnelles, qui peut conduire à des conséquences désastreuses pour une entreprise ou un client.

Les solutions existantes sont insuffisantes pour répondre aux défis soulevés par ces nouvelles approches. La mise en œuvre de ces modèles et de ces outils pose des défis sécuritaires à la fois d'ordre organisationnel, architectural et protocolaire, pour garantir à chaque client des niveaux de sécurité adaptés. Ces niveaux doivent être identifiés pour guider les choix architecturaux et techniques à prendre, pour répondre en particulier aux exigences LoA (Level of Assurance) et LoT (Level of Trust), qu'un fournisseur de Cloud doit mettre en place pour garantir et protéger ses ressources.

En effet, ces verrous et ces défis sécuritaires vont être relevés dans ce travail de recherche qui se situe dans le cadre du projet sécurité pour les réseaux du futur (SecFuNet : Security for Future Networks). C'est un projet collaboratif entre l'Europe et le Brésil, qui implique neuf partenaires européens (répartis sur la France, la Pologne, l'Allemagne et le Portugal et 7 partenaires académiques brésiliens).

Ce projet a pour ambition de proposer une nouvelle infrastructure de sécurité générale pour la communication des informations des utilisateurs sur Internet. L'objectif principal est de concevoir et développer une nouvelle architecture de sécurité cohérente pour les réseaux virtuels. Le périmètre du projet a été défini selon plusieurs axes de recherche stratégiques, qui constituent des évolutions complémentaires des infrastructures des réseaux du futur, en particulier, les deux axes suivants :

- ✓ La gestion sécurisée des identités pour préserver les propriétés de confidentialité, de disponibilité, d'intégrité, de traçabilité et la protection de la vie privée.
- ✓ La gestion et l'identification des ressources dans les environnements virtuels.

Dans ce travail de recherche, il s'agit principalement de sécuriser les environnements virtuels, l'identification des utilisateurs d'Internet et la protection de leur vie privée. Ceci nous a conduits à identifier les verrous et les besoins du nouveau contexte défini par l'intersection de ces ressources virtuelles et de l'utilisateur final.

La première partie de cette thèse a été consacrée à l'analyse et à l'étude approfondie des solutions et des standards de gestion des identités et leurs techniques d'intégration au sein d'une architecture de sécurité. Nos réflexions nous ont conduits à identifier quelques verrous à franchir pour proposer des solutions de gestion des identités pour résoudre les problèmes de confiance engendrés par l'absence des éléments sécurisés physiques associés aux ressources virtuelles. En réponse à ces nouveaux défis dégagés dans la première partie, nous avons

proposé des contributions, principalement dans la seconde partie de ce mémoire, qui prennent en considération les aspects organisationnels, architecturaux et protocolaires.

Du point de vue organisationnel, nous nous sommes intéressés à l'orchestration et à la segmentation des couches de ces environnements virtuels, pour délimiter le domaine de sécurité de chaque fournisseur de service et la gestion flexible de ses actifs. Cette nouvelle architecture organisationnelle permet de définir, à chaque niveau de visibilité, les acteurs et les rôles que chacun joue pour gérer et contrôler l'accès aux ressources dans un environnement virtuel et hétérogène.

Du point de vue architectural et protocolaire, nous avons proposé une solution de gestion des identités qui vise à sécuriser les ressources virtuelles, en utilisant des Modules Matériels de Sécurité (ou HSM : Hardware Security Module), permettant d'associer chaque ressource (virtuelle ou physique) à un microcontrôleur sécurisé distant. Ce module permet de fournir ainsi les avantages sécuritaires exigibles au cœur de notre environnement virtuel.

La gestion de ces éléments de sécurité, est rendue facile grâce à l'implémentation de briques architecturales, qui intègrent une couche protocolaire, pour faciliter les échanges des méthodes d'authentification embarquées dans ces composants, permettant ainsi d'établir des authentifications mutuelles fortes, et des relations de confiance entre les ressources.

Enfin, la dernière partie de ce mémoire est consacrée à l'intégration et l'implémentation de ces modèles et de ces briques logicielles dans le système de gestion de l'identité (OpenID), pour compléter ses fonctions de sécurité. Nous expliciterons plusieurs scénarios concrets, pour tester et valider l'orchestration de ces modèles de gestion d'identité, afin de constituer une preuve de concept.

Abstract

Today, the Internet is changing radically our habits, especially with the massive influx of the nomadic techniques, the Internet of objects, the growing use of grid computing, wireless networks and the emergence of new approaches in recent years. In particular, the virtualization of the computing infrastructures, which allowed defining a new model called Cloud Computing, introducing an enough frank breakdown with the traditional models, can be perceived as a preparatory stage towards the Internet of future.

The implementation of these approaches allows, in a different way: mutualization and organization of the computer system. It allows to dematerialize the physical infrastructures and to deport applications on distant containers. Therefore, the global architecture of Internet should be evolved. It will rely strongly on these new approaches and in particular, Cloud Computing and virtualization. However, no system is infallible especially if resources are distributed and mutualized. They raise a number of problems and involve directly security issues, which remain one of the main barriers to the adoption of these technologies.

Like any new technology, Cloud Computing and virtualization create new risks, which come to graft to traditional threats of the outsourcing management of the privilege separation, the identity and accesses management, the robustness of the virtualization software, the virtual machine isolation, the personal data protection, reversibility, privacy, etc. The traditional Internet architecture cannot provide the adequate solutions to the challenges raised by these new approaches: mobility, flexibility, security requirements, reliability and robustness. Thus, a research project (SecFuNet : Security For Future Networks) was validated by the European Commission, to provide some answers, to make a state of the art of these security mechanisms and a comprehensive study of orchestration and integration techniques based on protection components within overall security architecture.

The ambition of SecFuNet project is to propose a new general secure framework for communication of user information on the Internet. The main objective is to design and develop new coherent security architecture for virtual networks. The scope of the project has been defined in several strategic research areas, in particular, the following two areas :

- Secure identity management to preserve the confidentiality properties, availability, integrity, traceability and protection of privacy,

- Identification and mutual authentication of resources in virtual environments.

The main objective is to design and develop new consistent security architecture for secure identification of users and protect their privacy in virtual networks. The proposed architecture will make possible the communication security management for all machines connected to a virtual public Cloud.

Our research work is organized in three parts:

The first part of this thesis was devoted to make a state of the art of security mechanisms and an overall study of the orchestration and integration dynamic of different bricks of the security to ensure identification, authentication, confidentiality, protection of privacy, and management of secure key storage. In response to these new challenges identified in the first part, we propose contributions, mainly in the second part, which consisting to take into account three dimensions: organizational, architectural and protocol.

From an organizational perspective, we are interested in orchestration and segmentation of the virtual environments layers to define the security domain for each service provider and flexible management of these assets. This new organizational architecture allows defining, at every level of visibility, actors and roles that each plays to manage and control access to resources in a heterogeneous virtual environment.

From an architectural and protocol point of view, we suggest a new model of identity management in order to secure virtual resources, using HSM (Hardware Security Module). We associate each resource (virtual or physical) to a remote secure element. This module allows strong mutual authentications in our virtual environments. The management of these security features is made easy thanks to the implementation of architectural bricks which integrate a protocol layer, that facilitate the exchange messages of the authentication methods embedded in these secure components, thereby establishing strong mutual authentication, and trust relationships between resources.

The last part is devoted to the integration of these models and the implementation of these software bricks in the identity management system (e.g. OpenID), aimed to complement its security features. We explained specific scenarios to show how our different contributions fit together, to test and validate its orchestration and to provide a proof of concept.

Sommaire

Introduction générale :

Avant de proposer une nouvelle infrastructure globale de référence et de repenser le modèle de gestion des identités adapté à l'Internet de demain, il convient dans un premier temps de faire une analyse du contexte actuel pour clarifier le paradigme du Cloud Computing, les services qu'il offre, le contexte d'un modèle de gestion des identités au sein de ce paradigme et les enjeux liés à la sécurité.

Analyse du contexte :

Le paradigme du Cloud Computing est né d'un besoin de coopération et de mutualisation des ressources matérielles, logicielles, humaines pour atteindre un objectif commun : l'optimisation des ressources pour offrir des ressources et des services Internet.

Le Cloud a fait naître de nouvelles opportunités mais aussi de nouveaux risques. Malgré les contraintes liées à l'interconnexion et à la communication réseau, ce paradigme a révolutionné au quotidien le mode de travail et son organisation, impacté les habitudes des utilisateurs dans leurs consommations des services. Ces contraintes poussent, de manière plus accentuée, à la recherche de nouvelles solutions organisationnelles et techniques dans le domaine de la sécurité. Celles-ci permettront de conserver une vision globale de l'état des actifs en appliquant une politique de sécurité, d'améliorer les mécanismes de protection des ressources informatiques, d'offrir des outils de sécurité et des moyens de contrôle. En outre, ces solutions doivent offrir des méthodes et des outils pour supporter la mutualisation et l'interconnexion des infrastructures virtuelles, lesquelles posent des défis sécuritaires à la fois d'ordre structurel, organisationnel et technique.

Pour résoudre en partie ces défis, les fournisseurs du Cloud adoptent les mêmes mécanismes sécuritaires que ceux de l'Internet traditionnel, pour protéger leurs ressources informatiques, bâtir des infrastructures distribuées, collaborer et partager des ressources entre les fournisseurs et les clients de manière transparente. C'est lors d'une création d'une stratégie de gestion des identités et de contrôle des accès que ces fournisseurs découvrent toutes les variables (i.e. ressources) impliquées dans cette équation, et deviennent conscients de l'ampleur et saisissent la complexité que comportent ces environnements disparates et distribués.

Malheureusement, la sécurité ne se limite pas seulement à la protection des ordinateurs et des applications mais elle doit s'étendre à l'ensemble des nœuds d'accès au réseau, qu'ils soient interne ou externe.

Dans ce contexte la sécurité, la gestion des identités et le contrôle des accès de chaque ressource (physique, virtuelle, application ou utilisateur), sont devenus des enjeux primordiaux pour le développement du Cloud et la protection des environnements virtuels.

Enjeux de sécurité :

Ce modèle économique innovant, qu'il soit un phénomène de mode ou une véritable révolution dans le monde de l'informatique, soulève de nombreuses interrogations. Il suscite d'une part une méfiance pour la sécurité offerte par cette informatique délocalisée et d'autre part un intérêt pour son paradigme d'offre XaaS (X as a Service), accessible directement par l'Internet et facturé à la consommation réelle.

En effet, le Cloud Computing offre, à moindre coût, des avantages aux organisations en termes de flexibilité et d'efficacité opérationnelle. Seulement voilà, aucun système n'est infaillible dans le temps, surtout en informatique distribuée et mutualisée. Cette informatique entraîne de manière directe des menaces, ce qui demeure l'un des principaux obstacles à l'adoption de ces technologies [75]. Les mécanismes traditionnels ne répondent plus aux besoins de sécurité associés à ces approches [39].

En effet, de multiples affaires d'intrusion ou de pannes de centres informatiques (Datacenter) ternissent la réputation des fournisseurs de Cloud. On peut citer en particulier les affaires suivantes :

- Les difficultés auxquelles ont fait face les services Amazon S3 et EC2 [63],
- La découverte d'un Cheval de Troie (Bohu), chez Microsoft qui rend vulnérable les postes de travail [64],
- Le problème qui concerne le jeu PS3 de Sony [65].

On peut citer aussi l'incident survenu en 2009 au service OVI de Nokia, qui lui a fait perdre la totalité de ses données, suite à une panne de refroidissement, ou bien la panne qui a entraîné l'indisponibilité des sites Web d'Amazon (Foursquare ou Quora), etc.

Le Cloud Computing devient donc une cible privilégiée et intéressante pour le vol de données confidentielles. Pas vraiment de quoi séduire et tranquilliser les décideurs. 91 % des membres de l'Information Security Forum (ISF) qui regroupe plus de 300 membres dont des RSSI et des Risk manager, estiment que le Cloud Computing accroît les menaces de sécurité et ne sont donc pas prêts à l'adopter. Cela montre, qu'il y a encore beaucoup de progrès à faire pour convaincre les décideurs de la sécurité du Cloud.

Selon une étude du CLUSIF, les entreprises ont souvent un a priori négatif sur l'externalisation et les mécanismes de sécurité des infrastructures du Cloud. Pour les Directeurs des Systèmes d'Information (DSI), externaliser leurs actifs ou leurs données pour les confier à un tiers, c'est perdre une partie du contrôle de leur système d'information et de la maîtrise du contrôle de leur infrastructure.

Une autre étude de 2011, réalisée par Trend Micro, montre que 43 % des 1 200 entreprises interrogées du secteur des nouvelles technologies, rencontrent des problèmes de sécurité avec les fournisseurs de Cloud [50]. À cause de toutes ces zones d'ombre autour de la sécurité du Cloud Computing, certaines grandes entreprises préfèrent garder leur "Data Center" en interne.

Par conséquent, l'architecture globale d'Internet doit évoluer pour s'adapter à ces nouveaux besoins. Elle doit s'appuyer, non seulement sur la virtualisation et le Cloud Computing pour relever ces défis, mais doit prendre en compte le cycle de vie, la gestion des identités et le contrôle des accès pour offrir des accès aux ressources virtuelles, avec un haut niveau de sécurité et des niveaux de garanties à ses Clouds. Ces garanties vont influencer les choix architecturaux et techniques, pour répondre en particulier aux niveaux d'Assurance (LoA : Level of Assurance) et/ou aux niveaux de Confiance (LoT : Level of Trust) dont chaque fournisseur entend doter ses ressources dans le Cloud.

En effet, chaque prestataire doit être en mesure de proposer des outils à ses clients afin de lui permettre de définir et de contrôler ces niveaux (LoA et LoT). Ces outils seront acceptés et évalués en termes d'objectifs métiers.

Les fournisseurs doivent garantir également ces niveaux en matière de gestion d'identification et d'authentification, de confidentialité et de sécurité des données. Ces niveaux de garantie permettront de séduire de nouveau les entreprises, d'autant plus que ces hébergeurs sont des spécialistes de la sécurité et possèdent des moyens de défense efficaces : équipe informatique dédiée à la sécurité, moyens de cryptage, charte de qualité, etc.

Enjeux liés à la gestion de l'identité :

Pour des raisons de compétitivité, chaque entreprise est obligée de s'ouvrir vers l'extérieur. Cette ouverture augmente les risques de fraude et d'abus : montée en puissance des menaces extérieures (usurpation d'identité, attaque MITM (Man In The Middle), Cheval de Troie, etc.), perte des machines virtuelles (VM), manque d'interopérabilité entre ces

fournisseurs de services et de procédure pour la migration des ressources (virtuelles, données, applications, utilisateurs, etc.) d'un Cloud à un autre.

En outre, les systèmes actuels de gestion des identités et le contrôle des accès sont-ils capables de permettre aux utilisateurs des accès plus larges et plus simples aux systèmes d'information au-delà d'un usage interne ? Permettent-ils également de faciliter l'administration des comptes utilisateurs et l'utilisation des ressources dans un contexte distribué et virtuel ? Peut-on étendre le référentiel d'une organisation aux clients, aux partenaires et aux fournisseurs pour offrir un accès sécurisé à leurs ressources clés ?

En effet, le manque d'un standard de gestion des identités des ressources rend la gestion de leur cycle de vie plus complexe, ce qui constitue un des principaux défis à relever.

Ces risques qui pèsent sur la violation des données sensibles et la vie privée des utilisateurs, sont négligés et récoltent peu d'intérêt de la part des cadres dirigeants et des fournisseurs de services, particulièrement pour des raisons de coûts élevés liés aux contraintes de développement de solutions de sécurité. Bien souvent, le mécanisme obsolète le plus répandu est l'utilisation du couple "Login/mot de passe", dont les lacunes sont encore nombreuses. Parfois ce modèle est combiné avec la fameuse "authentification unique" (SSO : Single-Sign-On). Ce modèle augmente encore les chances de violation d'un système d'information et la non-détection d'une attaque avant plusieurs semaines voire même plusieurs mois.

La mise en œuvre d'un système de gestion des identités implique l'établissement de relations de confiance pour l'étendre à des partenaires externes. Toutefois, cette procédure complexe peut augmenter les frais liés à la gestion. La nécessité donc de mettre en place des prestataires de gestion des identités et de contrôle des accès peut s'avérer une alternative intéressante pour jouer le rôle d'intermédiaire essentiel pour authentifier des ressources réseau et réduire ainsi le coût de gestion du cycle de vie du système de gestion des identités.

Contexte du projet SecFuNet :

SecFuNet (Security for Future Networks) est un projet collaboratif entre l'Europe et le Brésil, qui implique neuf partenaires européens répartis sur la France, la Pologne, l'Allemagne et le Portugal et 7 partenaires académiques brésiliens.

Ce projet a pour ambition de proposer une nouvelle infrastructure de sécurité générale pour la communication des informations des utilisateurs sur Internet. L'objectif principal est

de concevoir et de développer une nouvelle architecture de sécurité cohérente pour les réseaux virtuels.

Le périmètre du projet a été défini selon plusieurs axes de recherche stratégiques (Figure 1), qui constituent des évolutions complémentaires des infrastructures des réseaux du futur. Dans ce travail de recherche, il s'agit principalement de sécuriser les environnements virtuels, l'identification des utilisateurs d'Internet et la protection de leur vie privée.

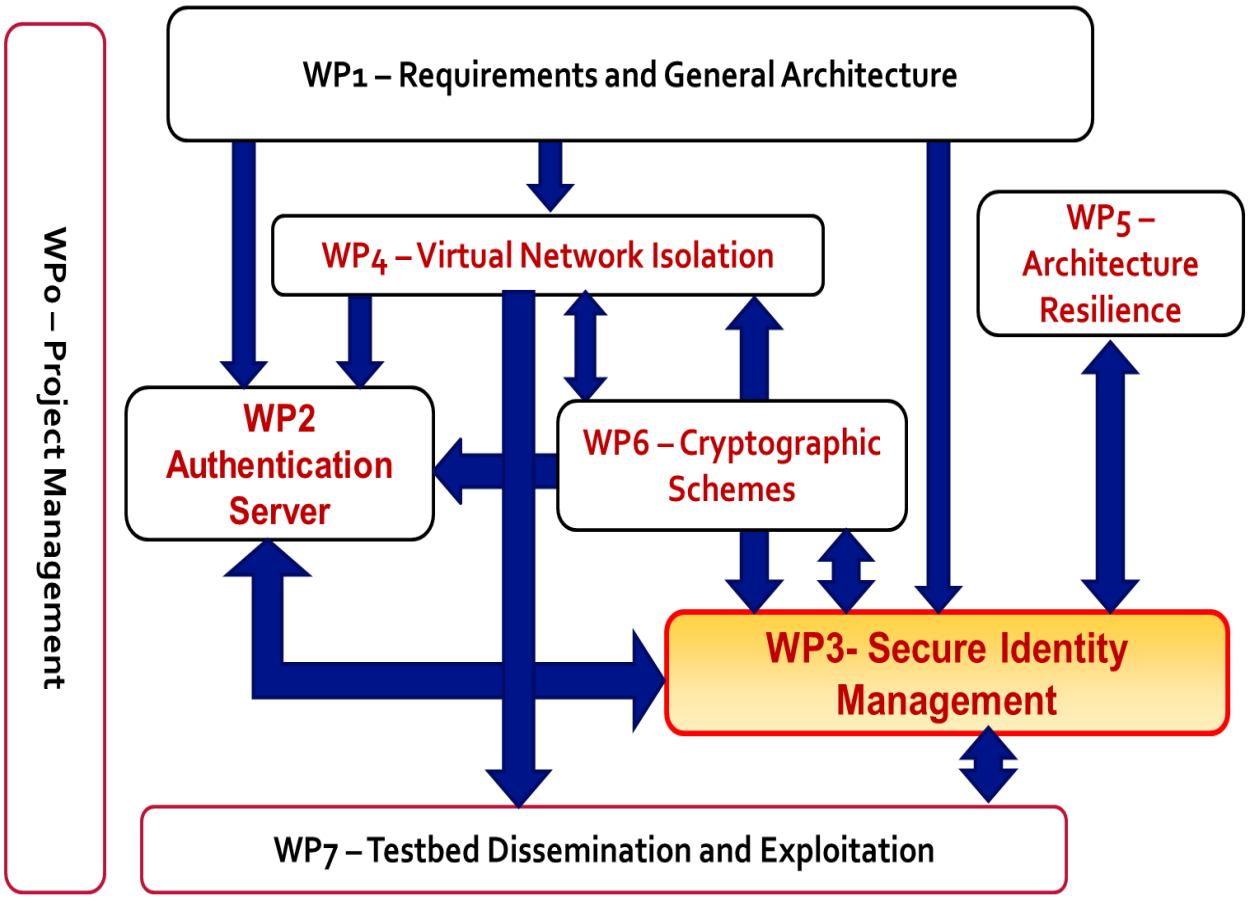

Figure 1 : Les axes de recherches du projet SecFuNet.

Périmètre de la thèse :

Après avoir décrit les différents contextes et enjeux, nous délimitons notre périmètre d'étude, comme le montre la Figure 2, à l'intersection de trois contextes dans lesquels se posent un ensemble de problématiques, en particulier la gestion des identités des ressources et leur orchestration.

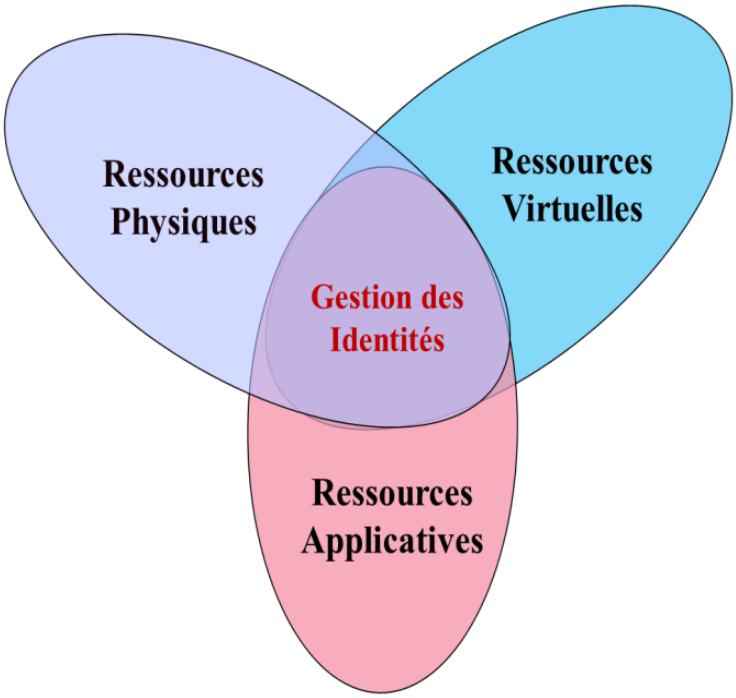

Figure 2 : Périmètre de la thèse.

Problématiques :

Le Cloud Computing amène donc un changement profond du modèle économique, la façon de développer des applications hébergées, et pose de nouveaux défis, aux équipes de sécurité informatique, pour la gestion de la stratégie et la politique de sécurité. L'impact des services offerts par le Cloud sur la sécurité informatique et les communications Internet est un sujet inexploré, qui suscite beaucoup d'interrogations et devient donc une priorité de recherche.

En effet, pour assurer la communication entre des entités, il existe actuellement un ensemble de mécanismes et d'outils utilisés au niveau système et réseau pour gérer et contrôler les ressources physiques d'une entité, ce qui répond aux exigences spécifiques à la sécurité de l'informatique traditionnelle interne à une organisation. Par contre, quand il s'agit d'un environnement virtuel hébergé dans le Cloud, lequel est modifié en raison d'une nouvelle réorganisation, cela peut augmenter les risques de sécurité sur l'ensemble de l'architecture et éventuellement, interrompre le service réseau aux autres infrastructures virtuelles hébergées dans ce même Cloud. Chaque modification structurelle ou organisationnelle constitue un risque accru et devient impossible à gérer. Elle expose les autres entités à des risques indus en raison de la complexité de gestion des actifs dans le Cloud.

Les solutions de gestion des identités et de contrôle des accès actuelles ne sont pas facilement intégrables dans le contexte SecFuNet. La garantie de sécurité offerte pour une entité peut être affectée par la modification d'une ressource appartenant à une autre entité.

Pour cela, il faut pouvoir répondre aux questions suivantes : Quelle est la solution de sécurité qui peut, de manière transparente, assurer une étanchéité entre les environnements virtuels et garantir des niveaux d'assurance et de confiance personnalisés selon les exigences et les préférences des entités ? Quel est le modèle de gestion des identités et de contrôle des accès pour ajouter de nouvelles ressources virtuelles et de nouveaux utilisateurs, sans impact sur l'architecture globale ? Qui pourrait gérer les identifiants, les clés de session et le cycle de vie des VMs ? Sous quel format ils seront stockés ?

Certes, le Cloud Computing facilite la migration des ressources (physiques, virtuelles, données, applications, utilisateurs, etc.), et offre des avantages en termes de flexibilité et de modularité opérationnelle, mais seulement quand les ressources virtuelles appartiennent à un même Cloud.

Nous devons donc, trouver des solutions aux verrous suivants : Quelle architecture faut-il implémenter pour assurer l'interopérabilité entre ces fournisseurs de services ? Quel modèle organisationnel devrait être mis en place pour identifier les VMS et assurer l'élasticité du Cloud Computing et son automatisation ? Qui pourrait piloter cette nouvelle architecture pour satisfaire la sécurité des ressources virtuelles ?

Contributions de la thèse :

Les contributions de cette thèse se situent autour d'un ensemble de propositions concernant l'orchestration et l'organisation de l'infrastructure Cloud Computing, et l'implémentation des briques architecturales sur lesquelles reposent nos modèles de gestion des identités et de contrôle des attributs, dont l'objectif principal est de lever certains verrous identifiés précédemment.

Afin de proposer un modèle de gestion des identités numériques adapté à SecFuNet, il convient avant tout d'analyser correctement les modèles traditionnels existants. Dans cet esprit, et pour guider nos choix, nous avons réalisé un état de l'art de ces solutions ainsi que les approches utilisées dans les infrastructures d'authentification et d'autorisation. D'un point de vue technologique, nos études et nos analyses se réfèrent à des standards et des modèles qui serviront de briques de base dans notre proposition. Suite à cette étude, nous avons proposé une nouvelle architecture de gestion des identités et de contrôle des accès, permettant

de gérer l'accès aux ressources partagées. Ces ressources sont associées à des éléments sécurisés (Secure Element : SE). Cette nouvelle architecture peut déléguer certains services à d'autres organisations quand une entité n'a pas les ressources ou les compétences.

Pour atteindre cet objectif, nous avons proposé d'un point de vue organisationnel, une infrastructure modulaire et en couche, pour garder les caractéristiques d'élasticité, de flexibilité et de dynamicité du Cloud Computing. Ainsi, nous avons opté d'une part pour une gestion des identités centrée sur l'utilisateur "User-Centric Identity Management" qui constitue la première couche et d'autre part, nous avons subdivisé les infrastructures XaaS dans SecFuNet en trois autres couches. À chaque couche, nous avons associé un système de gestion des identités fédérées, permettant à chaque entité de gérer ses propres utilisateurs et ses propres ressources (physiques et/ou virtuelles), d'une manière transversale au sein d'un domaine de sécurité.

La gestion de la cinématique des messages d'authentification dans cette infrastructure, s'appuie sur des briques architecturales sous forme de couches logicielles, intégrées dans l'architecture SecFuNet. Ces briques constituent notre deuxième contribution. Ces couches logicielles sont constituées de différents Proxys installés et exécutés en tâche de fond dans des nœuds correspondants. Chaque Proxy joue le rôle de médiateur, pour assurer l'identification et l'authentification mutuelle de deux ressources. Il réagit d'une façon autonome lorsqu'un changement se produit au cours d'une session ou une ressource tente d'accéder à une autre. En outre, ces Proxys intègrent une couche protocolaire RACS (Remote APDU Call Secure Protocol) [90], pour assurer la gestion des éléments sécurisés, ainsi que les échanges d'authentification entre deux entités.

Chacune de ces propositions a été développée en tant que prototype pour le démonstrateur final. Ce travail de thèse a en effet été réalisé dans le cadre du projet européen FP7 (SecFuNet), en collaboration avec des partenaires brésiliens, et des partenaires industriels. En revanche, la première implémentation du démonstrateur s'est orientée seulement vers la mise en œuvre des principales contributions présentées dans le cadre de cette thèse. L'objectif est de fournir les fonctions de base permettant de valider les modèles proposés, et de montrer que ces deux modèles de gestion des identités peuvent être utilisés dans différents cas d'usage, et ne se limitent pas seulement à une utilisation dans le cadre du Cloud Computing.

La seconde implémentation a été consacrée à l'intégration dans OpenSSL, d'un kit de développement logiciel, sous forme de bibliothèques, pour gérer les interactions, en particulier

avec la carte à puce EAP-TLS [84]. Cette contribution permettra de faciliter le développement d'applications sensibles.

Organisation du rapport :

Le présent manuscrit est organisé en trois parties. La première a pour objectif de présenter les modèles et les approches de gestion des identités au sein des architectures actuelles et les avantages qu'ils peuvent apporter aux différents environnements virtuels. La deuxième partie vise à présenter notre proposition d'infrastructure organisationnelle et architecturale pour segmenter les ressources en plusieurs couches et délimiter les domaines administratifs afin d'appliquer des politiques de sécurité. Enfin, la dernière partie est consacrée, pour les besoins du démonstrateur, aux implémentations de nos contributions, lesquelles ont été testées et validées.

Pour répondre plus concrètement aux différents objectifs de cette thèse, ce rapport de thèse se décline en sept chapitres :

Le Chapitre I analyse et définit les fonctions de base d'un système de gestion des identités (AAA : Authentication, Authorization, Accounting/Auditing) et les composants pour assurer son cycle de vie. En outre, nous avons étudié les niveaux d'exigences (LoA et LoT), et quelques attaques liées à la faiblesse des mots de passe.

Le Chapitre II s'intéresse à l'étude de l'état de l'art des standards AAA qui assurent ces fonctions de base, examine les composants clés de quelques standards et leurs imbrications avec les protocoles d'authentifications pour les prendre en considération dans l'élaboration de notre infrastructure de gestion des identités.

Le Chapitre III s'intéresse également à l'étude des approches des infrastructures d'authentification et d'autorisation, aux mécanismes de gestion des attributs et à la propagation de l'authentification d'un domaine de sécurité à un autre. Ce chapitre s'intéresse en particulier à la comparaison de SAML, WS-Federation, Shibboleth et Liberty Alliance, ainsi qu'aux approches centrées sur les utilisateurs comme OpenID, Cardspace, Higgins.

Le Chapitre IV consigne l'ensemble de nos propositions qui concernent la nouvelle infrastructure globale organisationnelle et architecturale, sous forme de briques logicielles sécurisées, pour gérer les échanges d'informations d'authentification dans des environnements virtuels, offrir des services d'identification et de confiance dans ces nouveaux paradigmes "XaaS". Ce chapitre décrit comment identifier chaque nœud ou chaque ressource virtuelle dans le Cloud et comment sécuriser leur identification et leur authentification.

Le Chapitre V propose l'intégration d'une technologie qui servira de socle de base dans notre modèle de gestion des identités, en combinaison avec le standard OpenID. Cette combinaison permettra d'établir des relations de confiance entre les utilisateurs, le fournisseur d'identité (Identity Provider : IdP), et le fournisseur de service (Service Provider : SP). Elle reposera principalement sur une machine à états embarquée dans des microcontrôleurs sécurisés, lesquels réaliseront des authentifications mutuelles fortes en utilisant le protocole d'authentification TLS.

Le Chapitre VI est consacré à l'implémentation d'un prototype pour le démonstrateur final. Dans ce chapitre, nous avons développé un serveur d'authentification associé à un Proxy Server pour gérer la grille de cartes à puce. Ensuite nous avons implémenté les autres briques logicielles : (1) un Proxy Agent qui joue le rôle de médiateur entre le poste client, sa carte à puce et le serveur d'authentification, (2) un Proxy VM qui joue le rôle d'intermédiaire entre un utilisateur qui souhaite consommer un service offert par une VM, la carte à puce associée à celle-ci et le serveur d'authentification. Le développement de ces Proxys démontre la faisabilité et la performance de nos propositions à travers des scénarios.

Enfin, Le Chapitre VII propose une bibliothèque pour les développeurs, intégrée aux outils OpenSSL, sous forme d'un kit de développement logiciel, pour faciliter la création d'applications sensibles en intégrant des microcontrôleurs sécurisés. Ce chapitre décrit un ensemble de fonctions primitives pour envoyer des commandes ISO 7816, à une carte à puce. Il s'achève par la présentation de quelques mesures de performances effectuées en réalisant notamment des connexions simultanées avec plusieurs composants localisés dans une grille de cartes à puce.

Cette thèse s'achève par la présentation d'une conclusion générale mettant en valeur les propositions et leur implémentation, et consigne nos perspectives de recherche et les cas d'usage qui peuvent être bâtis autour de l'infrastructure organisationnelle et architecturale présentée dans ce rapport.

PARTIE I : La gestion des identités au cœur du Cloud Computing

Chapitre I : État de l'art de la gestion des identités

1.1) Introduction

Une identité numérique peut être définie comme la représentation numérique d'un ensemble de données de façon unique. Elle est relative à un domaine d'application et peut représenter un sujet, un objet, une organisation. Dans ce contexte, une organisation peut être une personne (physique ou morale) ou un ensemble de personnes, un objet peut être représenté dans un système par un identifiant sous forme d'un processus ou d'une application (physique ou virtuelle), capable d'interagir avec une infrastructure réseau, cet identifiant doit être pris en compte par un système de gestion des identités. Selon le contexte ou le profil d'un objet, il peut être représenté par plusieurs identités distinctes (i.e. simple utilisateur, nomade, cadre, administrateur, etc.) qui lui sont associées dans un domaine administratif [2] [3]. Une identité numérique est composée d'un ensemble d'éléments servant de preuve à l'identification et/ou à l'authentification d'un objet.

Les systèmes de gestion des identités et de contrôle des accès, sont des processus qui visent à gérer le cycle de vie de chaque ressource informatique. Une de nos principales préoccupations dans ce chapitre, est d'étudier et d'examiner les composants des systèmes de gestion des identités, pour établir les limites et les exigences du modèle de gestion des identités global qui sera développé en tenant compte des spécifications du projet de SecFuNet.

1.2) La gestion des identités et le contrôle des accès

La gestion des identités et le contrôle des accès (en anglais : Identity and Access management : IAM) est un processus transversal impliquant la création d'un identifiant distinct pour chaque ressource (utilisateur, réseau, système, machine virtuelle, processus, service, etc.). L'IAM est utilisée pour cartographier, capturer, approvisionner, enregistrer et gérer les authentifications et les autorisations d'accès à des informations confidentielles. Il peut être sous forme d'un référentiel unique chapeautant toutes les ressources informatiques (comptes utilisateurs, badges d'accès électroniques, etc.). Il s'agit de savoir, au fil du temps, qui a accès à quelle ressource ou service et qui a fait quoi ? Ce cycle de vie passe par trois principales étapes (Figure 3) : création, utilisation et suppression d'une identité. L'IAM doit rendre les identifiants et les mots de passe homogènes et mettre en œuvre des contrôles appropriés autour de la sécurité des données.

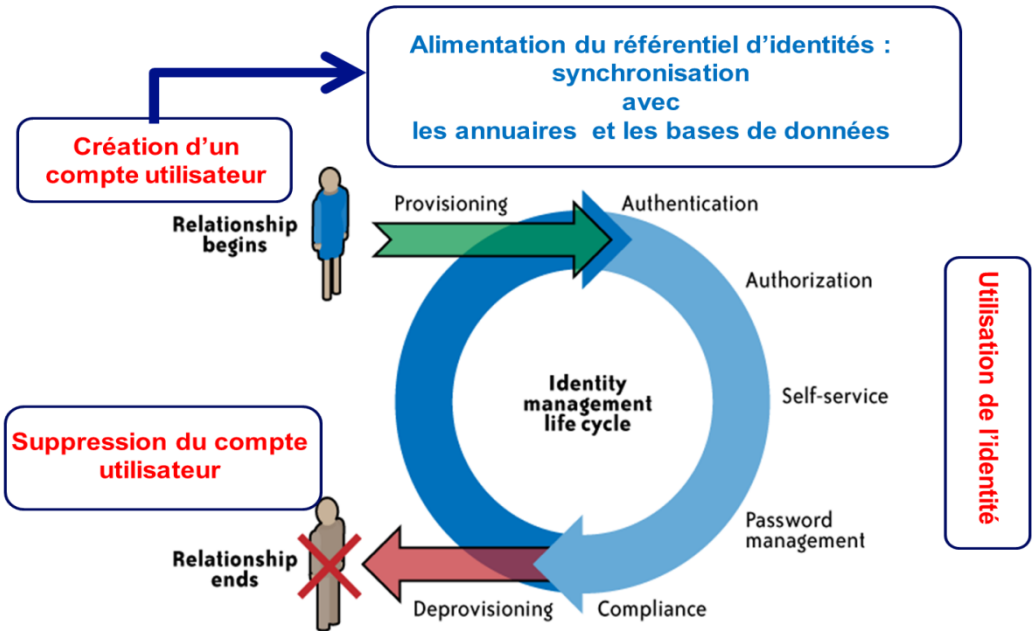

Figure 3 : Le cycle de vie de 'IAM et les trois principales étapes

1.2.1) Notions sur l'identité

Le mot identité peut être assimilé à un groupe d'appartenance, il peut désormais être utilisé indifféremment pour désigner une identité : ethnique, culturelle, nationale, etc., ou une pathologie mentale (troubles de l'identité), indique une préférence sexuelle (i.e. selon Jean-Paul Sartre, l'identité homosexuelle : identité lesbienne ou gay). Au sein de la littérature actuelle, on distingue plusieurs domaines d'étude relatifs à l'identité :

- Identité de groupe ou collective (anthropologues, historiens et spécialistes des sciences politiques),
- Identité personnelle (psychologues, psychanalystes et philosophes),
- Identité sociale (sociologues et psychologues sociaux),
- Identité numérique ou digitale (informaticiens).

L'identité est intimement liée aux groupes sociaux auxquels nous appartenons. La théorie de l'identité sociale place chaque individu en fonction de son interaction avec un milieu social, entre deux comportements extrêmes : interpersonnel et intergroupe. Suivant les cas, une personne adopte un comportement intergroupe, dans un contexte conflictuel, ce comportement est d'autant plus intense que le conflit est plus grave (par exemple : un supporter d'une équipe de football). Mais l'appartenance à un groupe peut avoir des répercussions positives ou négatives sur notre identité, en fonction de nos comparaisons à

d'autres groupes. La théorie de l'identité sociale représente le système de croyances des personnes et leur évaluation de ces structures relationnelles intergroupes dans deux pôles extrêmes : mobilité sociale ou changement social. Dans tous les cas, si la comparaison est défavorable à notre groupe (en cas de perte de performances de l'équipe de football), et par conséquent elle n'est pas satisfaisante pour notre identité sociale, nous cherchons à rejoindre le groupe qui a une meilleure classification par mobilité sociale ou à améliorer celle de notre groupe par un changement social (changement d'entraîneur par exemple).

1.2.2) Identité sociale ?

En psychologie sociale, l'identité de l'individu se traduit par l'appréciation de l'image qu'on a de nous-mêmes et la reconnaissance en tant que tel par autrui en fonction des contextes sociaux avec lesquels nous interagissons (dans ce cas on parle du rôle qu'on occupe dans un contexte social). Cette notion d'identité sociale est la connaissance de soi tirée de la reconnaissance singulière d'un groupe auquel on appartient et qui est considéré comme le lieu d'investissement identitaire. Elle englobe tout ce qui permet d'identifier une personne par son statut hiérarchique dans différents groupes d'appartenances et par son implication dans différents rôles. Décliner son identité, ce n'est pas seulement revendiquer une appartenance à un groupe communautaire, c'est aussi s'investir pour acquérir une position sociale.

Au final quand on se penche sur ce qui a été évoqué précédemment, on distingue deux concepts. Le premier concerne la notion de profil, rôle pour une même personne englobant ses différents groupes d'appartenances. Le deuxième concerne la notion d'ensemble d'informations qui représentent et caractérisent chaque personne d'une manière singulière dans une organisation et qui permettent de la différencier des autres. On voit plutôt poindre la notion d'identifiant et attributs.

1.2.3) Qu'en est-il de l'identité numérique ?

Dans le monde du numérique, tout devient virtuel. Ce nouveau monde nous incite à multiplier nos identifiants (carte à puce pour GSM, adresse e-mail, numéro de téléphone, login, pseudonyme pour Google, FaceBook, ou Twitter, etc.). Avec les réseaux sociaux, nous assistons à la naissance de l'individu schizophrène, car il choisit ses identifiants et son profil en fonction du contexte du réseau social ou du service offert par une application. Malheureusement, cet individu devient de plus en plus amnésique à force de multiplier ses

identités et par conséquent, oublie ses mots de passe. Aujourd'hui, l'identification des individus peut prendre plusieurs formes : cartes bancaires, badges pour un club, carte NAVIGO, etc. Ces formes d'identification comportent généralement des informations propres aux détenteurs. L'identité peut se composer de plusieurs éléments :

- L'identifiant : Il s'agit d'un ensemble d'informations et d'attributs qui identifient un sujet de manière singulière dans un contexte donné (également appelé parfois en anglais Login).
- Les informations d'identification (en anglais : credentials) : Il s'agit des informations publiques ou privées qui peuvent être utilisées pour prouver l'authenticité d'une demande d'identité, et prouver qu'il s'agit bien de celui qui prétend l'être.
- L'attribut principal : Il s'agit de données intrinsèques permettant de décrire l'identité d'une personne (état civil, photo, biométrie, empreinte, adresse, etc.).
- Les attributs propres au contexte : Il s'agit de données liées au métier (rôle, savoir-faire, statut, niveau d'accréditation, etc.) et de données organisationnelles (département, laboratoire, rattachement hiérarchique, secrétariat, etc.). Ils sont référencés et utilisés uniquement dans le contexte spécifique dans lequel l'identité est utilisée.

En général les identités et les attributs sont déclarés d'une manière unique dans un référentiel central de gestion des habilitations qu'on appelle un annuaire. Les habilitations et les autorisations sont attribuées en fonction du profil ou du rôle des utilisateurs. La cohérence et l'approvisionnement de ces informations sont maintenus automatiquement par le système de gestion des habilitations.

1.3) Les fonctions de sécurité associées à la gestion des identités

L'accès à des infrastructures distribuées par l'intermédiaire des réseaux locaux ou étendus, filaires ou sans-fil, doit passer nécessairement par une authentification des équipements, des services et des utilisateurs. Au cœur de ces infrastructures, les fonctions de sécurité associées à la gestion et au contrôle d'identité sont devenues primordiales pas uniquement pour des raisons sécuritaires mais aussi organisationnelles, car elles imposent aux organisations de mettre en place une nouvelle architecture du système d'information capable de répondre aux exigences de flexibilité des architectures orientées service (SOA) ou Cloud Computing. Il existe plusieurs fonctions de sécurité pour gérer l'identité numérique. Elles

englobent particulièrement les trois phases suivantes qu'on appelle le protocole AAA : Authentification, Autorisation, Journalisation ou Comptabilisation (Accounting en Anglais).

1.3.1) Identification et Authentification :

Les deux fonctions identification et authentification sont intimement liées dans les infrastructures informatiques et réseaux.

Identification : C'est ce qui nous permet de nous différencier des autres. Cette fonction d'identification d'un individu ne suffit pas à prouver qu'il s'agit bien de la personne qui prétend l'être.

Authentification : C'est une fonction de sécurité qui consiste à apporter la preuve de l'identité d'une personne, d'un service ou d'un équipement. Un système d'authentification permet de vérifier l'identité d'une personne en lui demandant une preuve tangible de son identité puis en validant ou en invalidant cette preuve. Le but de cette fonction est de certifier la véracité de l'identification.

Pour bien distinguer entre ces deux fonctions, nous prenons l'exemple d'un retrait et d'un dépôt d'espèce dans une banque. Quand une personne veut faire un virement d'argent en espèce sur son compte bancaire, le banquier ne lui demande que le numéro de son compte (i.e. c'est son identifiant), puisqu'il sait qu'elle n'a aucun intérêt à déposer de l'argent sur un autre compte. Par contre lorsqu'elle veut retirer de l'argent, il lui demande une pièce d'identité pour vérifier qu'il s'agit bien de cette personne. Cette vérification représente la fonction d'authentification.

Cette authentification sera réputée forte si elle rend difficile par exemple l'usurpation d'identité d'un utilisateur. Le principe de cette authentification forte consiste donc à combiner plusieurs facteurs d'authentification. Généralement, ces facteurs d'authentification sont basés sur un ou plusieurs mécanismes de reconnaissance :

- ✓ Ce que l'objet connaît : code PIN, un mot de passe, clé symétrique, etc.
- ✓ Ce que l'objet possède : module physique ou jeton d'authentification (clé USB, RFID, carte à puce, etc.).
- ✓ Ce que l'objet prétend être : une empreinte digitale, ou tout autre élément biométrique.

L'objectif est de pallier les faiblesses de l'authentification par mot de passe. Elle est considérée comme une des fondations principales pour garantir l'autorisation et le contrôle des accès et les propriétés de la sécurité : confidentialité, intégrité, traçabilité, etc.

En raison du coût très élevé, son usage reste aujourd'hui très limité aux grands comptes ou, plus généralement, aux secteurs critiques des services et de l'industrie : défense, aéronautique, automobile, recherche scientifique, banque, énergie, etc. L'authentification forte peut s'avérer donc une bonne solution pour s'ouvrir à l'extérieur.

1.3.2) Autorisation ou habilitation :

Autorisation : Quels sont mes habilitations ou mes droits pour invoquer ou activer une ressource dans un système d'information ? Cette fonction est totalement transparente pour l'utilisateur, car après son authentification dans un système, cette fonction vérifie ses habilitations pour lui permettre l'accès à la ressource invoquée.

La consommation d'un service est réalisée en deux phases, qui englobent les fonctions d'identification et d'authentification, et ensuite la fonction de contrôle des accès. La première phase (d'identification et d'authentification) est la partie visible pour l'utilisateur au moment où il souhaite consommer un service. Ce service va s'appuyer sur la fonction d'autorisation pour vérifier les droits d'accès de l'utilisateur et lui fournir les accès requis.

L'autorisation est un processus qui permet d'établir les habilitations dont dispose une identité au travers d'un rôle qui lui est associée ou des groupes auxquels elle appartient. Elle détermine donc les droits d'accès pour chaque ressource et repose sur la fonction d'authentification pour garantir la confidentialité et l'intégrité des données qu'elle doit protéger.

Elle se base sur des mécanismes de contrôle des accès en fonction des attributs d'une identité. Le modèle Role-Based Access Control (RBAC), peut être défini pour factoriser les droits par rapport aux applications. Aujourd'hui, la gestion des habilitations est beaucoup plus avancée avec l'utilisation du modèle Organization-Based Access Control (OrBAC) qui prévoit des couches d'abstraction supplémentaires pour faciliter l'affectation des habilitations.

Il existe d'autres modèles comme DAC (Discretionary Access Control) ou MAC (Mandatory Access Control). Ces deux systèmes ne sont pas adaptés à la gestion des droits dans de grandes organisations. Le système DAC est utilisé dans la gestion des droits du

système de fichier Unix où le propriétaire d'une ressource définit lui-même les droits sur celle-ci. Le modèle MAC se base sur une politique de sécurité qui calcule les droits d'accès sous-jacents. Il permet de limiter les accès à une ressource en fonction de la sensibilité des informations, lesquelles sont hiérarchisées, sous forme de labels, en différents niveaux de sécurité MLS (Multi-Level Security). Ce modèle facilite la gestion des habilitations en se basant sur un tiers.

1.3.3) Accounting ou Comptabilisation :

Accounting : Qu'est-ce que j'ai fait, quand et comment ? Selon la norme ISO 8402 : *"c'est l'aptitude à retrouver l'historique, l'utilisation ou la localisation d'un produit ou d'un processus de délivrance d'un service au moyen d'identifications enregistrées".*

Le terme "*Accounting*" peut être traduit par traçabilité dans ce contexte. C'est une fonction qui consiste à archiver l'histoire des transactions d'une entité. Cette fonction est importante pour auditer un système, ou pour déterminer l'imputabilité. Cette traçabilité permet de disposer à chaque instant, au moyen d'événements et d'informations enregistrés, de connaître l'historique d'utilisation d'une ressource. Elle sert aussi à améliorer la sécurité et les performances d'un système, à détecter des anomalies, ou à créer des indicateurs décisionnels. Elle peut agir de manière préventive et proactive pour prévenir les risques internes et externes (imputabilité, falsification, détournement, etc.).

Chez les opérateurs de télécommunication, cette fonction est utilisée pour comptabiliser et facturer toutes les transactions effectuées sur chaque ressource (équipement matériel, logiciel, utilisateur, etc.).

1.4) Composants et exigences d'un système de gestion des identités

Pour gérer efficacement le cycle de vie d'un système de gestion des identités, les composants suivants sont d'une utilité importante pour les fonctions précédentes et pour garantir plus de flexibilité et de performance pour les utilisateurs sans affecter la sécurité de leurs informations personnelles [4].

1.4.1) Le référentiel d'identité :

Il doit inclure, les annuaires, méta-annuaires et les bases de données, qui se composent des identités, des attributs des utilisateurs, et de l'ensemble des services d'une organisation habilité à accéder aux ressources (logicielles ou matérielles).

1.4.2) Le Provisioning / Deprovisioning :

Ce composant "Provisioning" permet d'alimenter le référentiel d'identité et de l'approvisionner avec des informations utilisateurs, de le synchroniser avec des annuaires ou des bases de données, de gérer les utilisateurs à l'aide d'un système de Workflow pour faciliter la délégation et la décentralisation de l'administration des habilitations. Le composant "Deprovisioning", joue un rôle très important dans des infrastructures décentralisées, car il doit propager en temps réel, sur les annuaires et les bases de données, toute révocation d'un utilisateur ou toute restriction de ses habilitations ou de ses rôles dans un système.

1.4.3) Le Workflow et Self-service (ou auto dépannage) :

La gestion du Workflow et du Self-service offre la possibilité de déléguer l'administration des habilitations, de contrôler, d'approuver de nouvelles fonctionnalités et d'implémenter de nouvelles politiques de sécurité d'accès de manière autonome et dynamique. En effet, les utilisateurs peuvent demander les habilitations et/ou les autorisations d'accès dont ils ont besoin à un instant donné. Ces demandes vont être approuvées par le manager ou les administrateurs systèmes si elles sont jugées légitimes. La fonction self-service permet donc aux utilisateurs habilités de réinitialiser leur mot de passe pour débloquer une ressource physique (e.g. code PIN d'une carte à puce) et/ou de se connecter en cas d'oubli avec un autre mot de passe temporaire. Cette fonction apporte donc, dans les infrastructures décentralisées, de l'agilité, de la dynamicité et de l'élasticité dans l'administration des ressources.

1.4.4) La *délégation des droits d'accès et révocation d'identité :*

Ce composant permet de fournir à l'utilisateur le moyen de gérer les informations, contenues dans leur identité et le moyen de les révoquer, en particulier dans le modèle de gestion des identités centré sur l'utilisateur. Il permet aussi de responsabiliser les managers et les collaborateurs en les plaçant au cœur du processus du cycle de vie de la gestion des

identités pour rendre la gestion des autorisations et l'administration des droits d'accès flexibles et dynamiques.

1.4.5) L'interopérabilité :

Cette fonction peut servir dans les modèles de gestion des identités fédérés pour faciliter le transfert des authentifications et des habilitations, d'une manière transparente, entre deux entités. En effet, les identités des utilisateurs devraient être représentées dans un format unique afin de permettre l'interopérabilité et la validation mutuelle dans des architectures à multiples domaines administratifs.

1.4.6) La gestion de confiance :

Il faut instaurer des relations de confiance entre les partenaires des différents domaines administratifs, permettant ainsi l'authentification mutuelle entre domaines. Il est également nécessaire de prévoir un moyen d'indiquer le niveau de confiance associé à chaque relation de confiance, ce qui aura une influence sur le comportement des fournisseurs de services.

1.4.7) L'Authentification unique et la fédération :

Cette approche d'authentification unique ou Single-Sign On (SSO), permet à un utilisateur de ne procéder qu'à une seule authentification pour accéder à plusieurs services dans un même domaine administratif. Ce moyen unique d'authentification permet de simplifier le nombre d'identifiants et par conséquent le nombre de mots de passe utilisé pour consommer un service. La fédération d'identité améliore la sécurité et la flexibilité d'une organisation, en se basant sur le SSO. Elle permet la propagation, le transfert de l'identification et de l'authentification dans une infrastructure transorganisationnelle ou dans plusieurs domaines administratifs de confiance. Ainsi elle permet de les regrouper dans une même stratégie de sécurité. Plusieurs approches de gestion des identités existent. Elles seront abordées, en détail, dans le chapitre III.

1.4.8) La vie privée et la protection des informations personnelles :

Comme nous l'avons évoqué précédemment, l'identité numérique est plus difficile à appréhender sur Internet. Tous les jours, les internautes laissent des traces pendant leurs interactions avec le cyberespace, sans se rendre compte de la quantité ou de la qualité des

données personnelles laissées, volontairement ou involontairement, sur les sites Web, et qui sont susceptibles de porter préjudice à leur vie privée. Certains d'entre eux ne se sentent même pas concernés.

Les données personnelles des internautes sont devenues très vulnérables : développement des réseaux sociaux, e-commerce, spams publicitaires. À l'égard de ce fléau, le débat sur la protection de la vie privée ne fait que commencer et certains usagers d'Internet s'intéressent à ce sujet.

La protection de la vie privée est un droit fondamental reconnu par la Convention Européenne de Sauvegarde des Droits de l'Homme et des Libertés Fondamentales (CESDHLF) et dans plusieurs traités internationaux. Elle est réglementée par le conseil européen, ainsi que la protection des données personnelles. C'est un réel défi dans le contexte du Cloud Computing et un concept difficile à saisir puisqu'il dépend du lieu géographique.

Par conséquent, les organisations doivent offrir des moyens et des mécanismes pour contrôler les préférences de confidentialité et les informations personnelles contenues ou associées aux identités des utilisateurs.

1.4.9) L'anonymat et les pseudonymes :

L'anonymat permet à un utilisateur de ne pas divulguer son identité réelle, tout en apportant la preuve qu'il s'agit bien de lui et qu'il possède les accréditations d'accès à un système. Le recueil des données ne doit en aucun cas permettre de façon directe ou via recoupement de remonter aux personnes. Par exemple lors d'un e-sondage, qui consiste à récolter des informations confidentielles et personnelles (opinions politiques, religieuses, étude épidémiologique, etc.), les utilisateurs devraient avoir la garantie de garder l'anonymat. Ainsi, tous les attributs associés à une identité numérique impliqués dans un processus d'authentification ne devraient pas révéler l'identité d'une personne.

L'utilisation de pseudonymes est un des moyens de garantir l'anonymat, et de mieux préserver la vie privée. Un pseudonyme est l'association d'un identifiant à un alias ou à un nom d'emprunt. Il peut être réversible dans le cas où on peut retrouver l'identifiant réel.

Certaines approches, dans des systèmes de gestion des identités fédérées, utilisent des pseudonymes permanents ou dynamiques pour préserver la vie privée des utilisateurs et masquer leurs attributs.

1.5) Critères de sécurité d'un système de gestion des identités

Le système d'information est un patrimoine important très convoité par la concurrence ou par des personnes malveillantes. Il convient donc de le préserver et d'en maîtriser les enjeux. Les critères de sécurité informatique sont des moyens d'évaluation, qui consistent à garantir l'interaction et l'interopérabilité entre les ressources dans un cadre prédéfini. Les critères de base sont la Disponibilité, l'Intégrité et la Confidentialité (DIC) :

- **La confidentialité** : Il s'agit de mettre en place des mécanismes pour ne pas divulguer un message d'un utilisateur, et donc le rendre inaccessible à tout attaquant ou espion potentiel. Seul l'auteur du message et les destinataires sont capables d'accéder à son contenu. La confidentialité est rendue possible par les mécanismes cryptographiques.
- **L'intégrité** : Il s'agit de la garantie qu'un message n'a pas été altéré, de façon fortuite, illicite ou malveillante, au cours d'une transaction, entre le moment où il a été émis et le moment où il a été reçu. Pour assurer cette fonction, il existe des fonctions de hachage à sens unique qui permettent de calculer un haché d'une donnée (MD-X, SHA-X, RIPEMD-X).
- **La disponibilité** : L'objectif est de garantir l'accès à un service ou à une ressource à un instant précis, avec des temps de réponse attendus et de maintenir le bon fonctionnement de l'ensemble des services en production.

Il existe d'autres critères (Figure 4) qu'il faut prendre en considération. L'imputabilité et la non-répudiation en particulier, trouvent leur importance dans un système de gestion des identités :

- **La non-répudiation** est l'impossibilité de prétendre ne pas être à l'origine d'une interaction avec un système d'information ou l'auteur d'un message. Elle permet de certifier le véritable auteur d'une modification ou une interaction avec un système de communication. Ce critère est assuré par des mécanismes cryptographiques asymétriques.
- **L'imputabilité** permet d'attribuer à une personne ou une entité une action qui a été commise. Ce critère de sécurité est lié à la non-répudiation. Elle peut être réalisée par un ensemble de mesures et d'enregistrements d'informations pertinentes et fiables pour identifier parfaitement une action dommageable à une entité.

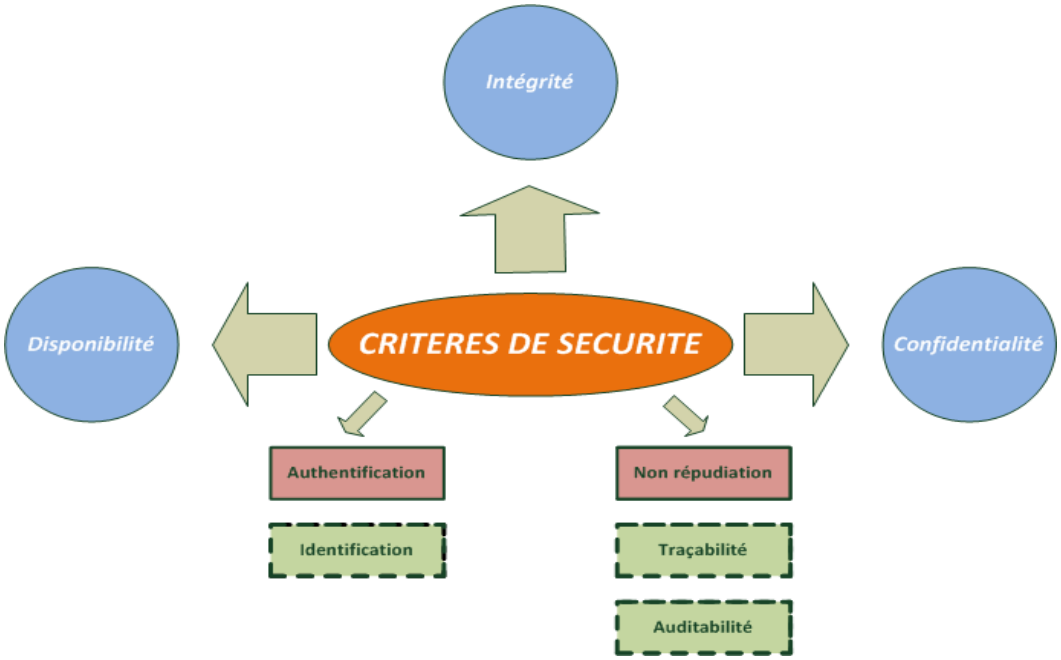

Figure 4 : Critères de sécurité.

1.6) Niveaux d'assurance et de confiance

Le NIST (National Institute of Standards and Technology) a publié un guide [40] sur le processus d'authentification électronique qui définit des niveaux d'assurance (LoA : Levels of Assurance) et des niveaux de confiance (LoT : Levels of Trust). Pour le LoA, il existe quatre niveaux d'assurance : le niveau 1 est considéré comme le plus faible et le niveau 4 le plus robuste (Figure 5).

Level	Description
1	Little or no confidence in the asserted identity's validity.
2	Some confidence in the asserted identity's validity.
3	High confidence in the asserted identity's validity.
4	Very high confidence in the asserted identity's validity.

Figure 5 : Description des quatre niveaux d'assurance.

Ce guide indique les exigences techniques pour chaque niveau. La Figure 6-A montre les types de jetons qui peuvent être utilisés à chaque niveau d'assurance de l'authentification et la Figure 6-B illustre les types de protocoles d'authentification qui sont applicables à chaque niveau d'assurance.

Un processus d'authentification qui utilise le nom de l'utilisateur et le mot de passe est considéré comme moins robuste qu'un processus qui utilise des certificats numériques avec des clés publiques. Dans ce même guide, le NIST mentionne aussi l'importance d'un processus d'enregistrement des utilisateurs dans un annuaire.

Par exemple, un système qui permet aux utilisateurs de créer leurs comptes à travers des formulaires Web, est classé dans un niveau d'assurance faible, puisque l'utilisation de cette procédure, ne permet pas à un fournisseur d'identité de garantir l'identité réelle de l'utilisateur. Par contre un processus d'enregistrement qui exige la présence physique d'une personne avec la présentation d'un document personnel : pièce d'identité, permis de conduire, etc. auront un niveau d'assurance plus élevé.

A

Token type	Level 1	Level 2	Level 3	Level 4
Hard crypto token	√	√	√	√
One-time password device	√	√	√	
Soft crypto token	√	√	√	
Passwords & PINs	√	√		

B

Protocol Type	Level 1	Level 2	Level 3	Level 4
Private key PoP*	√	√	√	√
Symmetric key PoP*	√	√	√	√
Tunneled or Zero knowledge password	√	√		
Challenge-response password	√			

*PoP : Proof of Possession

Figure 6 : LoA alloué à chaque type de jeton et types de protocoles d'authentification

L'utilisation du LoA permet de fournir différents niveaux d'authentification et d'autorisation [5]. Par exemple, le niveau 1 ne permet aux utilisateurs authentifiés qu'un accès en lecture à une ressource et ceux authentifiés avec un niveau 4 peuvent lire et écrire. Nous pouvons aussi utiliser les niveaux LoT pour garantir les différents niveaux de confiance dans une infrastructure transorganisationnelle.

1.7) Les risques liés à la faiblesse des mots de passe :

Depuis le début de l'informatique, les mots de passe représentent la solution la plus répandue et la plus simple à implémenter, bien qu'elle procure un minimum de sécurité. De nos jours, le mot de passe sert encore de bouclier dans des applications Web, des systèmes

distribués, ou embarqués. Pourtant, la faiblesse d'un mot de passe représente un risque réel pour un système d'information. De plus l'utilisateur est, généralement mal sensibilisé, ou n'a pas pris conscience des risques liés à cette faiblesse du mot de passe. Par conséquent, c'est l'utilisateur qui constitue le maillon faible de la chaîne dans un système d'information, il prend de mauvaises habitudes en donnant à ses comptes des mots de passe par défaut, ou triviaux, et même des fois pas de mot de passe du tout. Malheureusement, l'instauration d'une politique de sécurité complexe ou des règles trop contraignantes ne permettra pas de garantir la sécurité, car elle pourrait avoir pour conséquence d'encourager les utilisateurs à noter leurs mots de passe par exemple sur un pense-bête. Pour minimiser les risques, il faut sensibiliser l'utilisateur à sélectionner ses mots de passe avec soin.

Techniquement, les mots de passe forts sont l'une des briques de base dans une politique de sécurité. La force d'un mot de passe dépend de sa longueur et de la mixité du nombre de caractères qui le composent. En effet, un mot de passe constitué de caractères alphanumériques et spéciaux est plus difficile à retrouver qu'un mot de passe constitué uniquement de caractères minuscules ou majuscules.

Théoriquement, la complexité d'une attaque par force brute est une fonction exponentielle de la longueur du mot de passe : un mot de passe constitué de 4 nombres en base 10, correspond à (10^4) possibilités alors qu'un mot de passe composé de 4 caractères en base 26, correspond à (26^4) possibilités. Un mot de passe qui combine des chiffres et des caractères, voire même des majuscules et des caractères spéciaux sera encore plus difficile à casser. Malheureusement, dans la pratique, l'utilisateur final peine toujours à adopter ces bonnes pratiques pour palier les faiblesses et la vulnérabilité des mots de passe.

1.8) Typologie des attaques

Une attaque consiste essentiellement à déjouer les mécanismes de sécurité d'un système d'information et à identifier les failles techniques et organisationnelles. La Figure 7 illustre les différents types d'attaques : passives et actives.

Les attaques qui modifient les données sont dites actives (Figure 8, schémas b, c, d et e), tandis que celles relevant de l'écoute (interception sans altération des données) sont qualifiées de passives (Figure 8, schéma a).

La première motivation des attaques est le gain financier en accédant au compte bancaire d'une victime. Les menaces et les techniques liées au vol d'identité numérique sont nombreuses (Annexe A).

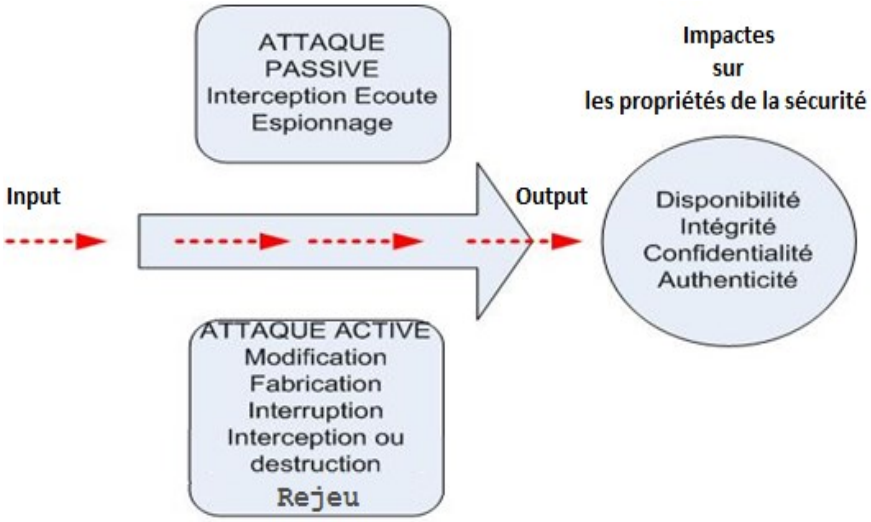

Figure 7 : Attaques passives et actives

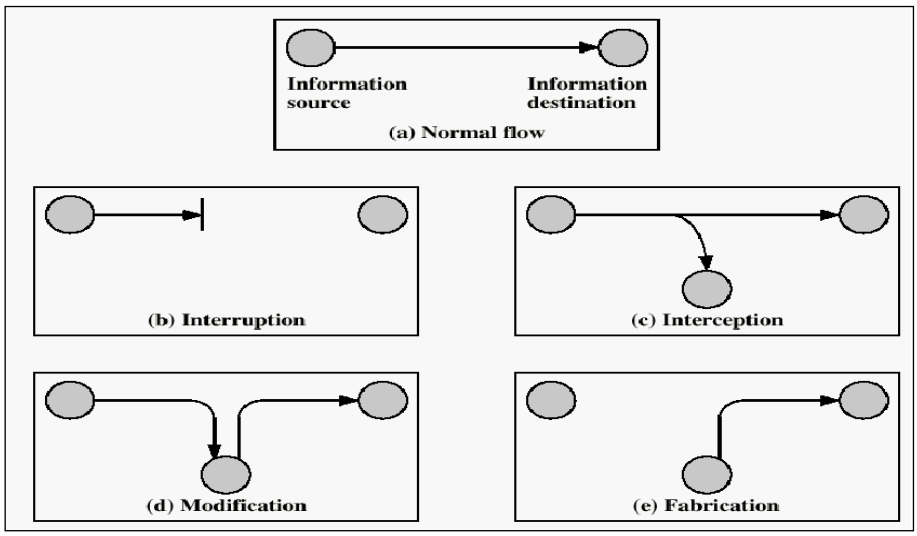

Figure 8 : Schéma des attaques passives (a) et actives (b, c, d et e).

1.9) Les défauts de conception des applications :

Nous donnons à titre d'exemple, la faille de sécurité " *Heartbleed*" [51], qui signifie saignement du cœur et qui a touché OpenSSL, un des boucliers de défense des principales infrastructures Web. C'est une vulnérabilité logicielle majeure trouvée et "introduite par erreur depuis mars 2012", dans le code source de la bibliothèque cryptographie d'OpenSSL. Elle permet à des pirates, de lire la mémoire d'un serveur ou d'un client, d'accéder aux

informations personnelles (jusqu'à 64 Ko de données) et de récupérer, par exemple, les clés privées ou les mots de passe lors d'une communication TLS (Transport Layer Security) [86].

Cette erreur est survenue au cours d'une correction d'un bug et d'améliorations de fonctionnalités d'OpenSSL dans un protocole "*Heartbeat*" (ce qui signifie battement de cœur). Le protocole "*Heartbeat*" qui est une extension décrite dans la RFC 6520, fonctionne de manière analogue, à la fonctionnalité de type "*Keepalive*", qui permet de maintenir une communication ouverte entre deux systèmes, malgré l'arrêt des échanges de données. Cet échange client/serveur sous la forme d'un envoi et d'une réponse rappelant le battement du cœur. La faille survient dans l'implémentation de cet échange. Un développeur aurait oublié d'initialiser une variable. En effet un client authentifié pouvait demander à un serveur TLS, de lui renvoyer non seulement une simple validation de sa présence, mais aussi un message allant jusqu'à 64 000 caractères.

Cette vulnérabilité a été découverte et rendue public en avril 2014, par les équipes de recherche en sécurité de Google et par des ingénieurs de la société Codenomicon (Finlande). Au moment de la découverte du bogue, 17 % des serveurs Web "sécurisés" ont été touchés par cette faille, soit environ un demi-million de serveurs.

Cette vulnérabilité est riche d'enseignements, car elle met en exergue que le risque zéro n'existe pas, et que chaque mécanisme de sécurité possède des failles sous-jacentes après sa conception et son implémentation. Elle montre aussi que le stockage des clés cryptographiques dans le serveur constitue une vraie vulnérabilité.

1.10) Conclusion

Dans ce chapitre nous avons mis en évidence la dépendance des processus métiers d'une entreprise à leur système d'information, et l'importance de protéger leurs informations. L'information est donc un actif de valeur qu'il faut protéger car il est à la fois indispensable et convoité. Elle est soumise à diverses menaces. L'exploitation des vulnérabilités du système d'information d'une entreprise fait peser un risque imminent sur ce patrimoine vital.

Nous avons également montré que les fonctions de sécurité associées à la gestion de l'identité peuvent donc être un rempart dissuasif permettant d'assurer la réduction des risques à un niveau tolérable et l'impact sur les propriétés de sécurité. En particulier, l'authentification et l'autorisation qu'il ne faut donc pas négliger. La garantie de la fonction d'autorisation

adaptée au Cloud Computing doit passer bien évidemment par l'implémentation d'une bonne méthode d'authentification.

Cette analyse a également permis de mettre en exergue les composants d'un système de gestion d'identité à mettre en avant ou à écarter pour préserver les critères de sécurité et dans la perspective d'élaborer des infrastructures de gestion d'identité flexible et dynamique.

Il faut aussi sensibiliser les concepteurs et les développeurs d'applications à l'importance d'une politique complète et cohérente de validation des processus de développement, ainsi qu'au stockage des clés sensibles.

Ces infrastructures doivent être évaluées selon différents niveaux LoA ou LoT pour garantir les exigences de sécurité et les différents niveaux d'authentification et de confiance dans une infrastructure transorganisationnelle. Cela implique de mettre en place des mécanismes et des protocoles d'authentification robustes comme indiqués dans le guide du NIST [40].

Par conséquent, l'adaptation d'un système de gestion des identités doit être conforme à des standards et à des protocoles d'authentification reconnus. Les performances offertes en termes de sécurité par certains protocoles (RADIUS, EAP, 802.1X, TLS, etc.) nous amènent à les prendre en considération dans notre rapport. C'est ce que nous allons étudier dans les Chapitres suivants II et III.

Chapitre II : Étude de quelques standards AAA et protocoles d'authentification.

2.1) Introduction :

Le chapitre précédent montre que l'exploitation d'une vulnérabilité du système de gestion des identités fait peser un risque sur le processus métier d'une organisation, et des enjeux considérables sur leur système d'information, à la hauteur des habilitations allouées aux utilisateurs et des conséquences lourdes en cas d'usurpation d'identité.

Pour répondre aux exigences de sécurité, les contre-mesures à implémenter dans un système de gestion des identités consistent à mettre en place des mécanismes d'authentification robustes. Cela implique de mettre en place des protocoles standards AAA (§2.2) reconnus et des méthodes d'authentification forte (§2.3). Dans ce contexte, les outils cryptographiques avec l'utilisation des clés asymétriques s'imposent comme un compromis idéal puisqu'ils nous offrent différents mécanismes et protocoles d'authentification, basés sur des standards et reposent sur des modèles reconnus.

Nous allons à présent examiner les composants clés de quelques standards AAA et leurs imbrications avec les protocoles d'authentification associés. Nous essayerons de dégager les avantages et les inconvénients de ces protocoles pour conclure (§2.4) ce chapitre, avec quelques considérations à prendre en compte dans l'élaboration de notre système de gestion des identités.

2.2) Étude de quelques protocoles implémentant la triade AAA :

Les protocoles de la triade AAA sont plébiscités par les opérateurs offrant des services de télécommunications à des utilisateurs. Ils peuvent ainsi facturer selon le temps de connexion ou selon la quantité d'information téléchargée. Plusieurs solutions existent sur le marché (propriétaires ou libres) supportant le protocole AAA :

- TACACS+ (Terminal Access Controller Access-Control System Plus) [93] de CISCO ;
- IAS (Internet Authentication Service de Microsoft sous Windows) [52] ;
- ACS (Access Control Server de CISCO sous Windows);

- OpenRadius (Linux);
- FreeRadius (Linux, BSD, Solaris, Windows, etc.).

Nous allons dans un premier temps décrire sommairement le successeur de RADIUS : DIAMETER (§2.2.1). Ensuite, nous étudierons en détail RADIUS (§2.2.2) qui présente plusieurs avantages, notamment la diversité des méthodes d'authentification (§2.2.3).

2.2.1) DIAMETER :

DIAMETER est un nouveau Framework défini par l'IETF (Internet Engineering Task Force) pour les serveurs AAA de nouvelle génération. Ce n'est pas une extension de RADIUS mais son successeur. Il a été développé pour améliorer plusieurs limitations de ce dernier. Il s'agit du protocole le plus à même de répondre aux nouveaux besoins de mobilité. En particulier, il permet à des domaines administratifs différents, en particulier les opérateurs télécoms, de collaborer pour réaliser les fonctionnalités AAA, dans des architectures LTE (Long-Term-Evolution of 3G) et IMS (IP-Multimedia-Subsystem).

DIAMETER s'appuie sur la couche transport (TCP). Il est compatible avec les protocoles EAP et RADIUS. Il est constitué d'un protocole de base (RFC 3588) qui définit le format des messages, et comment sont transportés les messages d'erreur, que toutes les implémentations doivent supporter, ainsi que les services de sécurité. Il a été conçu dans l'idée d'être facilement extensible. Pour cette raison, le protocole de base est séparé de ses applications (Mobile IPv4, NAS, CMS, etc.) :

- Mobile IPv4 (RFC 3344) [87] est un protocole qui permet à un mobile d'être joignable (par d'autres mobiles ou terminaux fixes) indépendamment de l'opérateur de service et de sa position géographique. L'extension Mobile IP de DIAMETER (RFC 4004) s'avère particulièrement intéressante pour les réseaux cellulaires 3G ; elle permet, à des domaines administratifs différents, d'authentifier et d'autoriser des mobiles à se connecter et à prendre en charge les aspects de comptabilisation (RFC 2977).
- NAS (RFC 4005 - DIAMETER Network Access Server) permet l'accès au réseau via PPP/EAP, il s'agit d'une amélioration de RADIUS ;
- CMS (Cryptographic Message Syntax) [88] permet de protéger les échanges DIAMETER au niveau applicatif entre serveurs ou entre un serveur et son client. Il permet d'encoder le message du client au serveur DIAMETER. Pour cela, CMS va

utiliser deux techniques : la signature numérique qui apporte la fonction d'authentification et d'intégrité et la cryptographie qui permet la confidentialité du message.

Ce protocole est utilisé par exemple lors des échanges entre les serveurs d'application et le serveur HSS (Home Subscriber Server) dans IMS (IP Multimedia Subsystem) [67], pour les transferts d'information des profils des utilisateurs. Les profils des utilisateurs sont stockés dans des annuaires (référentiels) ou des bases de données par les serveurs RADIUS et DIAMETER. Ces référentiels s'interfacent avec d'autres pour s'abstraire de la multitude d'identifiant et de cette complexité évoquée dans les paragraphes précédents.

2.2.2) RADIUS :

Le protocole RADIUS (Remote Authentication Dial-In User Service), qui a été mis au point par Livingston Enterprises Inc Lucent, et normalisé par l'IETF (Internet Engineering Task Force), est défini dans plusieurs RFC :

- RFC 2865 : protocole RADIUS, avec authentification et autorisation,
- RFC 2866 : rajoute l'Accounting,
- RFC 2869 : rajoute les méthodes EAP.

Dans des infrastructures d'authentification Client/serveur, il est considéré comme un standard AAA, et permet de centraliser et combiner les fonctions AAA. Il est utilisé par de nombreuses entreprises, notamment les fournisseurs d'accès aux réseaux de télécommunications, PPP ou VPN. Pour l'authentification, ce protocole s'appuie sur le protocole de transmission de données en mode non connecté et sans mécanisme de fiabilité de transmission (UDP), en utilisant le port 1812 et pour l'Accounting, le port 1813.

Aujourd'hui, ce protocole est aussi souvent utilisé pour la gestion des connexions à Internet, en particulier des réseaux Ethernet sans fil (WLAN), en se basant sur le protocole 802.1X et EAP. Il peut également, gérer l'identification et l'authentification des sites Web en rajoutant le module "*mod_auth_radius*" dans le serveur Apache. Ce module permet de valider une authentification en interrogeant le serveur Radius.

2.2.2.1) Les composants physiques de l'architecture RADIUS :

La Figure 9 illustre les composants principaux d'une architecture physique tripartie du standard RADIUS :

- **NAS** : équipement réseau appelé aussi client RADIUS, qui fait l'intermédiaire entre le serveur RADIUS et le terminal. Le NAS peut être une borne Wi-Fi ou un commutateur (Switch) qui implémente le protocole IEEE 802.1X et EAP, pour gérer les connexions et le transport des méthodes d'authentification.
- **Supplicant** : terminal de l'utilisateur,
- **RADIUS** : serveur d'authentification. Ce dernier est généralement relié à une base d'identification (fichier, base de données, annuaire, etc.). Dans certains cas, le serveur RADIUS peut jouer le rôle d'un mandataire (proxy).

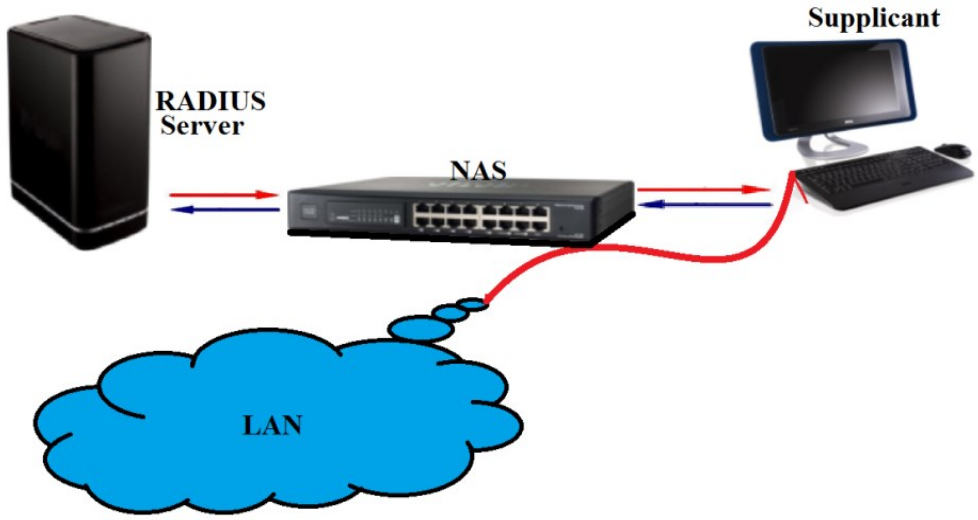

Figure 9 : Architecture physique du standard RADIUS.

2.2.2.2) Principe de fonctionnement :

L'architecture logique de RADIUS se base sur le standard 802.1x, mis en place par l'IEEE en juin 2001 et le protocole EAP (Extensible Authentication Protocol). Dans cette partie nous détaillerons les protocoles qui interviennent au cours d'une authentification réseau et le rôle de chacun :

- ✓ **IEEE 802.1X (Port-Based Network Access Control):**

Le standard 802.1X permet de valider une autorisation d'accès à un réseau après authentification d'un Supplicant, indépendamment du réseau physique utilisé. Le NAS qui implémente le protocole 802.1X autorise en cas de succès l'accès physique au service réseau.

En effet, les ports du NAS (Commutateur) sont configurés d'une façon logique. Avant d'être complètement ouverts, ils ne laissent passer qu'un seul type de trafic particulier vers le serveur. Il contrôle une ressource disponible au niveau du port physique réseau, nommé PAE (Port Access Entity). La Figure 10 illustre le port d'un commutateur qui est divisé en deux ports logiques :

- **En rouge :** le port non-contrôlé qui ne laisse passer que les messages d'authentification vers le serveur. Le NAS n'accepte que les messages EAP.
- **En noir :** le port contrôlé est maintenu ouvert tant que le serveur n'a pas accepté l'authentification d'un client.

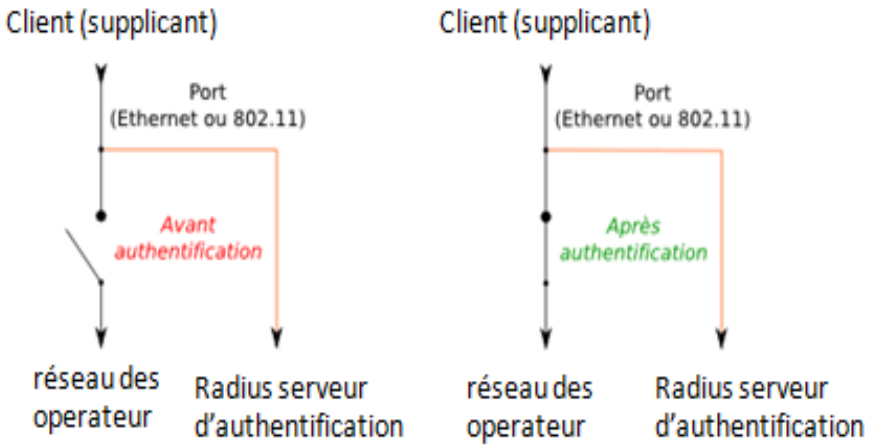

Figure 10 : *Schéma logique des ports 802.1X*

✓ ***EAP (Extensible Authentication Protocol)***

EAP est une extension du protocole PPP, normalisé (RFC 2284) par l'IETF. Les échanges du standard 802.1X reposent sur le protocole EAP qui encapsule ses paquets. EAP est un protocole de transport d'authentification. Il définit des mécanismes d'échanges entre les équipements, indépendamment des méthodes d'authentification. Il apporte ainsi plus de souplesse et de sécurité. C'est un protocole indépendant du protocole RADIUS. Il transporte

les informations d'identification et les attributs vers le serveur, au niveau de la couche 2 du modèle ISO, en utilisant deux méthodes pour encapsuler les paquets :

- ✓ **EAPoL ou EAPoWL** : EAP over LAN ou WLAN, transporte les paquets entre le Supplicant et le NAS,
- ✓ **EAPoRadius** : EAP over RADIUS, transporte les paquets entre le NAS et RADIUS.

La Figure 11 illustre la composition d'une trame EAP et les 4 champs de son en-tête :

- ✓ Code (1 octet) : spécifie, le type de message,
- ✓ Identifier (1 octet) : identifie le nombre de requêtes ou de réponses déjà envoyé,
- ✓ Length (2 octets) : donne la taille du paquet à envoyer ou recevoir,
- ✓ Type (1 octet) : désigne la méthode d'authentification utilisée ou des opérations particulières.

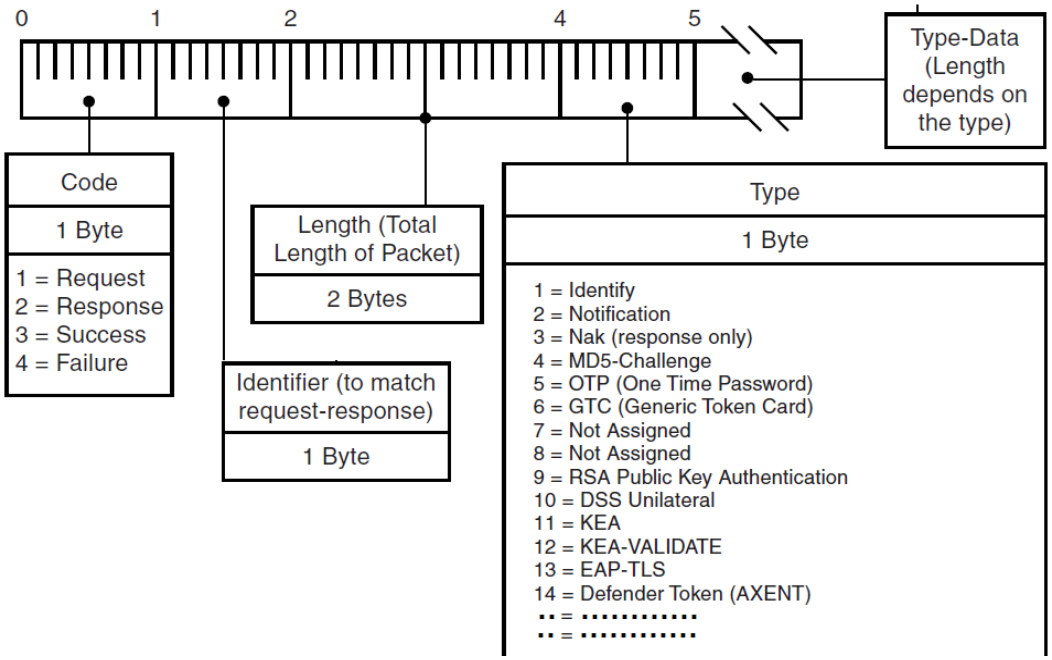

Figure 11 : Trame EAP requête/réponse

Le protocole EAP transporte plusieurs méthodes d'authentification (RFC 3748) qui sont représentées par le champ Type, codé sur un octet (soit 256 méthodes), dont c*ertaines seront abordées (§2.2.2.4) :*

- ✓ Type = 1, message relatif à l'identité (Identity)
- ✓ Type = 3, notification d'une erreur (NAK).
- ✓ Type = 4, protocole d'authentification à base de défi MD5 (EAP-MD5)

- ✓ Type = 13, transport de TLS (EAP-TLS)
- ✓ Type = 18 méthode d'authentification basée sur une carte SIM (EAP-SIM)
- ✓ Type = 26, transport de MSCHAPv2…

Initialement, il existait que deux méthodes d'authentification des utilisateurs, nommées PAP (Password Authentication Protocol) et CHAP (Challenge-Handshake Authentication Protocol).

2.2.2.3) Authentification dans une architecture RADIUS :

Le trafic réseau du protocole RADIUS est considéré comme très faible. Les différents paquets d'authentification sont de quatre types. Chaque paquet est véhiculé au moyen d'un paquet spécifique (Figure 12) :

- ✓ Access-Request : l'initiation d'une session commence par ce paquet émis par l'Authenticator (NAS) vers le serveur, lequel contient au moins l'attribut (User-Name) et une autre liste d'attributs (i.e. Calling-Station-Id, Nas-Identifier, etc.).
- ✓ Access-Challenge : Après réception d'un paquet (Access-Request), le serveur RADIUS peut éventuellement, envoyer un autre paquet (Access-Challenge) afin d'obtenir d'autres informations ou demander l'utilisation d'une méthode d'authentification au Supplicant.
- ✓ Access-Accept : Ce paquet est renvoyé, par le serveur RADIUS, à l'Authenticator en cas de succès de l'authentification, pour lui indiquer les autorisations accordées au Supplicant.
- ✓ Access-Reject : ce paquet est envoyé par le serveur, en cas d'échec de l'authentification, à l'Authenticator pour lui demander d'interdire l'accès au réseau à un Supplicant.

Schématiquement, après la requête du NAS, le Supplicant décline son identité et ses attributs d'authentification. Le NAS les reçoit par le port non contrôlé et les relaye au serveur RADIUS, lequel envoie un défi aléatoire (challenge) au Supplicant, pour calculer un HMAC-MD5 (RFC 2104), à partir de ce défi et du secret partagé avec le serveur. Ce hash est retransmis au serveur RADIUS. Celui-ci calcule son propre hash avec les attributs du client qu'il possède et compare les deux hash. Dès lors le NAS ne joue plus que le rôle de mandataire. Si le résultat de la comparaison est identique cela signifie que le Supplicant possède la bonne clé, ce qui se traduit par un succès de l'authentification. Dans ce cas, le

Supplicant et le serveur RADIUS calculent une clé (Unicast Key), laquelle est retransmise au NAS avec le message "Access-Accept". Suite à ce message, le port contrôlé passe à l'état autorisé. Ensuite, cette clé est dérivée par le Supplicant et le NAS en plusieurs clés, nécessaires au chiffrement des paquets échangés entre ces derniers. Dans le cas contraire, l'authentification échoue et le serveur rejette la demande en envoyant le message "EAP-Reject". Par conséquent, le Supplicant n'est pas autorisé à accéder au réseau.

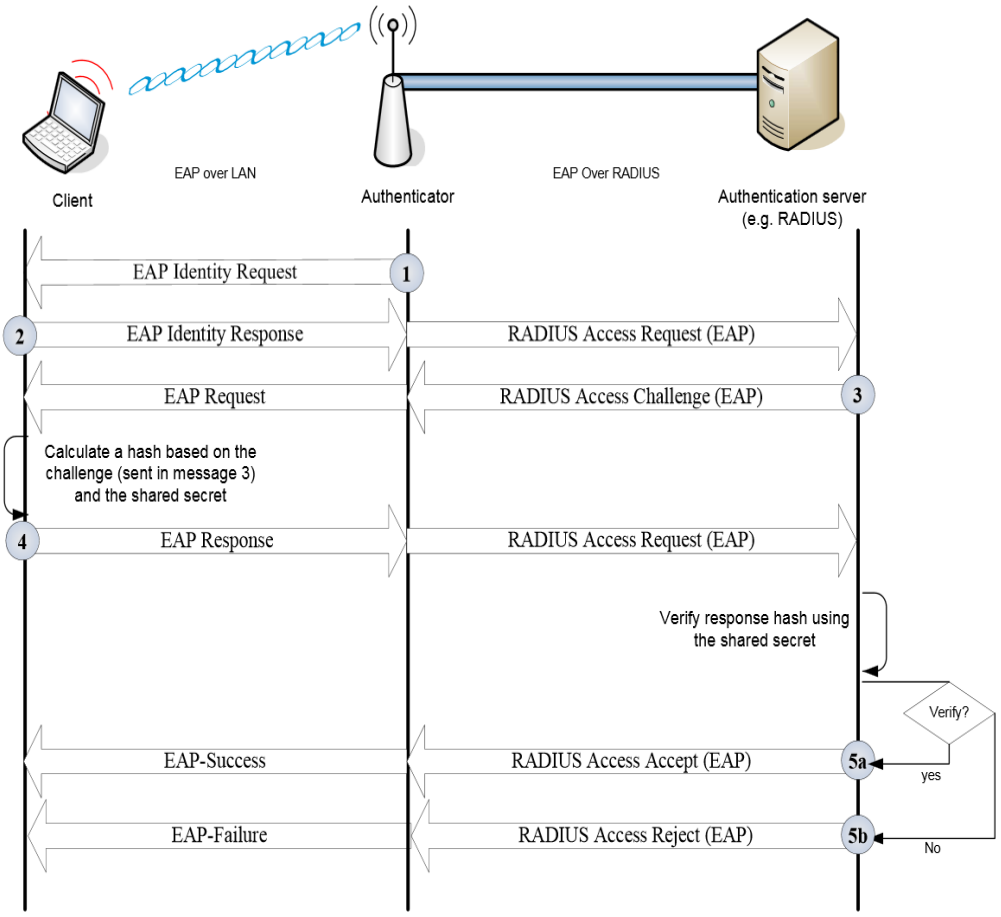

Figure 12 : Échanges de message EAP

Contrairement à DIAMETER, RADIUS est limité à un seul domaine administratif car il n'assure pas l'itinérance (Roaming) dans un contexte transorganisationnel.

2.2.2.4) Les méthodes d'authentification utilisées dans RADIUS

Comme évoqué dans le paragraphe précédent, les méthodes d'authentification standardisées sont nombreuses, et simple à mettre en œuvre : EAP-MD5, EAP-OTP (*One-Time-Password*) et EAP-GTC (*Generic-Token-Card*), qui ne proposent que l'authentification

unilatérale. La méthode EAP-MD5 [80] s'appuie sur le protocole CHAP [81], en utilisant le même principe "Challenge/Response" qui vise à authentifier le Supplicant. Elle nécessite un secret partagé entre le Supplicant et le serveur RADIUS. Ce secret réside généralement dans l'utilisation d'un mot de passe associé à un identifiant utilisateur (nom de l'utilisateur, adresse IP ou MAC, etc.). Cette méthode ne nécessite pas beaucoup de ressources pour sa mise en place, mais elle souffre des mêmes attaques que le protocole CHAP (§2.2.2.5). La méthode EAP-MD5 s'est révélée trop faible pour assurer une authentification de qualité satisfaisante, car la fonction MD5 est reconnue aussi comme vulnérable aux attaques de type dictionnaire, force brute. Généralement, MD5 est utilisé pour calculer des hachés au moment d'une signature électronique.

En outre, EAP-MD5 est une méthode d'authentification unilatérale dans le sens où il n'y a pas d'authentification mutuelle entre les deux parties. Il n'est donc pas possible de détecter de faux serveurs. Il est déconseillé d'utiliser des mécanismes d'authentification basés sur un secret partagé (e.g. WEP, WPA, etc.).

D'autres méthodes fonctionnent avec des cartes à puce spécifiquement utilisées dans les environnements GSM/UMTS, destinées aux opérateurs de télécommunication : EAP-SIM (RFC 4186) (Subscriber Identity Modules), et EAP-AKA (RFC 4187) (Authentication and Key Agreement). Certains opérateurs s'opposent fermement à l'intégration de ces cartes SIM (Embedded SIM) dans les téléphones, car derrière cet aspect technique, il y a des enjeux, en particulier la distribution des forfaits.

Actuellement, ce sont les opérateurs qui sont au cœur de la gestion des identités des utilisateurs : ils activent la carte SIM, lorsqu'un client s'abonne à leurs services, l'identifient, l'authentifient quand il accède à leur réseau téléphonique et lui facturent la communication. Du coup, si la carte SIM est directement intégrée ou embarquée sous forme de logiciel dans le téléphone, l'opérateur perd cette maîtrise précieuse sur le client. Ce dernier pourrait ainsi plus facilement passer d'un opérateur à un autre.

Il existe d'autres méthodes d'authentification, parmi elles, EAP-TLS (Transport Layer Security) [82], qui s'appuie sur le protocole TLS (RFC 2246), pour offrir un niveau de sécurité très élevé, lequel permet de réaliser des authentifications mutuelles fortes avec des outils cryptographiques robustes. Ce protocole répond à nos besoins en particulier l'authentification mutuelle, et mérite d'être examiné en détail (§2.3.2), d'autant plus que les spécifications de notre projet SecFuNet, exigent l'utilisation d'un microcontrôleur sécurisé

qui va incarner l'identité d'une ressource virtuelle ou d'un utilisateur. Ce dernier sera, à son tour, détaillé dans le Chapitre IV.

2.2.2.5) Les faiblesses de l'infrastructure RADIUS :

Le serveur RADIUS souffre de quelques failles de sécurité et des problèmes de disponibilité ou de timeout, en particulier lorsque les périphériques tentent de contacter le serveur. Ces problèmes sont généralement liés aux modes d'authentification utilisés qui se situent au-dessous du protocole RADIUS.

L'utilisation, par exemple de la méthode d'authentification EAP-MD5 qui repose sur la fonction MD5 comme primitive pour créer une empreinte à partir de l'attribut "*User-Password*". L'authentification repose seulement sur une empreinte calculée par la fonction MD5 à partir du secret partagé (S*hared Secret*) et sur la génération, par le NAS, d'un défi aléatoire (*Request-Authenticator*), d'une taille généralement de 16 octets et qui n'est même pas chiffré. Ceci facilite grandement les attaques. En effet, RADIUS autorise l'emploi du même secret partagé pour plusieurs clients. Ce secret peut être de longueur différente, mais il est plus sûr d'avoir un secret d'au moins 16 caractères et plus complexe. Dans certaines implémentations clientes, le nombre de caractères du secret partagé est limité parfois à 16 octets. Par conséquent, l'emplacement d'un serveur RADIUS dans une architecture réseau devrait être particulièrement bien choisi. La norme impose la mise en place des mesures de sécurité physique ou VPN pour protéger les échanges entre le Supplicant et le serveur.

Le mode d'authentification PAP (*Password Authentication Protocol*) ou CHAP (*Challenge Handshake Authentication Protocol*), peut compromettre la sécurité du système d'authentification [41] [42] :

Le protocole PAP est moins sécurisé puisqu'il utilise des mots de passe non chiffrés sur le réseau. L'accès à l'attribut "*User-Password*" entraînerait une usurpation d'identité, car ce protocole envoie cet attribut d'authentification sur le réseau sans chiffrement.

Le protocole CHAP utilise, lui, un mécanisme de type (C*hallenge/Response)*. Le mot de passe ne circule donc pas en clair sur le réseau. Par contre, l'interception à la fois de l'attribut "*Access-Request*" envoyé par le client et de l'attribut "*Request-Authenticator*", le défi aléatoire généré par le NAS, permet de trouver le secret en utilisant une attaque par force brute ou par dictionnaire, car certains attributs ("*Request-Authenticator*", "*Code Identificateur*" et "*Longueur*", etc.) ne sont pas chiffrés.

L'attaque par rejeu des réponses du serveur est possible si le client envoie des requêtes utilisant le même "*Request-Authenticator*" que dans la requête précédente interceptée avec sa réponse. Ceci est susceptible de se produire si l'implémentation du client n'utilise pas une valeur aléatoire comme recommandée dans la norme.

Une des faiblesses du protocole EAP est le déni de service (ou DOS : Denial of Service) [43]. En effet, à la fin de l'authentification, le Supplicant et le serveur RADIUS dérivent une clé appelée Pairwise Master Key (PMK) à partir du Master Key (MK) générée pendant une session TLS. Cette PMK est transmise, par le serveur RADIUS, au NAS.

Ensuite, pendant la phase de négociation de la clé Pairwise Transient Key (PTK) (ou le protocole 4-Way Handshake), le Supplicant et le NAS calculent cette clé en incluant en particulier la PMK. Les échanges entre le Supplicant et le NAS sont transportés par le protocole EAP dans des requêtes EAPOL-key. Si à ce stade de la session EAP, un attaquant envoie au Supplicant un message EAP-Failure, ce dernier ne pourra plus négocier la clé PTK en utilisant les paquets EAPoL-Key.

2.3) La méthode d'authentification EAP-TLS

La méthode d'authentification EAP-TLS s'appuie sur le protocole TLS. Ce protocole d'authentification est simple à implémenter, mais il souffre de plusieurs inconvénients qui le rendent difficilement exploitable dans les réseaux IP classiques [82], en particulier il repose sur une infrastructure de gestion de clé (PKI : Public Key Infrastructure) (§2.3.1).

TLS offre plusieurs possibilités pour assurer les services de sécurité (§2.3.2) avec différentes phases d'authentification. C'est un protocole qui fonctionne indépendamment des autres couches réseaux ou applicatives. Son imbrication est facile avec le protocole EAP (§2.3.3), pour assurer les fonctions AAA dans une architecture RADIUS.

2.3.1) Rappel sur les Infrastructures à clé publique et les certificats :

Une PKI (Public Key Infrastructure) est un système de gestion des clés publiques qui permet de gérer le cycle de vie d'un certificat numérique (création, gestion, distribution, révocation, etc.) et d'en assurer la fiabilité. Elle offre la fonction d'authentification et assure plusieurs critères de sécurité tels que la confidentialité, l'intégrité et la non-répudiation.

Une infrastructure à clé publique s'appuie sur une autorité de certification CA (Certification Authority) qui utilise des mécanismes cryptographiques pour signer et certifier des clés publiques lesquelles permettent de chiffrer et de signer des messages ou des données. Elle est constituée d'un ensemble de ressources (utilisateurs, matériels, logiciels, politique de sécurité, procédures, etc.).

Une CA a pour fonction de créer des certificats électroniques, conformes à la norme X509 sous forme d'un fichier. Ce fichier est signé par la clé privée du CA, et contient des informations d'identité de l'utilisateur, une clé publique, une empreinte calculée par une fonction de hachage (MD5 ou SHA-1), et une date de validité.

Une CA peut également se charger de publier ces certificats et de les révoquer sous forme d'une liste CRL (Certificate Revocation List). En revanche, la clé privée du propriétaire du certificat reste quant à elle secrète. Les certificats sont délivrés ensuite aux utilisateurs sous différents formats pour servir à les identifier, les authentifier ou comme preuve de non-répudiation. Généralement, la clé publique est utilisée pour chiffrer des données, ce chiffrement est dit asymétrique et ne pourra être déchiffré que par la clé privée qui lui est associée, alors que la clé privée sert pour signer une empreinte MAC (Message Authentication Code). Cette clé est très sensible et doit être placée en lieu sûr, dans des modules sécurisés (i.e. cartes à puce). Ces clés sensibles sont généralement stockées en local dans le système. Il existe plusieurs formats de fichier utilisés pour stocker un certificat :

- ✓ DER (Distinguished Encoding Rules) est un format PKCS#10 qui est en ASN.1, utilisé pour encoder des certificats X509.
- ✓ PEM (Privacy Enhanced Mail) est un fichier qui peut stocker toute une chaîne de certificats y compris la clé publique, la clé privée et les certificats racine. Le format PEM n'est que du DER transcrit en base64 auquel sont ajoutés des en-têtes en ASCII. La protection des clés privées est optionnelle : la clé privée pourrait être générée au format PEM sans chiffrement (i.e. cette option n'est pas recommandée) ou chiffrée en Triple-DES.
- ✓ PKCS#12 (Public-Key Cryptography Standards) est un des standards de cryptographie à clé publique, publiée par les Laboratoires RSA. Il définit un format de fichier pour stocker les certificats et les clés privées. Ce fichier est chiffré par un mot de passe.

2.3.2) SSL/TLS protocole d'authentification

Le protocole TLS, anciennement dénommé SSL (Secure Socket Layer), a été développé par Netscape pendant les années 1990. La dernière version TLSv1 (RFC 2246) est standardisée par l'IETF (Internet Engineering Task Force). Il s'agit d'un protocole de sécurisation des échanges, se situant entre la couche transport et la couche applicative, pour assurer la sécurité des transactions sur Internet, il a été intégré dans certains navigateurs depuis 1994. Il a vite été adopté par plusieurs communautés de recherche comme 3GPP et IETF-EAP-WG qui font de lui le protocole phare en matière de chiffrement/déchiffrement, d'identification/authentification et de négociation de clés symétriques dans les architectures de réseau. TLS fonctionne d'une manière indépendante par rapport aux autres couches réseaux ou applicatives. En effet, il a l'avantage d'utiliser un canal chiffré et sécurisé, sans se préoccuper des problèmes d'interopérabilité. Il est extensible car il permet d'intégrer de nouveaux algorithmes de chiffrement sans modifier la structure des sous composants du protocole. Ce qui explique en grande partie son succès, c'est qu'il permet d'offrir et d'assurer, à moindre coût, les services de sécurité suivants en se basant sur des certificats X509 :

• **L'authentification et la révocation** : permet l'authentification mutuelle et unilatérale entre deux ressources (l'authentification du client est optionnelle), basée sur des certificats X509, et permet la révocation d'un utilisateur en invalidant son certificat X509.

• **La confidentialité** : permet le chiffrement en utilisant des algorithmes de chiffrement par flots comme RC4 ou des algorithmes de chiffrement symétrique par blocs comme DES, 3DES, IDEA, FORTEZZA, ou AES.

• **L'intégrité** : en utilisation des fonctions de hachage (*MD5, SHA-1, SHA-2, etc*.) pour calculer des hachés (MAC).

• **La protection contre l'attaque par rejeu :** la contre-mesure consiste à utiliser deux numéros de séquence synchrones dans les messages chiffrés pendant une session TLS. En effet, pour chaque session, les deux extrémités (*Client/server*) possèdent deux variables d'état nommées (*Sequence Number*), codées sur 8 octets. Ces deux variables sont synchronisées à chaque échange (lecture ou écriture). Elles sont concaténées aux messages TLS, le résultat est chiffré et ensuite envoyé à l'autre extrémité.

2.3.2.1) Les phases de connexion du protocole TLS

Le protocole SSL/TLS est situé entre les deux couches (Transport et Application.) du modèle TCP/IP. Il se comporte comme une couche intermédiaire indépendante du protocole utilisé au niveau de la couche Application. Il est capable de sécuriser tous les services Internet (Web, SMTP, POP, TELNET, etc.). Cela signifie donc qu'il apporte une extension de sécurité au protocole TCP/IP sans devoir reprendre sa conception.

Une connexion TLS est subdivisée en deux phases : la première phase (Handshake) permet de négocier les outils cryptographiques qui vont être utilisés pour créer un canal sécurisé, la seconde phase (Application Data) commence juste après pour fiabiliser les échanges de données applicatives en utilisant les outils négociés dans la phase précédente.

Le protocole TLS est constitué de quatre sous-protocoles TLS (Figure 13) : Handshake, Record Layer, Change Cipher Spec (CCS) et Alert. Ces trois protocoles, ainsi que le CCS, sont encapsulés dans le sous-protocole Record Layer :

- ✓ **Handshake** permet d'authentifier le client et le serveur, de négocier les algorithmes de chiffrement, les clés cryptographiques (Cipher Suite), c'est-à-dire le choix des algorithmes cryptographiques (symétriques, fonctions de hachage, etc.) qui seront employés pour assurer le chiffrement des données après une connexion. Cette partie constitue la première phase de négociation. La deuxième phase commence juste après l'établissement de la connexion, pour laisser la place aux échanges de données applicatives (Application data protocol).
- ✓ **Application Data Protocol** contient des données transmises par le client ou par le serveur sous forme de messages de type *Application Data*. Ces messages sont fragmentés, chiffrés, éventuellement compressés, etc. par le protocole Record Layer, ensuite transmis vers l'autre extrémité. En cas de fragmentation d'un message, le premier fragment contiendra la longueur totale des données à transmettre ainsi qu'un premier fragment de ces données, et les paquets suivants contiendront les fragments suivants.
- ✓ **Record Layer** permet de mettre en œuvre les algorithmes de chiffrement négociés pour sécuriser les messages des autres sous-protocoles (Handshake, Change Cipher Spec et Alert), et protéger les données d'application. Il fournit aussi des services d'intégrité et de confidentialité en ajoutant un numéro de séquence à chaque paquet de données, avant d'appliquer le chiffrement et le hachage aux données applicatives. Il a

aussi pour fonction la fragmentation, la compression (optionnelle) des données, et l'acheminement de ce résultat vers l'autre extrémité. Une fois ces données reçues, elles seront déchiffrées, décompressées, réassemblées et vérifiées, puis délivrées à la couche supérieure.

✓ **Change Cipher Spec (CCS)** est un protocole qui permet la synchronisation des paramètres cryptographiques (i.e. passage des paramètres cryptographiques courants ou futurs qui sont négociés pendant le Handshake). Ce protocole comprend un seul et unique message (1 octet) qui porte le même nom que le protocole, il permet de signaler au protocole Record Layer les algorithmes de chiffrement qui viennent d'être négociés.

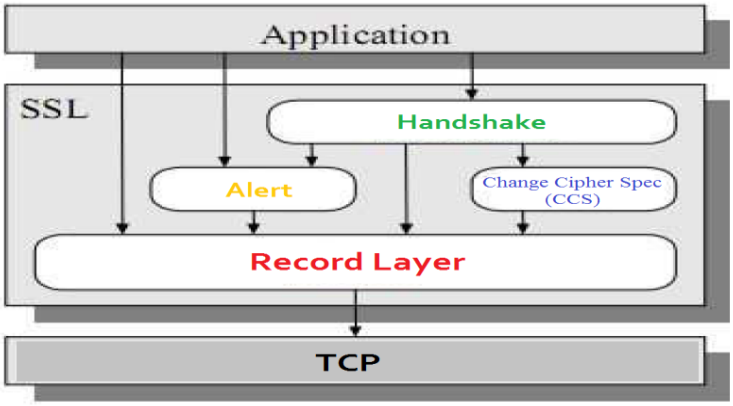

Figure 13 : Sous protocole de TLS

✓ **Alert** est un sous-protocole qui permet de signaler des anomalies suite aux erreurs provenant du client ou du serveur, au cours d'une session. Dans la deuxième phase, les messages d'alertes échangés sont chiffrés, pour contrer les attaques de demande de fermeture de connexion par un intrus. Les messages d'alerte contiennent 2 octets et sont de deux niveaux d'alertes : "*Closure alert*" met fin à une session lorsque la connexion est terminée normalement, et "*Error alert*" indique un problème survenu durant les échanges de données. Ce dernier message est subdivisé en deux niveaux de sévérité : "*Fatal Error*" ou "Warning Error". Le premier niveau implique une fermeture immédiate de la session, tandis que l'autre caractérise une erreur récupérable.

2.3.2.2) Les paramètres d'état d'une session TLS

TLS possède quatre états logiques qui sont définis par "*Current Read/Write*" et "*Pending Read/Write*". Ces états sont configurés et maintenus par la session sécurisée en

utilisant des variables. Ce sont des éléments d'information d'état, qui seront mis à jour pendant une session active ou lors de l'établissement d'une nouvelle connexion. Dans le cas du Record Layer, ces paramètres sont traités sous les états "*Pending Read/Write*", alors que pendant le Handshake, ces paramètres de sécurité sont dans les états "Current Read/Write" et leurs négociations se font sans chiffrement.

Pour établir une authentification, chaque session TLS comporte plusieurs variables d'état à négocier. Cette négociation est réalisée pendant les phases du sous-protocole Handshake [83]. Les paramètres à négocier dans une session SSL sont :

- ✓ **Session_Id** (l'Identifiant de la session) : c'est une séquence de 32 octets choisie arbitrairement par le serveur pour identifier un état d'une session active ou pour reprendre celle-ci.
- ✓ **Server_random et Client_random** : sont deux nombres aléatoires de 32 octets, générés respectivement par le client et le serveur lors de chaque nouvelle session.
- ✓ **Peer certificate (le certificat du pair)** : c'est un certificat X 509 qui correspond soit au serveur soit au client.
- ✓ **Cipher Spec (Suite de Chiffrement)** : permet de spécifier les algorithmes de chiffrement et de hachage : algorithme d'échange de clé (DH, RSA, etc.), le type de chiffrement (stream ou block cipher encryption), l'algorithme de chiffrement (DES, 3DES, AES, etc.), l'algorithme de hachage MAC (MD5, SHA-1, etc.), et IsExportable (pour indiquer si le Cipher est exportable ou non).
- ✓ **Is_resumable** est un drapeau indiquant l'existence de nouvelles connexions pouvant être créées à partir d'une session existante.
- ✓ **Compression method** est l'algorithme de compression utilisé, pour l'instant ce champ reste vide (NULL).
- ✓ **Master Secret** est une clé symétrique de 48 octets partagée entre le poste client et le serveur RADIUS.
- ✓ **Sequence number** est un numéro de séquence maintenu par chaque extrémité (Client/server), c'est une variable codée sur 8 octets pour numéroter chaque message échangé. Elle est synchronisée à chaque état (lecture ou écriture), maintenue séparément pour chaque connexion. Elle est incrémentée à chaque émission ou réception d'un message associé à une session. Il existe deux types de numéro de séquence : une séquence est prise en compte pendant l'émission d'un message et l'autre est prise en

considération pendant la réception. Le numéro de séquence s'intègre aux données avant d'appliquer le chiffrement et de calculer le haché (MAC).

2.3.2.3) Déroulement du protocole Handshake :

Le protocole Handshake permet de négocier les algorithmes cryptographiques (symétrique, asymétrique) et les fonctions de hachage qui seront utilisés dans une session. Une fois l'authentification terminée entre un serveur et un client, les deux parties partagent une clé secrète pour créer un tunnel permettant d'échanger des données d'une manière sécurisée. Le sous-protocole Handshake peut se dérouler de deux manières : Full Mode ou Resume Mode

A) Full Mode (Figure 14)

Etape 1 : elle se compose de deux messages, le premier message ClientHello envoyé par le client qui contient : la version du protocole TLS, l'identificateur de la session (Session_Id), une valeur aléatoire (random 32 bits), un Timestamp, la liste des suites de chiffrement choisies (CipherSuite), et l'algorithme de compression (la liste des méthodes de compression). Le serveur choisit parmi la CipherSuite, celle qu'il est capable de supporter et il renvoie un message ServerHello qui contient, entre autres, la CipherSuite sélectionnée et l'algorithme de compression choisi. Si le serveur ne supporte aucune "CipherSuite" proposée par le client, le serveur ne poursuit plus le Handshake. Les valeurs aléatoires échangées sont utilisées pour contrer les attaques de type MITM, et servent d'aléa ou de graine (Seed) pour calculer des clés d'intégrité et de chiffrement.

Étape 2 : le serveur envoie, juste après le message ServerHello, son certificat qui contient sa clé publique permettant de l'authentifier auprès du client par l'intermédiaire d'une autorité de certification (CA) ou d'une infrastructure à clés publiques (ICP) ou PKI (Public Key Infrastructure). De son côté le serveur peut aussi demander au client son certificat, mais cette authentification mutuelle est optionnelle, elle n'est généralement pas implémentée dans tous les sites Web. Le message "ServerHelloDone" marque la fin de la phase I d'authentification, en indiquant que l'étape du Handshake est complète.

Étape 3 : le client vérifie le certificat du serveur, la validité de l'identité du serveur et la validité de la signature du CA (Certificate Authority). Une valeur aléatoire de 48 octets : appelée "Premaster Secret", est générée et chiffrée par le client avec la clé publique du serveur. Cette valeur est envoyée au serveur dans le message "ClientKeyExchange".

Il envoie également le message "CertificateVerify" de confirmation explicite, dans le cas où le client est certifié. Ce message est une empreinte numérique "hash" de tous les messages échangés depuis le premier message "ClientHello". Ce condensât est ensuite signé avec la clé privée du client et envoyé au serveur. Le serveur vérifie la signature du client en utilisant la clé publique du client.

Le serveur déchiffre cette valeur pour obtenir ce "Premaster Secret", en utilisant sa clé privée. Le client et le serveur dérivent celle-ci pour obtenir toutes les clés de la session, en particulier la nouvelle clé "Master Secret" qui est un élément extrêmement sensible de sécurité dans une session TLS. Cette clé permanente est utilisée pour générer des clés temporaires symétriques nécessaires à la création d'un tunnel TLS et le calcul du MAC.

Par conséquent, le dévoilement de cette clé peut compromettre tous les échanges de données, car l'attaquant peut en déduire toutes les clés temporaires. Le "Master Secret" est donc un élément essentiel, qu'il faudrait préserver et protéger de la même façon que la clé privée asymétrique qui est généralement stockée dans des périphériques physiques comme les cartes à puce. La dérivation de cette clé est obtenue en utilisant la fonction Pseudo Random Function (PRF) [83].

	Client		Server
Etape 1	ClientHello	-------->	
			ServerHello
Etape 2			Certificate*
			ServerKeyExchange*
			CertificateRequest*
		<--------	ServerHelloDone
Etape 3	Certificate*		
	ClientKeyExchange		
	CertificateVerify*		
Etape 4	[ChangeCipherSpec]		
	Finished	-------->	
			[ChangeCipherSpec]
		<--------	Finished
<--------- Encrypted data exchange --------->			

** Indicates optional messages that are not always exchange*

Figure 14 : Déroulement des échanges du protocole Handshake

Étape 4 : Les deux messages "CCS" et "Finished" marquent la fin du sous-protocole Handshake et le début des échanges de données chiffrées. Le message CCS indique à son pair que le chiffrement des échanges de données postérieurs s'effectuera selon les choix des paramètres définis au cours des étapes précédentes. Quant au deuxième message "Finished", il est immédiatement envoyé après le message "CCS" pour vérifier que l'échange de clés et les processus d'authentification ont réussi. Il s'agit d'un haché de tous les messages échangés pendant le Handshake ainsi que les algorithmes cryptographiques négociés (à l'exception des messages CCS et Alert). Le bon déroulement du chiffrement et de déchiffrement de ce message "Finished" permet d'une part de valider les clés et les algorithmes cryptographiques négociés dans les étapes précédentes et d'autre part d'empêcher les attaques de rejeu en vérifiant l'intégrité de l'ensemble des échanges pendant le Handshake.

Après ces étapes de Handshake, les échanges de données applicatives chiffrées commencent entre le client et le serveur de la même manière que le message "Finished". Les chiffrements utilisent des clés symétriques temporaires, appelées "*Key Blocks*".

B) Reprise d'une session déjà établie (Resume Mode)

Une nouvelle session sécurisée peut être reprise à partir des variables d'état d'une session récente stockée dans le cache du serveur. Ceci permet d'améliorer les performances du protocole TLS en simplifiant la renégociation des clés cryptographiques et du Handshake. Dans ce cas, le Handshake est en "Resume Mode ".

Lorsque le client veut reprendre une session, il envoie dans le message "ClientHello" l'identificateur de la session à reprendre. De son côté le serveur vérifie l'existante d'une correspondance de cet identifiant dans son cache. Si le serveur accepte de reprendre la session en question, il enverra un message "ServerHello" avec l'identificateur de cette session et ensuite envoie des messages comme dans l'étape 4 précédentes "ChangeCipherSpec" et "Finished". Le client devra répondre avec les mêmes messages "ChangeCipherSpec" et "Finished". Les variables d'état seront renouvelées ainsi que la clé "Key Block" et les clés temporaires de chiffrement.

Dans le cas contraire, le serveur crée un nouvel identifiant de session et l'envoie dans le message ServerHello et cela implique l'initiation d'une nouvelle connexion en Full Mode.

C) La fonction PRF et la dérivation des clés

La fonction PRF est au centre de la génération des clés temporaires. Elle est également utilisée dans le message Finished du Handshake. La fonction PRF commence par une fonction d'expansion (P_H) qui prend une clé secrète et un peu de texte pour produire une valeur arbitraire. Les deux fonctions sont définies comme suit Figure 15 :

$$P\_H(secret, text) = HMAC\_H(K, A(1), T) \oplus HMAC\_H(K, A(2), T) \oplus HMAC\_H(K, A(3), T) \oplus \ldots$$

Where: (A)

⊕: concatenation

A(): $A(0) = text$

$A(n) = HMAC\_H(secret, A(n-1))$

H: hashing function, SHA1 or MD5

K: key

T: text

$$PRF(secret, text) = P\_MD5(S1, text) \otimes P\_SHA1(S2, text)$$

Where: (B)

⊗: exclusive or

S1: is the first half of the secret bitwise

S2: is the second half of the secret bitwise

Figure 15 : La fonction d'expansion et la fonction PRF (source [37])

La fonction PRF est une fonction cryptographique appliquée sur deux paramètres : un secret et un texte. Ce dernier est une concaténation qui comporte un label (string) et une valeur aléatoire (seed). Le *PRF (secret, label*, *seed)* est représenté sous la forme suivante :

P_MD5(s1, label || seed) XOR P_SHA1(s2, label || seed).

Label : permet de générer différentes clés en utilisant le même secret dans le PRF,

seed : valeur aléatoire.

Le secret est divisé en deux parties égales (S1 et S2). Chaque partie est utilisée comme un secret pour la fonction P_H (P_MD5 ou P_SHA-1) qui est basée sur HMAC (RFC 2104). La fonction P_H est calculée en utilisant un ou exclusif (Figure 15-A). Elle est exécutée autant de fois que possible.

Pour calculer par exemple une clé "Master Secret", d'une valeur de 48 octets en sortie, la PRF sera donc exécutée trois fois pour P_MD5, et P_SHA-1. Par contre, les 12 derniers octets de sortie seront éliminés pour obtenir les 48 octets. Cette clé est à son tour transformée en "Key Block" qui est constituée d'un ensemble de quatre clés pour chiffrer et calculer l'intégrité de données échangées : Client_Write_MAC_Secret, Server_Write_MAC_Secret, Client_Write_Key, Server_Write_Key, et 2 Initial Vectors. Ces clés sont calculées comme suit (Figure 16) :

✓ Server_MAC_write_secret et Client_MAC_write_secret sont deux clés différentes, employées pour calculer le MAC. La taille du MAC dépend des algorithmes de hachage choisis. La taille du MAC calculée à partir de la fonction MD5 est de 16 octets alors que celle de la fonction SHA est de 20 octets.

✓ Server_write_key et Client_write_key sont deux clés différentes utilisées pour le chiffrement symétrique des données, la première est utilisée par le serveur et la deuxième par le client.

✓ Initialization Vectors (IV) sont deux vecteurs d'initialisation pour le chiffrement symétrique en mode blocs, i.e. CBC (Cipher Bloc Chaining) [83], dont le résultat dépend du chiffrement du bloc précédent.

$$
\begin{aligned}
X(secret, label) &= PRF(secret, label \oplus client\_random \oplus server\_random) \\
master\_secret &= X(pre\_master\_secret, \text{“master secret”}) \\
key\_block &= X(master\_secret, \text{“key block”})
\end{aligned}
$$

Key Block

client write MAC secret	server write MAC secret	client write key	server write key	client write IV	server write IV

Figure 16 : Dérivation des clés avec la fonction PRF (source [37])

2.3.3) La méthode d'authentification EAP-TLS

La méthode d'authentification EAP-TLS [82] s'appuie sur le protocole TLS [83] pour offrir un niveau de sécurité très élevé qui vise à protéger les échanges de données sur Internet en mettant en œuvre l'authentification mutuelle entre serveur et client, la confidentialité et l'intégrité des données. EAP-TLS s'impose donc comme le protocole le plus populaire de sécurisation des échanges. C'est un protocole robuste face aux attaques de type MITM [6] [7], puisqu'elle repose sur la cryptographie asymétrique (Certificats X509).

Le serveur d'authentification RADIUS et le Supplicant s'appuient sur la méthode d'authentification EAP-TLS pour établir une authentification TLS avec des certificats X509, en utilisant les précédentes étapes du Handshake. Ces étapes commenceront après les messages (*Identity_Request et Identity_Response*) (Figure 17). Les étapes d'authentification par la méthode EAP-TLS sont les suivantes :

Étape (3-a) : le serveur envoie au client une requête de démarrage de TLS au moyen d'un paquet (*EAP-TLS_Start*). La négociation du protocole TLS comprend plusieurs échanges. Aux étapes (4 à 6), le client répond par un message (*ClientHello*) qui déclenche les quatre étapes du (*Handshake*) déjà détaillées (§2.3.2.3). Après une authentification réussie, le serveur envoie au client un paquet (*EAP-Success*) pour lui signifier que l'authentification est acceptée et un paquet (*Access-Accept*) qui contient la clé PMK, est envoyé au NAS pour lui demander d'autoriser le trafic pour ce client.

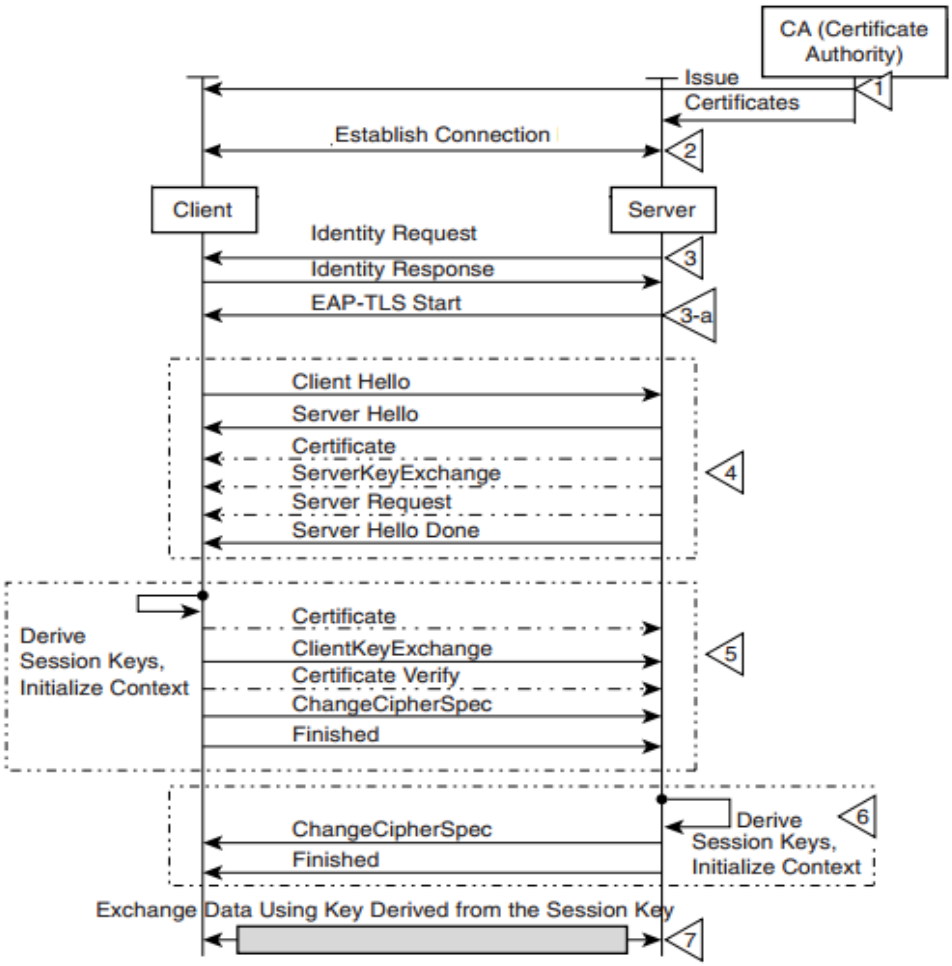

Figure 17 : Chorégraphie de négociation avec le protocole EAP-TLS

2.3.4) Les limites et faiblesses de TLS

TLS est très souple dans son intégration dans des solutions de sécurité, dans le sens où il n'exige pas de respecter l'authentification mutuelle, la vérification des certificats, leur statut de révocation (CRL). Malheureusement, cette souplesse constitue une menace pour le

protocole, car un attaquant peut se substituer à un serveur original et rediriger le trafic réseau vers un autre faux site.

Comme tous les protocoles, TLS est théoriquement vulnérable aux attaques par force brute dans le cas où les clés de 40 bits sont utilisées au lieu des clés de 128 bits. Il comporte également, un certain nombre de failles : attaque par déni de services (DoS), introduction d'un code malveillant ou encore utiliser l'attaque de type MITM (Annexe A).

2.4) Conclusion et premiers choix

Dans ce chapitre nous avons présenté les architectures de sécurité basées sur des standards qui ont fait leur preuve dans des infrastructures à grande échelle (chez les opérateurs de télécommunication). Nous avons tenté de réaliser une analyse concise des mécanismes de sécurité définis pour les réseaux. Nous avons constaté l'existence d'une multitude de méthodes d'authentification, permettant de mettre en œuvre des infrastructures AAA basées sur des standards (802.1X, EAP, MD5, TLS, etc.).

Pour satisfaire les exigences de SecFuNet et respecter les critères de sécurité, notre système de gestion des identités doit implémenter des fonctions et des composants étudiés dans les paragraphes précédents. Dans ce contexte, les standards (RADIUS et DIAMETER) s'imposent comme un compromis idéal pour notre projet, puisqu'ils sont implémentés dans plusieurs solutions libres, et ont fait leur preuve dans des infrastructures à grande échelle (chez les opérateurs de télécommunication).

Le Tableau 1 présente une comparaison des composants supportés par DIAMETER et RADIUS et leurs imbrications avec le protocole d'authentification TLS. Les fonctions de la triade A sont implémentées, dans ces protocoles, d'une manière indépendante. Elles peuvent être installées sur le même serveur ou sur des proxys pour déléguer une partie de ces fonctions. Par conséquent, ces deux standards nous offrent la possibilité de les mixer avec différents protocoles d'authentification, basés sur des standards et reposent sur des outils cryptographiques asymétriques (i.e. TLS).

RADIUS et DIAMETER permettent de centraliser l'authentification, non seulement des utilisateurs mais aussi des équipements et des applications (exemple : DIAMETER). Les deux systèmes contrôlent aussi l'accès aux ressources et collectent des informations sur leur utilisation. Cette étude, nous montre que DIAMETER est actuellement le protocole le plus à même de satisfaire les nouveaux besoins suscités par la mobilité. Notamment, il permet

d'assurer l'itinérance (Roaming) dans une architecture transorganisationnelle, contrairement à RADIUS. L'implémentation de DIAMETER constitue malheureusement un investissement très conséquent pour une simple preuve de concept, d'autant plus qu'il y a une compatibilité ascendante entre RADIUS et DIAMETER.

RADIUS est donc plus simple à implémenter dans notre infrastructure d'autant plus qu'il s'imbrique facilement avec le protocole TLS, lequel fournit des outils cryptographiques asymétriques ainsi que l'authentification mutuelle.

Tableau 1: Comparaisons des standards AAA et le protocole d'authentification

	RADIUS	DIAMETER	TLS
Identification / Athentification	X	X	X
Autorisation	X	X	
Accounting	X	X	
Référentiel d'identités	LDAP,…	LDAP,…	LDAP,…
Méthode d'authentification	EAP-TLS	EAP-TLS	TLS
Révocation		X	X
Provisioning / Deprovisioning	X	X	X
Authentification unique (SSO)	X	X	X
Fédération d'identités	X	X	X
Vie privée	X	X	X
Anonymat et pseudonyme	X	X	X
Interopérabilité entre eux	X	X	X

Nous avons également présenté une étude critique de plusieurs méthodes d'authentification basées sur EAP, en particulier, nous avons analysé les faiblesses et les limites du protocole TLS en termes de résistance contre plusieurs attaques, dont le véritable problème est le manque de mise à jour des applications ou le manque de déploiement de l'authentification mutuelle. Malgré ces limites et ces vulnérabilités, le protocole TLS reste quand même le protocole le plus utilisé, le seul protocole de sécurité robuste déployé à grande échelle, capable de renforcer les propriétés de sécurité, en particulier l'intégrité, l'identification et l'authentification mutuelle. Il est considéré comme fiable lorsque les contre-mesures adaptées sont appliquées. Il possède de nombreuses qualités, souvent insuffisamment

exploitées et de grands avantages. Il ne se limite pas qu'aux applications traditionnelles (HTTP, POP, IMAP, FTP, TELNET, etc.), mais il s'intègre facilement dans des briques réseau, (réseau filaire, sans fil, etc.), en association avec RADIUS.

Ce chapitre, nous a amenés à bien préciser les besoins de sécurité à mettre en place pour protéger les ressources réseaux. Parmi ces besoins, nous citons l'authentification mutuelle, la confidentialité, l'intégrité, la possibilité de mettre en place des identifiants anonymes, la protection des clés cryptographiques. Ces besoins nécessitent bien évidemment, de définir de nouvelles infrastructures de confiance intégrant des équipements physiques inviolables pour stocker des éléments sensibles (clés secrètes). En revanche, l'absence de ces infrastructures de confiance peut compromettre les propriétés de sécurité offertes par le protocole TLS. La mise en place des PKI et des composants sécurisés (i.e. cartes à puce) permet, en partie, de limiter certaines failles.

Ces infrastructures impliquent des configurations particulières de ces composants, qui doivent être aisément compatibles avec les Infrastructures d'Authentification et d'Autorisation AAI (Authentication and Authorization Infrastructure), lesquelles doivent implémenter les approches d'authentification unique (SSO) et fédératives pour limiter les authentifications multiples. C'est ce que nous aborderons dans le chapitre suivant.

Chapitre III : Comparaison de quelques Infrastructures d'Authentification et d'Autorisation

3.1) Introduction

Une de nos principales préoccupations dans ce chapitre est d'établir les limites et les exigences du système d'identité global qui sera développé pour offrir des services d'authentification dans des environnements virtuels. Ces environnements impliquent des relations transorganisationnelles, évolutives, dynamiques et mutualisables. Ils proposent des ressources interopérables, résilientes pour des utilisateurs nomades et autonomes.

Dans le chapitre précédent, nous avons montré que la gestion des identités et la sécurisation des accès constituent un enjeu majeur. L'enjeu est d'autant plus considérable que des menaces bien réelles pèsent sur les sphères personnelles, professionnelles, économiques, et sociales. Les conséquences sont aussi lourdes en cas d'usurpation d'identité, peuvent aggraver les critères de sécurité d'un système d'information, à la hauteur des habilitations allouées à un utilisateur. Ces conséquences ont naturellement amené la gestion de l'identité à se placer au cœur des préoccupations des décideurs et des prestataires de service.

Face à cette complexité, certaines solutions tentent de relever certains défis (gestion du processus de l'authentification unique et des habilitations d'accès aux différentes ressources informatiques, gestion de la fédération), pour permettre à un utilisateur de naviguer sur plusieurs domaines différents en s'authentifiant une seule fois, sans pour autant que ces domaines aient accès à des informations confidentielles. Elles visent à réduire le nombre d'identifiants associés aux différents services, tout en assurant un bon niveau de sécurité.

Ces différents services informatiques, accessibles généralement par le Web (intranet, courrier électronique, forums, agendas, applications spécifiques, etc.) doivent donc se baser sur une infrastructure commune pour assurer une authentification unique et mutuelle aux utilisateurs ainsi que des autorisations d'accès nécessaires à ces services.

Plusieurs solutions AAIs existent, nous en avons identifié, pendant l'élaboration des spécifications du projet SecFuNet quelques-unes comme celles répondant le plus aux problématiques du projet : (OpenID [66], Shibboleth [69] et Higgins [59]). Cependant, après nos prospections sur Internet, certaines infrastructures pertinentes ont été incluses dans notre étude (Annexe B), ce qui nous a permis d'accroître notre état de l'art et d'explorer les approches utilisées dans ces infrastructures pour gérer par exemple : la propagation des

authentifications, la gestion des identités multiples et centrées sur l'utilisateur, la gestion des pseudonymes et la fédération des identités...

Dans ce chapitre, nous détaillerons les modèles et les approches des AAIs (§3.2) et l'authentification unique (§3.3), nous essayerons de faire une comparaison de quelques AAIs (§3.4). Nous dégagerons (§3.5) quelques considérations concernant la protection de la vie privée et les niveaux d'assurance de ces produits (§3.6). Enfin, nous mettrons en exergue les avantages offerts et les inconvénients de ces architectures (§3.7) ainsi que les briques communes pour enfin conclure (§3.9) ce chapitre, avec des orientations précises dans nos choix à prendre en compte pour concevoir notre système de gestion des identités compatible avec les différents contextes Cloud Computing dans lesquels il pourra être intégré pour gérer aussi des ressources virtuelles.

3.2) Les modèles de gestion des identités

Avant d'analyser et étudier les infrastructures existantes, il convient de définir les termes utilisés dans ces modèles. L'IAM (Identity and Access Management) est basée sur des concepts et repose sur des modèles, impliquant des relations transorganisationnelles. Ces modèles de gestion des identités [8] [9] [10] sont classés comme conventionnels, centralisés, fédérés, coopératifs et centrés sur l'utilisateur. Chacun d'eux a une forme différente mais le principe reste le même, et se compose de trois entités principales : un utilisateur (généralement substitué à un navigateur Web), un fournisseur de service (ou SP : Service Provider), et un fournisseur d'identité (ou IdP : Identity Provider). Les différents modèles sont illustrés (Figure 18) :

- *Le modèle conventionnel* est une architecture monolithique. Dans ce modèle les Identités (ID) sont manipulées individuellement par chaque fournisseur de services (SP), qui joue également le rôle d'un fournisseur d'identité (IdP) (Figure 18-a). Un utilisateur crée son identité numérique pour chaque SP, avec lequel il interagit. Habituellement, les IDs des utilisateurs ne sont pas partagées entre les différents prestataires de services. Cette approche a tendance à être coûteuse pour les utilisateurs et les SPs, puisque chaque SP exige son propre ensemble d'attributs pour former l'identité numérique de l'utilisateur (nom de compte, mot de passe, adresse, date de naissance, etc.). Par conséquent, cette gestion multiple, pour un utilisateur, est contraignante, car il doit se soucier de créer un login/mot de passe différent pour chaque SP et fournir les mêmes informations à plusieurs reprises. Cette opération devient

fastidieuse pour lui au moment de la création de ses comptes dans les SPs et le conduit à devenir plus négligent ou imprécis pour remplir ses attributs, lesquels peuvent s'avérer essentiels pour accéder aux ressources offertes par les SPs. Ce modèle conventionnel n'est pas adapté à la gestion des identités dans une architecture distribuée. Les modèles de gestion centralisés, fédérés et User-Centric offrent des alternatives beaucoup plus flexibles pour consommer des services d'une manière transparente.

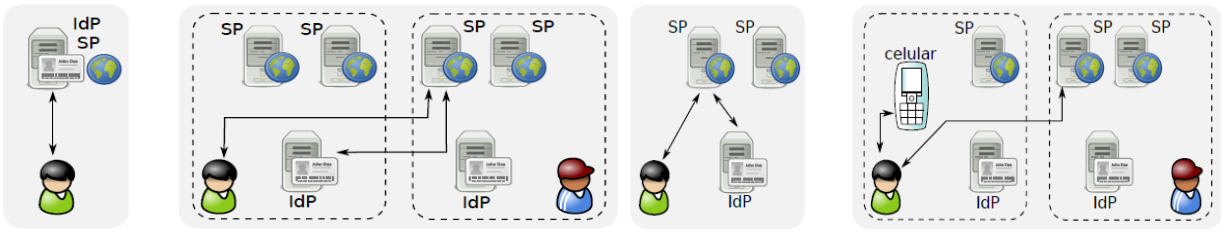

Figure 18 : Les Modèles de gestion des identités

- *Le modèle centralisé* (Figure 18-c) : ce modèle est apparu comme une solution alternative à la rigidité du modèle conventionnel. Il est basé sur le partage des identités des utilisateurs entre les SPs et sur le concept d'authentification unique [8]. Microsoft Passport Network a été un précurseur de ce modèle et a essayé d'éviter les incohérences et les redondances du modèle classique, ce qui donne aux utilisateurs la possibilité d'interagir avec les différents fournisseurs de services sans avoir besoin de s'authentifier dans chacun des services. Dans ce modèle, tous les SPs entretiennent des relations de confiance avec un IdP (un fournisseur d'identité ou une autorité d'authentification), ils délèguent entièrement le service d'authentification des utilisateurs à l'IdP. Ce dernier prend en charge uniquement le processus d'authentification des utilisateurs et fournit les informations nécessaires concernant les attributs des utilisateurs aux SPs. Ce concept d'authentification unique (SSO), sous forme de jeton, présente une grande flexibilité pour les utilisateurs, car ils peuvent utiliser les mêmes informations d'identification pour consommer les services offerts par les SPs, jusqu'à expiration du jeton. Ce modèle centralisé présente un point faible [11] puisque le fournisseur d'identité a le contrôle absolu sur les informations des utilisateurs, et peut les utiliser de quelque manière que ce soit. C'est la raison principale pour laquelle Microsoft Passport Network n'a pas été couronné de succès.
- *Le modèle fédéré* : c'est une architecture distribuée, ce modèle est apparu afin d'éviter les carences présentées par le modèle centralisé. Le principe de ce modèle se base sur

la délégation du service authentification aux IdPs (Figure 18-b). Ces derniers sont disposés dans différents domaines administratifs. Un domaine administratif est composé d'utilisateurs et de plusieurs SPs associés à des IdPs. Ce concept repose sur des relations de confiance qui s'étendent sur plusieurs domaines administratifs, impliquant la signature d'accords entre ces derniers. Ce modèle fédéré est une approche pour optimiser les échanges d'informations d'authentifications des utilisateurs grâce aux relations de confiance entre IdPs [12]. Une identité authentifiée par IdP dans un domaine sera forcément reconnue par les SPs dans les autres domaines de confiance. Ce modèle offre des facilités aux utilisateurs car il leur évite d'utiliser différentes identités pour s'authentifier de nombreuses fois. L'avantage de ce modèle fédéré, c'est que la fonction d'authentification est déléguée aux IdPs par conséquent, les autres attributs utilisateurs sont gérés par les SPs.

- *Le modèle coopératif :* chaque domaine administratif gère ses propres ressources (utilisateurs, serveurs, applications, etc.) et possède ses propres IdPs. L'authentification se fait donc auprès de l'entité de rattachement (IdP), car les identifiants ne sont pas répliqués sur les autres domaines et chaque domaine gère indépendamment sa propre politique de sécurité et délivre un nouveau niveau d'habilitation pour chaque authentification. Quand un utilisateur souhaite consommer un service, l'IdP associé à ce service l'authentifie et lui délivre un niveau d'habilitation sous forme de jeton, selon les accords passés avec le domaine administratif de cet utilisateur.

- *Le modèle User-Centric Identity* est une architecture de gestion des identités centrée sur l'utilisateur (Figure 18-d). Ce modèle vise à donner à l'utilisateur un contrôle total de ses identifiants numériques. Les principales applications de ce modèle sont intégrées dans des modèles évoqués précédemment (§3.2). Cependant, ce modèle est plus largement utilisé avec le modèle d'identité fédérée [9]. Chaque identité utilisateur, destinée aux différents fournisseurs de service, est stockée sur un support physique qui est maintenu par l'utilisateur lui-même : carte à puce ou téléphone cellulaire. L'utilisateur s'authentifie dans ce dispositif physique. Cette approche respecte pleinement les préférences de confidentialité de l'utilisateur et la vie privée.

3.3) L'authentification unique (SSO)

L'authentification unique est un mécanisme qui permet à un utilisateur d'accéder successivement à plusieurs ressources informatiques hétérogènes, ou de se connecter à plusieurs sites Web sécurisés, sans avoir besoin de s'authentifier à chaque fois en utilisant

son nom d'utilisateur et son mot de passe. Les contraintes de réauthentification sont ainsi éliminées. D'un point de vue technique, l'utilisateur s'authentifie uniquement une seule fois dans un domaine administratif et le service SSO s'occupe ensuite d'approvisionner les ressources, appartenant à ce domaine : validité de l'authentification et des habilitations associées à cet utilisateur.

3.3.1) Le service SSO

Le SSO n'est pas une fonction de sécurité au sens du protocole AAA, mais un service qui émerge du système de gestion des identités et du contrôle des accès. Il assure la propagation et la diffusion de l'authentification valide d'un utilisateur auprès des multiples applications appartenant à un domaine de sécurité SSO. Il existe deux services :

- ✓ Le Web SSO est un service d'authentification orienté Web service qui fournit une signature unique pour un client léger. Il est facile à implémenter car le client léger n'a besoin d'aucune configuration, tout se passe du côté du serveur. La plupart des solutions existantes de ce modèle sont des solutions libres.

- ✓ Le SSO "Lourd" (e-SSO) concerne toutes les ressources système et réseau. Ce type de produit nécessite d'installer un agent sur le poste de travail. Le déploiement de ce modèle reste très marginal et ne concerne que très peu d'applications. La plupart des solutions existantes de ce modèle sont des solutions commerciales qui se basent sur le même principe que celui de KERBEROS (l'ancêtre du SSO) : Imprivata (OneSign SSO), Avencis (SSOX), CA (CA-SSO), Evidian (WiseGuard), Actividentity (SecureLogin), etc.

Un système SSO minimum est composé de plusieurs briques classiques suivantes :

- ✓ **Le navigateur de l'utilisateur** est un client léger qui permet à l'utilisateur de s'authentifier au niveau du serveur d'authentification et de consommer des services.

- ✓ **Le serveur d'authentification** est un élément central du SSO ; il gère l'authentification des utilisateurs, la persistance de la session, et la propagation d'identité entre les applications (souvent en créant des jetons).

- ✓ **L'agent de redirection :** est l'interface de communication entre les applications et le serveur d'authentification. Son rôle est de vérifier que l'utilisateur est authentifié. Sinon, il doit être redirigé vers le serveur d'authentification. L'agent de redirection est souvent un plug-in intégré dans chaque client léger.

✓ **Les serveurs d'applications :** délivrent les services qui vont être consommés par l'utilisateur final, les services offerts doivent être compatibles avec le système SSO, et avec l'agent de redirection.

3.3.2) Les avantages du SSO

Chaque utilisateur ne gérera qu'un seul identifiant/mot de passe pour toutes les applications. Il n'est pas nécessaire de s'authentifier à chaque invocation d'un nouveau service. La gestion du cycle de vie des jetons d'authentification auprès des applications se fait d'une manière transparente et automatique. Une nouvelle réauthentification n'implique que les informations nécessaires dont elle a besoin. Ainsi, la sécurité est renforcée, car la circulation des identifiants et des mots de passe est limitée et ces derniers sont remplacés par des jetons de session, après la première authentification. Ces jetons facilitent également la traçabilité des utilisateurs dans un même domaine de sécurité.

3.3.3) Les inconvénients du SSO

L'usurpation d'identité constitue une faiblesse de sécurité de SSO. Une fois un usurpateur d'identité authentifié, il peut accéder à l'ensemble des services de cette identité usurpée. Pour cette raison, il faut combiner les solutions de SSO, avec des systèmes d'authentification forte (i.e. carte à puce). Les SSO sont orientés Web services et compatibles avec tous les clients légers, cette compatibilité n'est pas garantie avec toutes les applications, notamment pour tous les clients lourds. L'interopérabilité des différents SSO est loin d'être possible pour tous les produits existants, bien qu'ils s'appuient sur des standards (SAML, XACML [60], Web Services, etc.). La centralisation du référentiel et de l'authentification dans un IdP devient encore plus sensible, puisqu'elle contient tous les mots de passe des entités et constitue une faiblesse de sécurité. Une attaque par déni de service (DoS) rend la fonction d'authentification inaccessible, ce qui peut avoir des répercussions sur la disponibilité de toutes les ressources informatiques d'un domaine, d'où l'intérêt de privilégier les approches coopératives ou centrées sur l'utilisateur (User-Centric Identity).

3.4) Comparaison de quelques AAIs

Cette étude comparative n'a pas pour vocation de dresser un inventaire exhaustif des produits présents sur le marché ni de détailler leurs différences, d'autant plus que les multiples solutions offertes (libres ou commerciales) dans ce domaine, rendent la comparaison plus

difficile si nous tenons compte de la diversité des spécifications de chaque produit (en particulier les terminologies et les référentiels utilisés dans les différentes infrastructures). Toutefois les produits cités en Annexe B, prennent en charge le Web SSO.

L'idée est d'analyser le comportement et la cinématique d'accès aux différents services, étudier les mécanismes utilisés pour sécuriser les échanges d'information d'identité et protéger la vie privée. L'objectif de l'étude est de dégager les briques fondamentales communes à la majorité de ces infrastructures, ensuite, il sera possible d'orienter nos réflexions vers la proposition d'un modèle global de gestion des identités compatible et interopérable avec celles-ci, et la réalisation de notre modèle apportera la preuve de notre concept.

En effet, les AAIs étudiées (Annexe B) tentent de conserver un maximum d'interopérabilité en partageant quelques points communs, tels que l'authentification unique (SSO), les procédures d'authentification, l'authentification par mot de passe, les échanges d'attributs, les préoccupations concernant la confidentialité des utilisateurs et l'anonymat. Elles sont basées sur l'idée que l'utilisateur utilise un navigateur Web et que les échanges d'informations entre les fournisseurs d'identité et les services sont effectués par une redirection HTTP. Or, les exigences de sécurité dans ce contexte distribué devraient être plus drastiques que dans le cas d'une gestion des identités conventionnelle « monolithique ».

Les spécifications du langage SAML présentent un cadre général pour gérer des identités fédérées, dans lesquelles les métadonnées, sous la forme de messages XML, sont définies pour représenter des informations de sécurité. Les protocoles d'échange d'assertions et des relations de confiance répondent spécifiquement aux problématiques de gestion des identités fédérée. La version 2.0 de SAML est devenue le standard de facto suite à un certain nombre d'extensions ajoutées et développées par Liberty Alliance et Shibboleth. Ces extensions le rendent plus flexible et facilitent son intégration dans plusieurs infrastructures de gestion des identités, y compris dans certaines solutions étudiées (Annexe B). Il est compatible avec les spécifications des Web Services puisque la spécification WS-Federation [71] s'inspire de nombreux mécanismes de SAML. En effet, WS-Federation a une certaine similitude avec SAML. Les deux solutions offrent des mécanismes de fédération et fournissent un ensemble de fonctions semblables : SSO, Single LogOut (SLO), l'échange d'attributs basé sur des politiques de confidentialité, des relations de confiance, des pseudonymes permanents et transitoires et des métadonnées basées sur du XML. La

différence entre les deux normes, est que WS-Federation se base sur le modèle de confiance de WS-Trust.

Le projet Liberty Alliance (LA) est un des acteurs majeurs de la fédération d'identité et du Web Services ; il vise à faciliter les interactions commerciales en profitant des avantages de l'architecture orientée services (SOA) et le concept de fédération qui se caractérise par la notion de cercles de confiance. Ce projet fournit des fonctionnalités similaires à celles de la spécification WS-Federation qui est pour sa part fondée sur une pile de spécifications de tous les services Web telles que WS-Trust et WS-Policy.

LA offre une approche basée sur le contrôle d'attributs utilisateurs. Par conséquent, l'utilisateur doit être placé en mode contrôle de ses attributs qui sont stockés dans un IdP (ou un fournisseur d'attributs). Ces derniers sont en mesure de les révéler grâce à un protocole d'échange d'attributs. Une demande d'attribut doit préciser l'objet de son utilisation. Et la réponse à cette requête peut déterminer les préférences de l'utilisateur contenues dans un attribut en termes de confidentialité, de protection de la vie privée ou la politique exigée pour l'accès aux ressources.

Le projet Shibboleth est basé sur des standards ouverts tels que SAML, XML. Il hérite de leurs fonctions et fournit un moyen facile pour permettre aux applications d'utiliser le modèle d'identité fédérée. Ce projet partage donc de nombreuses similitudes avec LA puisqu'il implémente plusieurs fonctionnalités spécifiées dans SAML2.0 afin de fournir du SSO et l'échange sécurisé des attributs des utilisateurs pour tous les SP qui interviennent dans une fédération Shibboleth. L'authentification des utilisateurs peut se faire par nom d'utilisateur et mot de passe, KERBEROS, X509, etc.

OpenID, Cardspace et Higgins sont des Frameworks qui supportent les approches de fédération des identités et la gestion des identités centrée sur l'utilisateur. Le standard OpenID est largement utilisé, notamment en raison des partenariats avec les entreprises qui offrent des applications Web 2.0. L'approche OpenID adopte généralement le modèle d'identité basée sur une adresse URL (Uniform Resource Locator) ou, dans certains cas, une XRI (Extensible Resource Identifier). Alors que, Cardspace et Higgins adoptent des approches de gestion des identités basées sur les cartes purement logicielles installées, dans le système d'exploitation de l'utilisateur, sous forme d'agent actif Identity Selector (IS) pour sélectionner les identités. Lorsqu'un utilisateur sélectionne une carte associée à un IdP, l'IS prend contact avec ce dernier, lequel authentifie l'utilisateur, et enfin obtient un jeton XML signé contenant les informations demandées par ce SP. L'agent actif se présente sous forme d'une interface

d'abstraction logicielle pour sélectionner une identité, similaire au portefeuille d'un utilisateur avec plusieurs cartes d'identités. Chaque carte représente une identité différente permettant ainsi d'aider l'utilisateur a contrôlé ses multiples identités et ses préférences dans différents contextes. Dans ces modèles, il est également possible de croiser des contextes et de gérer tout type d'information utilisateur stocké sur la carte (chansons préférées, numéro du permis de conduire, numéro de l'assurance sociale et régime de l'assurance-maladie). Dans le modèle Higgins, les sélecteurs sont interopérables avec Cardspace.

Higgins est compatible avec plusieurs solutions, car il prend en charge les fournisseurs d'identité basés sur WS-Trust, mais aussi basé sur SAML 2.0. La littérature montre que l'approche Higgins est considérée comme plus flexible, car elle prend en charge les identités fournies par différentes sources et empêche un IdP de tracer des SPs (Applications Web) accessibles par l'utilisateur. Le client actif de Higgins (IS) est disponible pour différentes plates-formes, contrairement à Cardspace, et accepte tous les protocoles connus pour les identités numériques, y compris WS-Trust, OpenID, SAML, XDI, LDAP, etc.

L'approche adoptée par Cardspace assure une portabilité seulement avec les normes Web Services, en se focalisant principalement sur WS-Trust pour fournir un support pour tout type de jeton de sécurité. Malheureusement, le manque de portabilité avec les autres solutions lui a fait défaut, et par conséquent, il n'a pas été couronné de succès, ce qui a conduit Microsoft à l'abandonner en 2 011.

Un des avantages d'OpenID est qu'il ne nécessite pas de logiciel côté client. Il s'attache à offrir beaucoup plus de simplicité et de souplesse dans son implémentation. En effet, il permet à l'utilisateur de consommer des services appartenant à des domaines administratifs différents sans que cela implique de relation de confiance préétablie entre ces derniers. En fait, OpenID est un modèle de gestion des identités centré sur l'utilisateur, mais supporte l'approche coopérative.

En outre, dans OpenID un identifiant est représenté sous forme d'URL sous OpenID. Cette représentation constitue un des avantages d'OpenID qui favorise en partie la fédération des identités et l'interopérabilité avec d'autres modèles. Cette URL peut dans une certaine mesure servir d'annuaire global des utilisateurs pour publier dans leur profil la clé publique associée grâce à son extension "*Simple Registration*" (Annexe B). L'inconvénient d'OpenID est l'absence de cercle de confiance préalable entre les SPs et l'IdP, qui conduit à une grande

incertitude dans les phases d'authentification car c'est l'utilisateur qui construit ses propres relations de confiance avec les différents SPs.

3.5) Considération concernant la protection de la vie privée

Les approches de gestion des identités fédérées offrent une simplicité et une flexibilité aux utilisateurs et un moyen pratique pour consommer des services d'une manière transparente. Cependant, ces approches peuvent devenir, dans un système distribué, une menace pour la protection de la vie privée et la confidentialité des données des utilisateurs. En outre, dans les systèmes fédérés, les identifiants sont manipulés souvent dans des contextes différents. Le partage des informations privées des utilisateurs représente un défi très important à lever dans les systèmes de fédération ou de coopération. Il ne faut donc divulguer qu'un minimum d'informations sur la vie privée.

Par exemple, les authentifications et les autorisations peuvent être effectuées avec un niveau d'assurances élevé et une divulgation de données minimums, en ne fournissant que des identifiants et des attributs nécessaires pour assurer la consommation d'un service ou l'accès à une ressource. Dans l'Annexe B, nous avons étudié une technique importante pour préserver la vie privée qui consiste à utiliser des pseudonymes. Ce sont des identifiants qui ne permettent pas de retrouver la véritable identité d'une personne, les propriétés ou les attributs d'un utilisateur. Les pseudonymes peuvent avoir une signification locale, dépendant du contexte entre l'utilisateur et le SP, ou globale indépendante du contexte et valide pour l'ensemble de la fédération. La validité peut être temporaire ou permanente [36].

WS-Federation définit un service de pseudonymes, responsable de l'association des pseudonymes aux identités des utilisateurs. Dans WS-Federation, le pseudonyme peut avoir différents niveaux de volatilité permettant différents niveaux de personnalisation et de protection de la vie privée. Par exemple, un sujet peut avoir des pseudonymes qui ne durent que pour une session d'authentification, dans le but d'accroître le niveau de protection de sa vie privée et, d'empêcher les services de lui associer toutes informations persistantes. Ce service pseudonyme utilise une interface définie dans la spécification WS-Transfer. Cette spécification définit des méthodes pour créer, supprimer, mettre à jour et accéder à des pseudonymes existants.

Les pseudonymes qui sont utilisés dans les assertions SAML sont construits sur des valeurs pseudo-aléatoires qui n'ont pas de corrélation discernable avec les identifiants des utilisateurs dans l'IdP ou le SP. Un pseudonyme n'a de sens que dans le contexte de la

relation entre les deux parties communicantes. Les pseudonymes sont également destinés à rendre difficile l'association des utilisateurs à leurs transactions.

Liberty Alliance prévoit également des mécanismes de confidentialité et de protection de la vie privée dans leur modèle de gestion des identités fédérées. Leurs spécifications s'intéressent également aux questions relatives aux politiques de protection des informations personnelles multiniveaux [74], qui utilisent des labels de confidentialité similaires aux étiquettes de sécurité utilisées dans MAC (Mandatory Access Control). Dans le contrôle MAC, chaque ressource est étiquetée avec un label de sécurité qui représente la sensibilité de la ressource considérée. Un utilisateur souhaitant accéder à une ressource doit avoir un niveau d'autorisation (Clearance Level) approprié à l'étiquette de sécurité de la ressource. Dans les spécifications de Liberty Alliance, les niveaux de confidentialités qui seront appliqués pour consommer une ressource sont des politiques de confidentialité disponibles dans les IdPs (ces derniers peuvent également servir de dépôts pour les attributs d'utilisateur).

Shibboleth met l'accent sur la protection de la vie privée et les attributs des utilisateurs. Leur libération pour un SP est conditionnée par la politique du domaine d'origine (IdP) et également par les préférences de l'utilisateur.

La grande limitation de Cardspace et Shibboleth est que l'utilisateur ne peut sélectionner qu'un seul IdP et présenter un seul identifiant à un SP. Pour résoudre ce problème, il est proposé un composant appelé *Linking Service* [35] ; ce service permet aux utilisateurs d'ajouter divers attributs à partir de différents IdPs, tout en préservant leur vie privée. Le projet Higgins a également l'intention de trouver une solution à ce problème, mais la version actuelle ne l'offre pas encore. Malheureusement, OpenID présente des problèmes de protection de la vie privée : le fournisseur d'identité (OP) a accès, par construction, à tout l'historique de navigation de ses utilisateurs. Il présente aussi plusieurs dangers pour la vie privée, en particulier le vol du mot de passe et des cookies.

3.6) Niveaux d'assurance des solutions étudiées

Lors de la création des comptes des utilisateurs dans OpenID, un IdP ne dispose d'aucun moyen pour confirmer l'identité réelle de l'utilisateur. Dans ce cas, le LoA (Level Of Assurance) est faible (LoA = 1). OpenID est actuellement utilisé principalement pour des applications à faible risque comme les blogs et les réseaux sociaux.

Dans la spécification WS-Trust, le paramètre "*AuthenticationType*" a été défini pour indiquer un type d'authentification qui est exigé (ou exécuté) par rapport à une demande de jeton de sécurité particulier. Toutefois, aucune recommandation concernant le mécanisme LoA n'a été définie. Pour faciliter l'interopérabilité, WS-Federation a identifié et défini un ensemble *URIs* : *Uniform Resource Identifiers*, pour spécifier les types d'authentification communs et les niveaux d'assurance pouvant être utilisés par le paramètre WST "*AuthenticationType*" dans les messages RST and RSTR.

SAML permet d'associer des niveaux de qualité à ses assertions d'authentification, offrant de cette façon un moyen standard pour définir des niveaux d'échange d'information entre les IdPs et les SPs. Par conséquent, les niveaux LoA peuvent être inclus en utilisant un contexte d'authentification (*AuthnContext*) [61]. Ce contexte est une nouvelle spécification définie dans SAML V2.0, pour représenter LoA d'une manière simplifiée et pour inclure dans ce système des informations relatives à la qualité du processus d'authentification.

Selon les règles de l'E-Authentication Fédération (EAF), la valeur LoA est un attribut obligatoire qui doit être présent à chaque fois qu'une assertion d'authentification SAML est délivrée. L'EAF a défini une URI spéciale [62], uniquement pour identifier les attributs de LoA, et cet attribut peut avoir seulement des valeurs 1, 2, 3, 4 ou "test".

Comme SAML, Liberty Alliance a formé le groupe *Identity Assurance Expert Group* (IAEG). L'objectif de ce groupe d'experts est de créer un Framework de politiques de base, avec des règles pour les processus métier et les conditions commerciales sur lesquelles les IdPs peuvent être évalués et examinés. Le résultat principal de l'IAEG est le "*Liberty Identity Assurance Framework*" [72], dont le but est de faciliter la fédération d'identité et de promouvoir l'uniformité et l'interopérabilité entre les IdPs. Avec un focus spécifique sur le niveau de confiance "Level of trust" (LoT), ou LoA, associé aux assertions des identités.

Par ailleurs, les deux méta-systèmes Cardspace et Higgins sont entièrement agnostiques par rapport au format du jeton de sécurité. En fait, ces technologies n'ont même pas connaissance du contenu du jeton de sécurité. Les deux systèmes définissent un mécanisme permettant aux IdPs de spécifier des exigences d'authentification des services qu'ils offrent. Ils sont construits autour de la couche d'abstraction de la carte, qui est une représentation standard de l'information de l'utilisateur. Pour cette raison, ces deux systèmes peuvent fonctionner avec n'importe quel système d'identité numérique, en utilisant n'importe quel type de jeton de sécurité, y compris les noms d'utilisateur simples, les certificats X509, les tickets KERBEROS, les jetons SAML, etc.

Généralement, dans ces systèmes, lorsque l'utilisateur tente d'accéder à un service la carte d'information installée dans le dispositif du client récupère la politique du SP pour déterminer les exigences du service. Ensuite, L'utilisateur choisit une des cartes répondant à ces politiques de sécurité, l'application de la carte d'information tente de s'authentifier à l'IdP qui a émis cette carte pour obtenir un jeton signé. Finalement, ce jeton signé est envoyé au SP pour une demande d'autorisation d'accès au service concerné.

Les exigences LoA requises pour obtenir l'accès au service dépendent de la politique du SP et des exigences d'authentification définies pour ce service. L'utilisation de la politique du SP dans ces systèmes fournit le même niveau d'assurance qui est décrit dans les infrastructures de Liberty Alliance ou SAML.

3.7) Considérations finales

Nous avons illustré à travers ce chapitre, différentes solutions de gestion des identités. Ces solutions présentent les mêmes caractéristiques technologiques et essayent de relever les mêmes défis : la sécurisation de l'authentification unique et la maîtrise de la circulation des jetons de session dans une approche fédérative ou coopérative. Par ailleurs, il est important de noter que les protocoles d'échanges d'attributs diffèrent d'un standard à l'autre ainsi que les formats des jetons de session. Ceci engendre un autre problème, celui de l'interopérabilité.

En effet, ces solutions apportent des bénéfices assez évidents d'un point de vue de l'utilisateur, car le nombre d'identifiants est drastiquement réduit, ainsi que le nombre de mots de passe associés à mémoriser. Le couple (utilisateur/mot de passe) est remplacé par un jeton. En revanche, du point de vue de la sécurité, la gestion de la circulation du jeton avec des informations sensibles est extrêmement complexe, puisque l'authentification d'un utilisateur dans un domaine administratif matérialisée par ce jeton, doit se propager vers un autre domaine, dont la politique de sécurité est différente. Cependant, une solution d'authentification sera crédible en fonction des niveaux LoA et/ou LoT qu'elle peut offrir en termes de sécurité dudit jeton. En revanche, ces niveaux seront conditionnés par le choix de l'infrastructure organisationnelle, architecturale et le protocole d'authentification qui sera implémenté.

Les techniques en termes de protection de la vie privée utilisées dans ces solutions sont différentes, mais le déploiement et l'implémentation sont très complexes, bien que ceux-ci dépendent du type de la solution considérée. Les conséquences en termes de coût et de risques

relatifs à la protection de la vie privée et à la sécurité doivent être mesurées et traitées avec beaucoup d'attention car les enjeux d'usurpation d'identité sont extrêmement importants.

Cependant, l'utilisation des niveaux LoA et LoT, est intéressante comme moyen pour affiner le contrôle des accès aux ressources sensibles et évaluer le processus d'authentification. Cela peut permettre aux fournisseurs de services d'établir leurs propres décisions de contrôle des accès basé sur le LoT et le LoA pour les consolider avec les décisions d'autorisation.

L'interopérabilité prévue par les spécifications SAML et WS-Trust, peut résoudre une grande partie des problèmes, mais il n'est pas évident que ces spécifications puissent résoudre tous les problèmes de transposition des informations d'identification entre les domaines dans une fédération. Il est à noter que la fédération d'identité présente également des défis complexes pour résoudre les problèmes techniques et les besoins utilisateurs.

Nous avons souligné l'importance de SAML 2.0 dans les Frameworks étudiés : Shibboleth qui est devenu un standard "de facto" dans les réseaux universitaires et une large adoption de Liberty Alliance dans les institutions privées (commerciales) et publiques. Quant à OpenID, il a suscité beaucoup d'intérêt, malgré ces défauts, en particulier chez les fournisseurs de services qui suivent de près les approches Web 2.0 et les gouvernements qui souhaitent inclure activement les gens dans leurs réseaux sociaux et dans des programmes E-Gov. Il a reçu le support de Microsoft (après l'abandon de Cardspace en 2011 et l'échec de MS Passeport), Google, AOL, Yahoo, Orange, Wikipedia, LiveJournal, etc.

Tout va désormais dépendre de ce que ces acteurs majeurs de l'Internet décideront. Google et Yahoo l'ont déjà intégré dans leur système de gestion des identités et offrent des services de type OpenID. Actuellement, les infrastructures et les applications E-Gov, tels que ceux du gouvernement américain, se basent sur des technologies ouvertes, notamment OpenID et Higgins. Certes, le langage SAML est considéré comme une technologie ouverte, mais elle est lourde et complexe, elle nécessite des relations de confiance préétablies entre les IdPs et les SPs, cela rend cette technologie non extensible aux applications Web 2.0.

3.8) Conclusion et premiers choix de l'infrastructure SecFuNet

Les différentes solutions explorées, ne prétendent nullement à dresser un inventaire exhaustif. Mais visent à décrire des facettes suffisamment différentes afin d'en faire ressortir, pour chacune, les avantages et les inconvénients, ou les briques communes.

Le comparatif (Tableau 2), résume et donne une vision intéressante des divergences fonctionnelles de ces solutions.

Tableau 2: Comparaisons des Infrastructures d'authentification et d'autorisation

	Shibboleth	Liberty Alliance	OpenID	Cardspace	Higgins
AAA	AAx	AAx	Axx	Axx	Axx
Compatibilités	SAML	WS SAML		WS	SAML
Approches	Coopérative	Fédérative	User-Centric / Fédérative	User-Centric	User-Centric
Exigences LoA	X	X			
Vie privée / Anonymat et pseudonyme	X	X		X	X
Révocation	X	X			
Authentification unique (SSO)	X	X	X	X	X
Single Logout (SLO)	X	X			
Référentiel d'identités	LDAP	LDAP	URI		
Mécanismes de sécurité	XML Signature	TLS	HTTPS	WS-Trust	WS-Trust/SAML

Les infrastructures basées sur le langage SAML (Shibboleth, Liberty Alliance, etc.) sont de bons candidats pour être adoptées comme système de gestion des identités dans le cadre du projet SecFuNet. Elles intègrent les modèles de gestion des identités et présentent plusieurs caractéristiques intéressantes pour SecFuNet comme les mécanismes avancés de protection de la vie privée, avec un intérêt particulier pour le LoA.

L'interopérabilité garantie par l'utilisation des assertions SAML permet de s'interfacer avec différentes technologies d'authentification et de fédération, le rendant très mature et intéressant pour les environnements hétérogènes. Paradoxalement, cette maturité qui constitue sa force, devient un gros handicap pour lui, le rendant assez lourd et complexe, pour une implémentation massive et simple dans des entités plus modestes, lesquelles préféreront le déploiement d'une technologie plus facile et moins complexe.

En outre, et malgré les avantages de SAML et de certaines solutions étudiées, il existe des limitations pour satisfaire les exigences de SecFuNet. En particulier, le support de la carte à puce qui est limité ou inexistante. Le standard SAML est basé sur des piles protocolaires complexes, qui sont difficiles à implémenter dans des clients légers. Compte tenu de la capacité des éléments de sécurité (e.g. carte à puce), elles seront encore plus difficiles à

embarquer dans ceux-ci. Par conséquent, cette complexité nous amène à privilégier plus simplement des technologies plus légères et des scénarios plus réalistes à l'image d'Higgins ou d'OpenID.

Le protocole d'authentification et les échanges d'attributs dans OpenID sont extrêmement simples à implémenter. En effet, OpenID s'intègre facilement dans une grande variété d'applications et peut être combiné avec différents types de solutions d'authentification, pour satisfaire au moins une partie des exigences de SecFuNet. Le protocole d'authentification TLS peut être une des solutions pour apporter des réponses satisfaisantes à ses défauts et à son faible niveau LoA. Il permet d'éviter les attaques de type phishing et d'usurpation d'identité, en particulier lors de la phase d'authentification et des échanges de cookies. L'autre solution complémentaire, à mettre en place pour préserver la vie privée des utilisateurs, consisterait à implémenter un système de gestion des identités centré sur l'utilisateur. Selon nos analyses et nos investigations précédentes, la combinaison d'OpenID et d'un modèle équivalent à Higgins semblerait une solution plus adéquate à notre projet, puisque ces deux systèmes sont indépendants, interopérables, simples à déployer, et moins complexes en termes de développement. Ce modèle équivalent à Higgins pourrait donc donner satisfaction à l'utilisateur pour s'authentifier sur différentes technologies en gérant efficacement ses propres identifiants et attributs.

Ces choix s'inscrivent donc entièrement dans nos orientations, suite à l'ensemble des analyses et les conclusions évoquées précédemment. Ils nous amènent donc à nous intéresser en profondeur à des protocoles d'authentification qui reposent sur le protocole TLS et des éléments de sécurité, qui s'imposent comme modèle de gestion des identités centré sur l'utilisateur. Ce modèle "User-Centric", s'appuiera nécessairement sur l'utilisation des cartes à puce. Par conséquent, ces cartes à puce nous permettront d'augmenter les niveaux LoA et LoT d'OpenID.

PARTIE II : Propositions d'architecture

Chapitre IV : Proposition d'une architecture globale de gestion des identités.

4.1) Introduction

Comme nous l'avons souligné précédemment, l'Internet du futur s'appuiera fortement sur le Cloud Computing et la virtualisation. Par conséquent, l'un de ses principaux défis est d'offrir des ressources virtuelles et des accès aux Clouds avec un haut niveau de sécurité.

L'objectif de ce chapitre est de décrire notre proposition qui concerne l'infrastructure d'authentification et d'autorisation sécurisée basée sur des standards et des protocoles d'authentification reconnus. Un aspect particulièrement novateur dans notre proposition est lié à l'utilisation des microcontrôleurs sécurisés. La combinaison de ces derniers avec OpenID et Higgins, permettra à notre système global de gestion des identités de construire des domaines administratifs et d'offrir des approches coopératives ou fédératives. De cette manière, la session d'authentification obtenue à partir d'un serveur d'authentification, peut être utilisée, durant sa validité, dans des réseaux virtuels ou pour accéder aux différentes ressources dans un autre Cloud. Un des objectifs ambitieux de SecFuNet est de démontrer et d'expérimenter ces propositions de gestion des identités afin de parvenir à une solution standard pour identifier les utilisateurs et les nœuds dans l'architecture SecFuNet.

Notre infrastructure d'authentification globale offre deux modèles de gestion des identités : un modèle de gestion d'identité pour les services et les nœuds physiques ou virtuels, et un autre modèle centré sur l'utilisateur (User-Centric Identity).

Le premier modèle de SecFuNet est découpé en quatre couches (Figure 19) : les utilisateurs, les services applicatifs, les plateformes virtuelles et les nœuds physiques. Pour chaque entité, un système de gestion des identités est prévu pour coopérer avec des éléments sécurisés locaux ou une grille distante d'éléments sécurisés, laquelle est gérée par un protocole RACS (Remote APDU Call Secure Protocol) qui a fait l'objet d'un Draft à l'IETF (Juillet 2014). Le deuxième modèle est une approche centrée sur les utilisateurs équipés de deux classes d'éléments sécurisés : EAP-TLS (Chapitre V et VI) et PKCS#11 Smart card.

Cette dernière solution a été proposée par l'équipe brésilienne (Annexe D) partenaire du projet SecFuNet pour utiliser des jetons [1]. Ceux-ci sont utilisés pour établir des canaux TLS sécurisés avec authentification mutuelle forte.

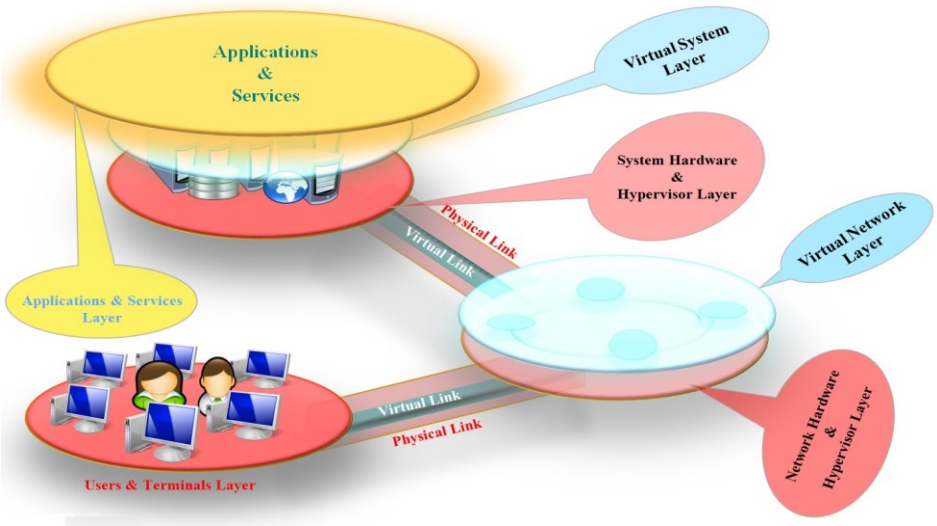

Figure 19 : Les quatre niveaux de visibilité SecFuNet

Avant d'exposer notre proposition d'architecture en détail, nous allons tout d'abord présenter l'architecture de la grille d'éléments sécurisés et les travaux connexes (§4.2) sur lesquels repose notre proposition pour offrir aux usagers des ressources virtuelles, et des accès aux Clouds avec un haut niveau de sécurité. Cette architecture s'appuie sur la virtualisation pour offrir des services dynamiques et flexibles. Après avoir analysé le contexte, et les principaux résultats de l'existant, nous allons ensuite consigner notre proposition d'architecture globale (§4.3) basée sur l'hyperviseur XEN et des briques architecturales et protocolaires (§4.4) qui nous permettront de définir notre domaine administratif (§4.5) pour appliquer une politique de sécurité.

4.2) Travaux connexes

Les choix que nous avons pris dans les chapitres précédents, et qui nous ont amenés à choisir un modèle d'identité qui repose sur l'intégration des microcontrôleurs, s'inscrivent assez naturellement dans notre infrastructure modulaire, puisque les éléments de sécurité peuvent s'insérer facilement dans des modèles de gestion de l'authentification.

Deux classes de microcontrôleurs sécurisés ont été étudiées : les cartes à puce et les TPMs (Trusted Platform Modules) [49]. Ces puces électroniques ont des capacités différentes de calcul informatique : une carte à puce exécute habituellement une machine virtuelle Java (JVM : Java Virtual Machine), elle est donc en mesure d'exécuter des procédures cryptographiques complexes, tandis que les TPMs sont dédiés à des algorithmes RSA.

Cependant, ces dispositifs peuvent être groupés dans des grilles et utilisés pour renforcer la confiance dans les protocoles d'authentification ou pour garantir un stockage sécurisé des clés cryptographiques des ressources dans un environnement virtuel.

4.2.1) Vue d'ensemble de la grille de cartes à puce SecFuNet

Une grille d'éléments de sécurité (ou GoSE : Grid of Secure Elements), est un Module Matériel de Sécurité (HSM : Hardware Security Module). Il s'agit d'un serveur TCP/IP qui héberge un ensemble de microcontrôleurs sécurisés, considérés comme inviolables, qui offrent des fonctionnalités cryptographiques. Un élément sécurisé (ou SE : Secure Element) est aussi appelé carte à puce. Celle-ci est conçue spécifiquement dans un dispositif physique pour gérer le cycle de vie des clés de chiffrement (protection, gestion, traitement et stockage des clefs cryptographiques), grâce à un protocole dédié, embarqué dans la grille qui permet de manipuler les éléments de sécurité à chaque instant et de partout. Leur niveau de sécurité est évalué selon des critères communs [29] [97] ou des normes FIPS-140 [98] [99]. Ils supportent les APIs comme le PKCS#11 ou "Cryptoki" [73].

La grille de cartes à puce de la société IMPLEMENTA (Figure 20), partenaire du projet SecFuNet, peut fonctionner comme un HSM. Malgré le fait qu'elle intègre des ressources informatiques modestes, elle est bien équipée pour répondre aux besoins des applications de SecFuNet qui traitent de petites quantités de données et nécessitent des niveaux de confiance élevés. Elle est constituée d'un multiplexeur pour adresser chaque carte. Elle dispose d'une interface logicielle indépendante pour sa gestion à distance. Le dispositif prend en charge l'accès simultané à un maximum de 416 cartes SIM, utilisé typiquement comme un serveur SIM central dans des applications GSM distribuées. Il est compatible avec la norme ISO/CEI 7816 pour la gestion des cartes à puce.

Au sein de cette grille chaque élément sécurisé a un emplacement physique qui lui fournit des ressources en énergie. Chaque logement est identifié par un attribut SlotID (Slot IDentier) et un SEID (Secure Element Identifier). Le SEID est un identifiant unique indiquant qu'un SE donné est hébergé par une GoSE. Implicitement, il se réfère aussi à l'emplacement physique (SlotID) auquel le SE est intégré. Chaque module (slot) de la grille est identifié par un préfixe (Grid) et un nombre à trois chiffres, à partir de 001. La ligne de commande par exemple : "*READER ? Grid002*" démarre une session avec le slot 002 de la GoSE. Si le

fichier script nommé "scriptdefault.txt" est présent, il est automatiquement chargé et exécuté dans la carte à puce en question.

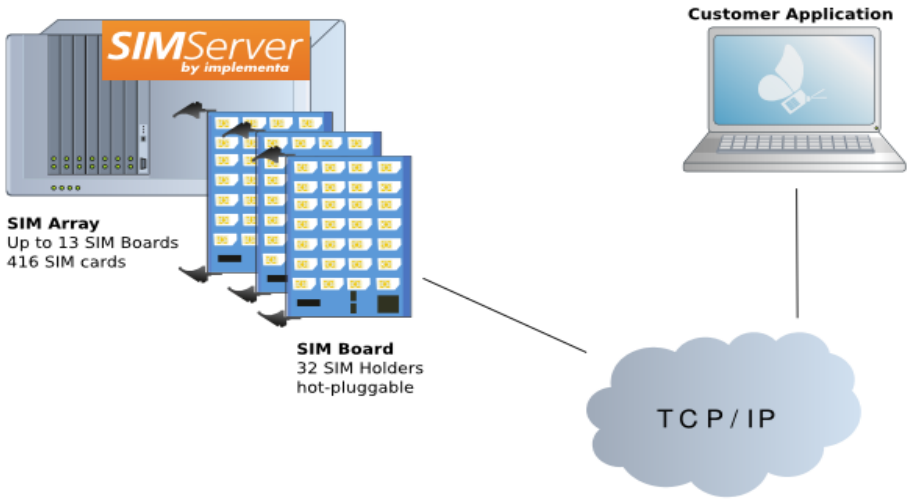

Figure 20 : Grille d'éléments de sécurité

La grille de cartes à puce IMPLEMENTA se compose de plusieurs modules rackables, nommés SIM Array. Un serveur peut gérer jusqu'à 13 modules, chaque module est équipé de 32 cartes à puce. Par conséquent, un seul serveur peut contenir un maximum de 416 éléments sécurisés. Chaque serveur est connecté à Internet, disposant d'une adresse IP et d'un port 51500 par défaut. Le protocole dédié comporte un ensemble de fonctionnalités élémentaires nécessaires pour la gestion et l'inventaire des éléments sécurisés de chaque module. Ce protocole offre des fonctionnalités qui permettent, aussi l'acheminement des commandes ISO7816 APDU (Application Protocol Data Unit) vers la grille. Ces commandes sont écrites en caractères ASCII. Les principales commandes prises en charge par ce protocole sont listées ci-dessous :

- APDU permet d'exécuter une requête sous forme d'APDU ISO 7816.
- CONTENT permet d'obtenir des informations sur les cartes à puce d'une grille d'éléments sécurisés.
- PING permet de contrôler rapidement l'état de fonctionnement de l'unité.
- QUIT permet de terminer une session avec une carte SIM. Le serveur termine et ferme la connexion.
- FREE permet de libérer une carte à puce allouée précédemment.
- RESET permet de réinitialiser et de fermer toutes les connexions des cartes à puce.

- STATUS permet de vérifier l'état de fonctionnement d'une unité et de changer son statut.
- USE permet de réserver et/ou d'allouer une carte à puce pour une utilisation, en lui transmettant ou en lui adressant des commandes ISO7816.

L'objectif initial de SecFuNet est d'effectuer des opérations cryptographiques avec ces cartes à puce enfichables dans cette grille pour gérer l'identité des machines virtuelles, car un module matériel procure un haut niveau de sécurité, en particulier pour protéger les clés de session. Pour satisfaire cet objectif, nous devons développer une nouvelle architecture logicielle pour gérer le cycle de vie des éléments sécurisés, les APIs adaptées pour s'interfacer avec ces derniers, les protocoles de transport des méthodes d'authentification entre les utilisateurs, les applications, les VMs, et les serveurs d'authentification.

L'infrastructure SecFuNet comprend plusieurs composants principaux : le serveur d'authentification, la grille de microcontrôleurs qui embarquent la méthode d'authentification EAP-TLS, sous forme de machine à états (§4.3), conçue par la société EtherTrust, partenaire aussi du projet SecFuNet, et enfin des briques architecturales logicielles pour transporter les messages d'authentification entre les cartes et un serveur distant, permettant ainsi d'établir des connexions sécurisées avec authentification mutuelle basée sur des fonctionnalités PKI.

4.2.2) Architecture EAP-TLS Smart card

Les cartes à puce EAP-TLS [89] se présentent sous forme de deux machines à état (EAP-TLS en mode Client et/ou EAP-TLS en mode Server), avec des méthodes d'authentification TLS différentes [83] [84] [85]. Elles sont capables également de dérouler la norme IEEE 802.1x [95] afin de calculer la clé (MSK) dans des architectures AAA, ou pour établir des canaux sécurisés après une procédure d'authentification mutuelle avec le protocole TLS, basée sur la cryptographie asymétrique. Le logiciel, embarqué dans ces composants, fournit un nouveau concept d'identité universelle pour les écosystèmes (Web et Cloud Computing).

Ces propriétés de sécurité sont directement fournies grâce à ce logiciel embarqué, mais nécessitent des composants logiciels supplémentaires. Toutefois, pour être entièrement cohérents avec nos orientations, il faudra nous assurer que le rajout de ces modules, reste compatible avec les modèles de gestion des identités que nous proposons (i.e. compatible à la fois avec le modèle de gestion d'identité centré sur l'utilisateur, et le modèle fédéré ou

coopératif). C'est pourquoi nous nous attarderons plus particulièrement sur les propriétés des couches logicielles de la carte EAP-TLS Smart card. Nous nous intéressons davantage à la pile protocolaire embarquée dans cette carte à puce [15] [16] et les opérations cryptographiques effectuées par cette machine à états, pour offrir à l'utilisateur la possibilité de gérer pleinement ses données et ses identifiants.

A) La carte à puce et la norme ISO 7816

La carte à puce [17] est un composant électronique qui comporte une CPU, avec des architectures de 8 à 32 bits, une ROM (Read Only Memory) de quelque 100 ko dans laquelle est stocké le système d'exploitation, une RAM (Read Access Memory) de quelque 10 ko, et une mémoire non volatile E²PROM dont la taille varie de 32 ko à 1 Mo. Ce composant est accessible que par code PIN car il contient des informations sensibles (clés cryptographiques) et stocke des applications Java. La carte à puce est alimentée électriquement grâce à un lecteur CAD (Card Acceptance Device) qui de plus la synchronise.

Conformément aux spécifications de la norme ISO/IEC 7816 [96], les échanges d'information avec un terminal peuvent être une liaison série "Half-Duplex" (RS-232) ou bien une interface USB, avec un débit variant entre 9600 bauds à quelques Mbauds. Les paquets échangés sont sous forme d'une suite d'octets entre la carte et le terminal APDU. Selon le protocole de transport utilisé (le protocole T = 0 ou T = 1), la taille de ces dernières peut varier de 256 octets à 65535 octets. L'entête d'une commande APDU comprend cinq octets [89] : CLA, INS, P1, P2 et P3. Ce dernier octet (P3) représente la longueur des données (Lc) pendant une opération d'écriture, mais lorsqu'il s'agit d'une opération de lecture, P3 représente la taille des données attendues (Le) en provenance de la carte à puce. Lorsque la quantité de données échangée dépasse P3 = 256 octets, le standard ISO/IEC 7816 définit des mécanismes de segmentation.

B) Architecture logicielle de la carte EAP-TLS

Dans le chapitre précédent, nous avons évoqué l'usage des cartes à puce par les opérateurs pour sécuriser les réseaux mobiles (cartes EAP-SIM). Cette méthode d'authentification a permis de limiter fortement le nombre de fraudes et une exploitation fiable des réseaux mobiles. Par conséquent, elle a assuré une bonne rentabilité financière aux opérateurs. La technologie EAP-TLS Smart card a fait l'objet d'un standard décrit dans le draft IETF [89]. Il propose que les opérations du protocole EAP-TLS soient traitées directement dans la carte à puce, contrairement aux autres méthodes (e.g. EAP-SIM). Parmi

les méthodes d'authentification qui peuvent être visées, se trouvent EAP-SIM, EAP-AKA et EAP-TLS. C'est sur cette dernière implémentation que nous allons nous focaliser.

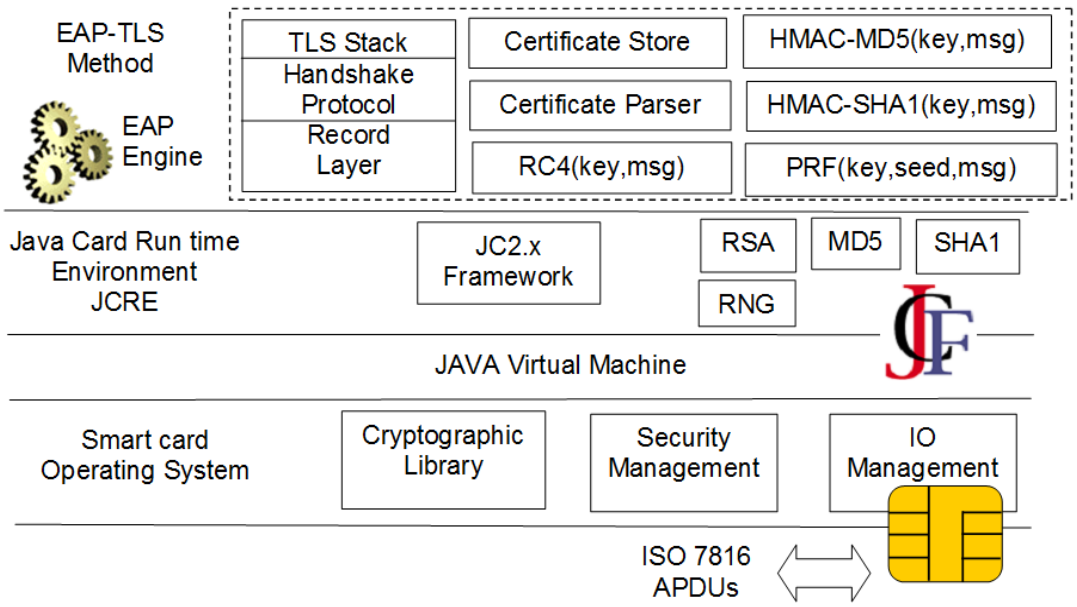

Figure 21 : L'architecture logicielle de la carte EAP-TLS

Comme détaillé au chapitre II (§2.3.4), l'authentification dans une infrastructure RADIUS est établie directement entre le Supplicant et le serveur d'authentification. L'utilisation de la carte EAP-TLS, suppose que le Supplicant doit être doté d'un logiciel de médiation qui joue le rôle de Proxy entre le serveur d'authentification et la carte à puce du poste client ou d'une ressource virtuelle (Chapitre VI). Ce Proxy formate ou encapsule les paquets échangés entre le serveur d'authentification, le client et la carte à puce. Les paquets EAP-TLS échangés entre la carte à puce et le poste client sont transportés par des APDUs ISO7816. L'architecture logicielle de la carte à puce EAP-TLS (Figure 21) [89] [18] est la suivante :

- **EAP-Engine** met en œuvre quatre services fondamentaux (*EAP messages treatment, identity management, security functions, and personalization*) et assure l'acheminement d'EAP vers des méthodes d'authentification prises en charge par la carte.
- **EAP-TLS Method** est une méthode qui gère les mécanismes de fragmentation et de réassemblage des messages TLS.
- **TLS-Stack** est une pile protocolaire qui prend en charge les mécanismes d'authentification, et réalise les opérations de chiffrement et assure l'intégrité des données transportées en toute sécurité par le tunnel TLS.

- **Certificates Store** est un conteneur pour stocker les clés sensibles (clés privées et symétriques, les certificats, etc.).

C) La pile protocolaire de la carte EAP-TLS et le cycle de vie des applications

Le protocole RACS fonctionne au-dessus de la couche de transport TCP et est sécurisé par le protocole TLS (Figure 23). Un client doit être authentifié par un certificat pour envoyer des requêtes RACS. Ce protocole a été récemment introduit [24] [90].

La Figure 22 illustre la pile protocolaire embarquée dans la carte à puce sous forme d'une application Java Card (EAP-TLS ou machine à états), représentée par un ensemble de classes présentes dans un fichier. cap. Elle traite les messages et produit lorsque c'est nécessaire un paquet de réponse.

Cette application possède, conformément au standard ISO7816-5, un identifiant unique nommé AID (Application Identifier), qui se compose d'un RID (National Registered Application Provider) de 5 octets obligatoires, attribué par cette norme ISO et d'un PIX (Proprietary Application Identifier Extension) qui représente un champ facultatif constitué de 0 à 11 octets. L'assignation de ce dernier reste à la charge de l'entité qui a développé l'application Java Card. Pour transmettre une commande à cette application, il est nécessaire de la sélectionner en précisant à la machine Java Card son AID en utilisant une commande APDU spécifique.

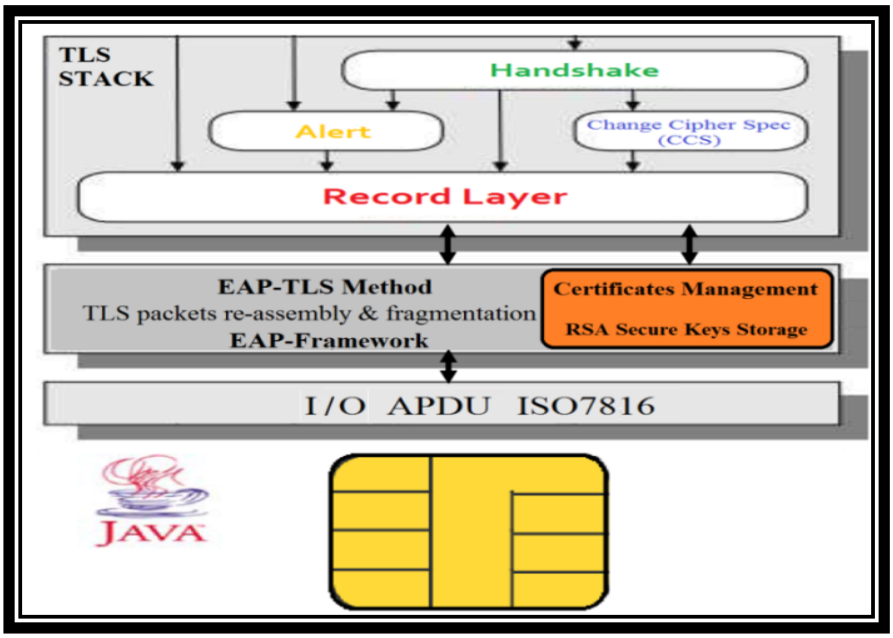

Figure 22 : La pile protocolaire de la technologie EAP-TLS Smart card

L'application EAP-TLS est une implémentation allégée de la méthode d'authentification EAP-TLS en tant que protocole d'authentification embarqué. Cette implémentation conserve du protocole EAP juste le transport des messages TLS en mode datagramme. En effet, cette application réalise toutes les opérations du protocole TLS, en particulier la segmentation et le réassemblage des messages TLS et elle est capable de les analyser pour dérouler les fonctionnalités des sous-protocoles TLS (Handshake, Record Layer, etc.). Elle peut réaliser un ensemble d'opérations : une authentification mutuelle en mode session complète (Full Mode) ou en mode reprise de session (Resume Mode), la vérification du certificat, le chiffrement ou/et le déchiffrement symétrique ou asymétrique et le calcul du MAC (MD5).

Pour s'interfacer avec le poste client, plusieurs commandes APDUs ont été programmées (Chapitre VI) en plus des commandes habituelles qui sont relatives aux lectures/écritures de la mémoire E²PROM, la sélection de l'AID de l'application Java Card, ou la demande de code PIN. L'ensemble de ces commandes a permis de réaliser des opérations efficaces du protocole EAP-TLS implémenté en tant que machine à états (SM : State Machine). Nous citerons en particulier :

- ✓ **Get-Current-Identity** : Retourne l'identité courante d'un client ou serveur, lors d'une session TLS mutuelle,
- ✓ **Set-Identity** : Charge et initialise l'identité devant être prise en considération par l'application Java Card pendant une connexion. Car les identités d'un utilisateur sont déjà chargées dans la carte à puce d'un utilisateur et chaque identité correspond à un conteneur d'identité personnalisable.
- ✓ **Reset** : Réinitialise la machine à états du protocole EAP-TLS,
- ✓ **Process-EAP** : Redémarre la machine à états du protocole EAP-TLS,
- ✓ **Get-Ciphersuite** : Retourne la CipherSuite utilisée,
- ✓ **Get-Key-Block** : Retourne le Key Block.

Contrairement aux postes de travail qui peuvent être vulnérables, la carte à puce peut stocker en toute sécurité une quantité de données sensibles. Elle est inviolable en théorie et par conséquent, les informations qu'elle contient ne peuvent pas en être extirpées. De ce fait, la carte à puce est en mesure de nous offrir une sécurité accrue de nos informations d'identité et une méthode d'authentification mutuelle entre la carte EAP-TLS d'un utilisateur et le serveur distant. En effet, l'interaction directe avec des informations critiques stockées dans la mémoire non volatile E²PROM de la carte, nécessite un code PIN. Ce dernier est demandé à

l'utilisateur à chaque sollicitation d'une zone sensible de la carte. Le cycle de vie des applications embarquées dans ces cartes à puce est géré selon la norme Global Platform (GP) [94]. Il permet les opérations suivantes : téléchargement, activation et/ou suppression.

D)Le protocole RACS

Il définit un ensemble de fonctionnalités de base nécessaires pour un usage distant des éléments sécurisés, par exemple : inventaire des éléments sécurisés, échanges d'informations avec les éléments sécurisés, etc.

Le protocole RACS fonctionne au-dessus de la couche de transport TCP et est sécurisé par le protocole TLS (Figure 23). Un client doit être authentifié par un certificat pour envoyer des requêtes RACS. Ce protocole a été récemment introduit [24] [90].

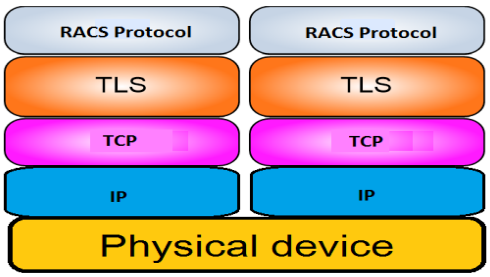

Figure 23 : La pile protocolaire RACS

L'un des principaux objectifs du protocole RACS est d'envoyer efficacement un ensemble de requêtes ISO 7816 [17] [96] vers un élément sécurisé afin d'exécuter des opérations cryptographiques.

4.3) Architecture globale du modèle de gestion des identités SecFuNet :

Comme nous l'avons évoqué précédemment, le Cloud Computing et la virtualisation sont plus que jamais au cœur du système d'information. Les environnements virtuels peuvent héberger une plateforme informatique complète. Ce qui constitue le paradigme IaaS (Infrastructure as a Service). L'Internet du futur s'appuiera fortement sur le Cloud Computing et sur la virtualisation. Par conséquent, l'un des principaux défis pour l'Internet du futur est d'offrir de tels réseaux virtuels et des accès aux Clouds avec un haut niveau de sécurité.

Nos propositions visent à offrir des services d'identification et d'authentification sécurisés aux utilisateurs et aux ressources virtuelles, en s'appuyant fortement sur

l'intégration des éléments de sécurité dans notre contexte virtuel et transorganisationnel. L'intégration de ces technologies se manifestera à tous les niveaux de notre infrastructure (utilisateurs, équipements, VMs, applications, etc.). Ceux-ci nous amènent à faire valoir dans ces propositions, l'idée que la segmentation du Cloud Computing en couches abstraites (horizontales et verticales), permet l'identification des ressources et la séparation des rôles de chaque fournisseur. Cette segmentation peut constituer une première réponse efficace aux problématiques de délimitation du domaine administratif de chaque acteur. Une nouvelle réorganisation de l'infrastructure, dans un environnement virtuel, totalement hétérogène et décentralisé, permettra donc, d'obtenir un maximum de flexibilité, et de mieux piloter dynamiquement le cycle de vie d'une ressource système (identification, authentification, autorisation, approvisionnement, etc.).

La première proposition est essentiellement organisationnelle, consacrée à l'orchestration des ressources et à la représentation des responsabilités des acteurs sous forme de domaine administratif dans le Cloud Computing. Cette nouvelle organisation est structurée en plusieurs niveaux de visibilité abstraits pour faire apparaître les éléments architecturaux impactés.

La deuxième proposition est d'ordre architectural et protocolaire pour gérer la cinématique de l'authentification et de l'autorisation dans notre modèle global de gestion des identités. Ce modèle architectural et protocolaire offre des services d'authentification mutuelle et des services de confiance entre les ressources, basé sur des éléments de sécurité, lequel constituera notre concept TaaS (Trust as a Service).

4.3.1) Le concept TaaS de SecFuNet

À travers ces différents niveaux d'abstraction nous distinguons deux catégories d'entités (Figure 24) : les utilisateurs (SecFuNet Users) et les nœuds SecFuNet.

Les nœuds sont subdivisés en trois niveaux de visibilité, similaires aux paradigmes XaaS du Cloud Computing : infrastructures physiques, environnements virtuels et applicatifs. Les composants de chaque niveau sont identifiés par un certificat. Chacun est associé à un élément de sécurité (i.e. une carte à puce). La première couche infrastructure utilise directement les services d'une GoSE. Les couches supérieures (N) délèguent leur authentification à la couche inférieure (N-1), cette couche possède une identité et une clé

privée stockée dans un élément de sécurité. Une couche N a besoin d'un jeton cryptographique qui établit une liaison logique entre la couche N et N-1.

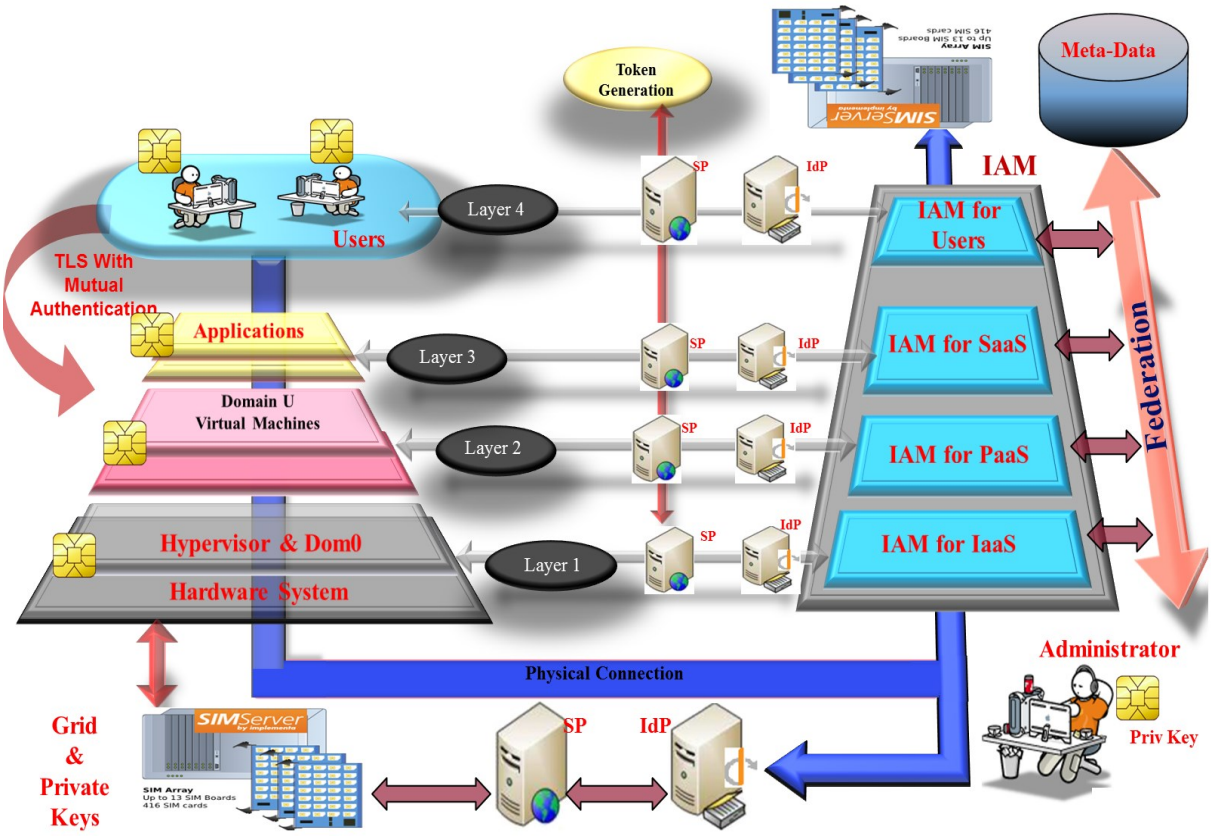

Figure 24 : L'architecture globale du modèle d'identité SecFuNet (TaaS)

L'interaction entre les composants SecFuNet se base sur l'utilisation des identifiants cryptographiques asymétriques et des jetons ; celle-ci implique la mise en place d'une entité racine qui délivre des certificats. Cette autorité de certification (CA) racine est considérée comme le système de gestion des identités et de contrôle des accès (IAM : Identity and Access Management) de chaque niveau de visibilité dans l'infrastructure globale de SecFuNet.

Notre architecture globale est un ensemble d'IAM, qui fournit une séparation claire entre les niveaux de visibilité. Chaque IAM représente un fournisseur de service, elle se décline en quatre niveaux horizontaux : utilisateurs SecFuNet, infrastructures physiques, environnements virtuels et des services applicatifs. Les relations de confiance entre les fournisseurs de service et la coopération entre ces niveaux de visibilité sont assurées par des approches coopératives ou fédératives, basées sur notre modèle TaaS.

Une IAM peut donc être associée à plusieurs niveaux de visibilité horizontale ou transorganisationnelle pour délimiter un domaine de politique de sécurité. Un administrateur système d'une couche (physique, virtuelle ou applicative), n'a de pouvoirs que sur cette couche, il est considéré comme simple utilisateur dans les autres couches. Chaque IAM est interfacé avec un serveur OpenID, et une GoSE pour stocker les outils cryptographiques d'un hyperviseur, d'une VM, ou d'une application.

Un serveur d'authentification OpenID est interfacé avec des fournisseurs de service (SPs), qui peuvent être des VMs ou des applications. OpenID et les SPs sont équipés chacun de certificat fourni par l'entité IAM. Quand un utilisateur souhaite consommer un service par exemple, le SP réclame une authentification du serveur OpenID sous forme de jeton. En fonction du contexte, les jetons sont délivrés directement par l'IAM (GoSE) ou par l'administrateur de la plateforme. Ces jetons sont de différents types pour caractériser l'authentification d'une ressource :

- ✓ un jeton d'authentification pour un Hyperviseur (*HV-Auth-Token : Hypervisor Authentication Token*),
- ✓ un jeton d'authentification pour une machine virtuelle (*VM-Auth-Token : Virtual Machine Authentication Token*),
- ✓ un jeton d'authentification pour une application (*APP-Auth-Token : Application Authentication Token*).

4.3.2) L'identité d'un utilisateur SecFuNet incarnée par un SE

Les utilisateurs sont associés à des certificats, qui incarnent leurs identités (*User-ID*), lesquelles sont stockées dans des éléments sécurisés. Ces éléments de sécurité assurent l'établissement des sessions TLS avec des serveurs distants. Chaque utilisateur détient donc un certificat et une clé privée, lui permettant de réaliser une authentification mutuelle forte. La figure 25 illustre l'établissement d'un canal TLS de confiance entre un utilisateur et un serveur distant. Pour établir une authentification mutuelle forte avec le serveur Web, l'utilisateur a la possibilité d'utiliser deux classes d'éléments sécurisés (EAP-TLS et PKCS#11) :

- ✓ La carte à puce EAP-TLS, gère entièrement le protocole d'authentification TLS (§4.2.2), grâce à des briques logicielles détaillées dans le Chapitre VI.

- ✓ Les cartes à puce PKCS#11, qui stockent et calculent les clés privées, pilotées par des briques logicielles dédiées, généralement dénommé Cryptoki [73].

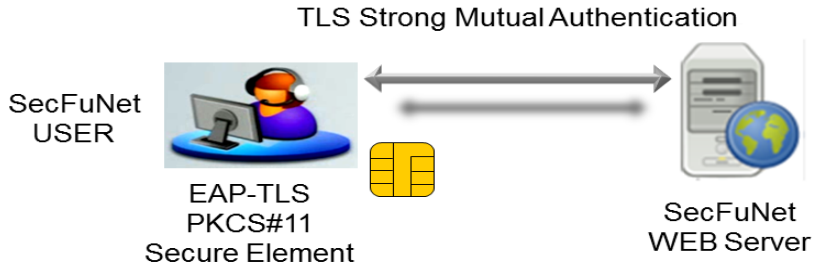

Figure 25 : Utilisation EAP-TLS et/ou PKCS#11 avec un serveur Web

Ces deux classes (EAP-TLS et PKCS#11), peuvent être regroupées dans un même dispositif physique (i.e. carte à puce), sous forme d'une ou deux applications Java Card [17] [23], ces applications nous permettront de mettre en place des approches de gestion des identités centrées sur l'utilisateur "User-Centric Identity" [1].

Dans notre infrastructure SecFuNet, les services sont accessibles via des portails (SPs). Ces fournisseurs de services sont rattachés à un serveur d'authentification OpenID (Figure 26) qui permet de délivrer des jetons cryptographiques selon des mécanismes d'authentification qui seront détaillés dans (Chapitre V).

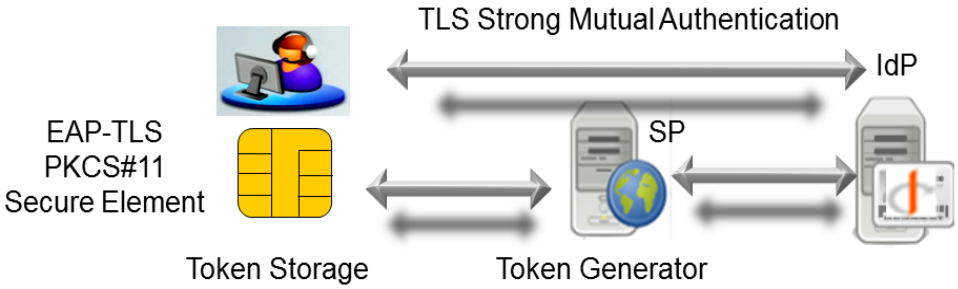

Figure 26 : Génération des jetons cryptographiques pour l'utilisateur SecFuNet

Ces jetons peuvent également être stockés dans des éléments sécurisés pour des architectures de service qui ont besoin d'un modèle de gestion des identités centré sur l'utilisateur [1].

4.3.3) Les nœuds SecFuNet :

Comme indiqué précédemment, les nœuds sont constitués de trois couches principales : infrastructure physique, environnement virtuel et applicatif. Chaque couche est associée à un administrateur système ou à un fournisseur de service.

A) L'infrastructure :

L'infrastructure (Figure 27) comprend des équipements physiques et des hyperviseurs XEN (*HV*) exécutés dans le Domain 0 (Annexe C). Chaque hyperviseur possède un certificat, qui caractérise son identité (ID_{HV}). Il est associé éventuellement, à un microcontrôleur sécurisé qui stocke sa clé privée.

Pour renforcer la confiance et le niveau de sécurité, les sessions TLS avec des authentifications mutuelles fortes entre les HV et les administrateurs sont obligatoires pour toutes les opérations de transfert de machines virtuelles basées sur une interface Web. Les administrateurs sont aussi équipés d'éléments sécurisés.

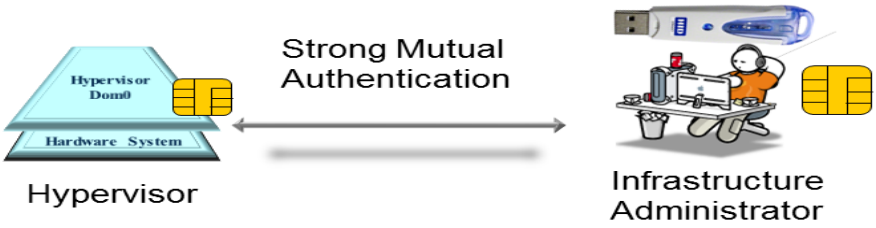

Figure 27 : La couche infrastructure

Toutefois, et pour des raisons de compatibilité, les sessions SSH classiques qui fonctionnent avec une clé privée du côté hyperviseur et Login/mot de passe du côté administrateur peuvent encore être prises en charge. La GoSE peut être reliée directement au serveur XEN ou déportée sur un serveur distant, lequel est administré par un fournisseur de service grille, compatible OpenID ou à une interface WEB classique.

B) L'environnement virtuel :

L'environnement virtuel est constitué de VMs instanciées dans le Domain U (Annexe C). Chaque VM détient un certificat qui incarne son identité (ID_{VM}), sa clé privée est stockée dans un élément sécurisé enfiché dans une GoSE (Figure 28). Toutes les procédures cryptographiques (signature, cryptage ou décryptage symétrique) utilisant les clés secrètes

sont effectuées par l'élément sécurisé lié à la VM, car la protection de ces clés de chiffrement constitue un enjeu crucial pour le modèle de gestion des nœuds SecFuNet.

L'authentification entre la machine virtuelle et un serveur d'authentification OpenID nécessite la clé privée de l'hyperviseur (*HV-Priv*) et le jeton appelé (*VM-Auth-Token*). Ce jeton établit la preuve que la machine virtuelle, identifiée par son certificat ID_{VM}, est rattachée à un hyperviseur identifié par son certificat ID_{HV}.

Conceptuellement, pour instancier une VM, il faut le jeton (*VM-Auth-Token*) signé par l'IAM qui gère le niveau de visibilité où la VM est exécutée. Ce jeton se compose de deux certificats (ID_{VM} et ID_{HV}), d'une date de validité (*Timestamp*) :

$$VM\text{-}Auth\text{-}Token = \{ID_{VM}, ID_{HV}, Timestamp\}Sign_{CA\text{-}IAM}.$$

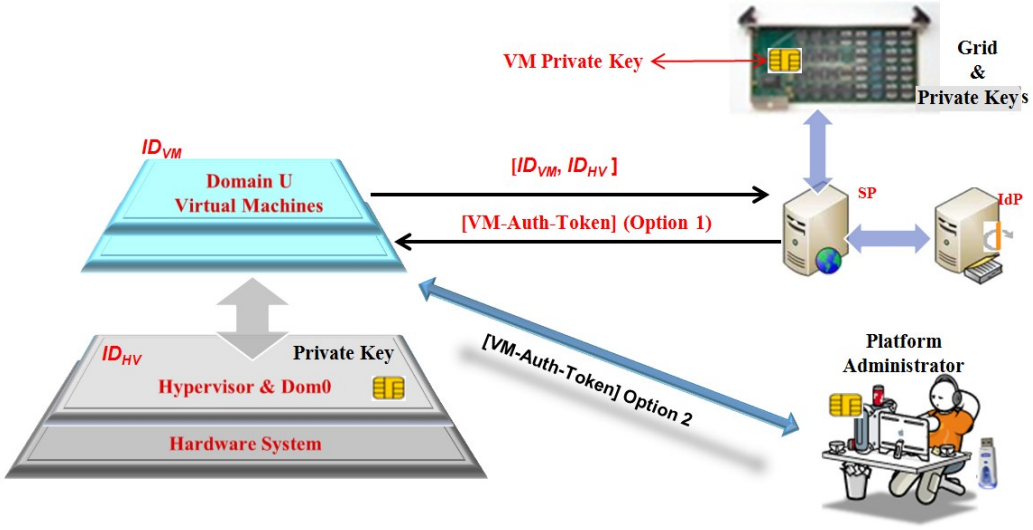

Figure 28 : Processus d'échanges avec la clé privée de la VM dans SecFuNet

Le SP, qui contrôle les services de la grille a besoin de trois éléments, afin d'autoriser les opérations avec l'élément sécurisé lié à une VM : d'abord une authentification réussie avec le serveur d'authentification qui gère le certificat ID_{HV}, le certificat ID_{VM}, et la possession de l'attribut (*VM-Auth-Token*). Ce dernier paramètre pourrait être fourni dynamiquement par le serveur OpenID ou statiquement assignée par l'administrateur du domaine administratif pour le transfert de la machine virtuelle.

C) L'environnement applicatif :

L'environnement applicatif est constitué des applications (APP) ou des services invités qui sont exécutés dans des conteneurs (VMs). Une ou plusieurs applications sont gérées par

des administrateurs dédiés. Une application est identifiée par un certificat (ID_{APP}) et une clé secrète stockée dans la grille d'éléments sécurisés.

De la même manière que l'environnement virtuel, la grille est interfacée avec un SP, qui peut être compatible OpenID ou une interface WEB classique (Figure 29). L'authentification entre une application et un serveur d'authentification OpenID nécessite la clé privée de la VM et le jeton (*APP-Auth-Token*). Ce jeton établit la preuve que l'application identifiée par son certificat ID_{APP} est attachée à la VM identifiée par son certificat ID_{VM}.

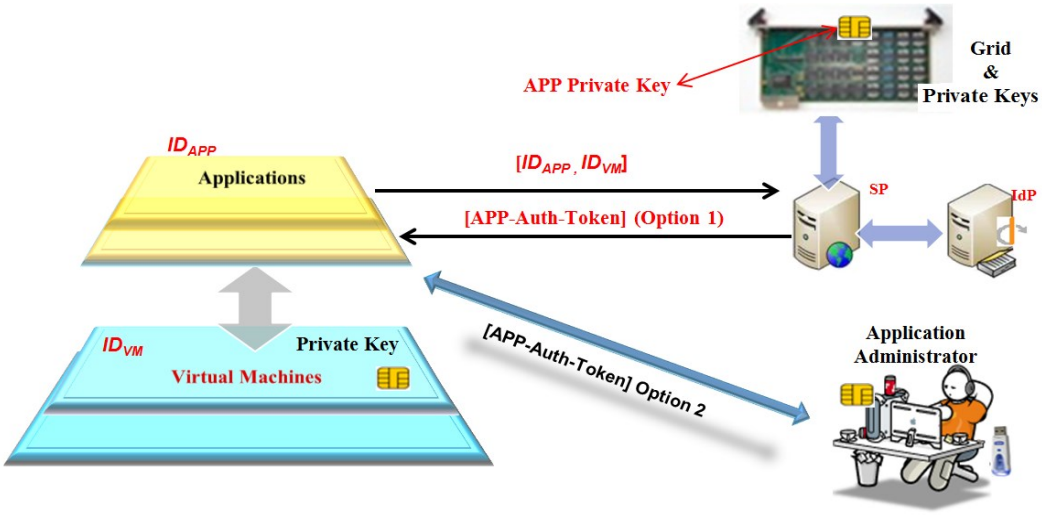

Figure 29 : La couche application

Il y a deux possibilités pour délivrer un jeton : soit il est affecté statiquement par l'administrateur de l'application, soit il est transmis de manière dynamique par le serveur d'authentification OpenID.

Conceptuellement le jeton (*APP-Auth-Token*) est un élément d'information signé (comme le certificat) comprenant les deux certificats (ID_{APP}, ID_{VM}) et une date de validité (Timestamp) : $APP\text{-}Auth\text{-}Token = \{ID_{APP}, ID_{VM}, Timestamp\}Sign_{CA\text{-}IAM}$.

Le SP, qui contrôle les services de la grille, a besoin de trois éléments, afin d'autoriser les opérations avec l'élément sécurisé lié à une application : d'abord une authentification réussie avec le serveur d'authentification qui gère l'ID_{VM}, puis l'ID_{APP}, et la possession de l'attribut (*APP-Auth-Token*).

4.4) Les briques architecturales et protocolaires

Comme évoqué précédemment, les propositions de SecFuNet sont d'ordre organisationnel, architectural et protocolaire. La proposition d'une infrastructure organisationnelle subdivisée en plusieurs couches permet de délimiter les accès directs aux couches systèmes invités et aux environnements virtuels qui doivent coopérer dans un même conteneur. La seconde proposition architecturale consiste donc à associer chaque ressource à un élément de sécurité et à un outil de gestion des identités. Cette association permet de définir le domaine administratif virtuel pour appliquer une politique de sécurité sur une ressource, obtenir une vue d'ensemble d'une infrastructure virtuelle.

Par conséquent, l'approvisionnement de cette dernière devient agile, dynamique et flexible. L'identification des ressources et le contrôle des droits d'accès sont rendus faciles dans une architecture décentralisée.

Cet outil de gestion s'appuiera sur des briques logicielles et protocolaires pour gérer la cinématique de l'authentification entre deux ressources et orchestrer le contrôle des accès dans un environnement virtuel.

Notre architecture se base donc, sur l'attribution à chaque ressource (poste client, VM, XEN, etc.), d'un élément de sécurité qui fait office de preuve de son identité. Dans l'idée de l'identifier et/ou l'authentifier physiquement dans le Cloud en s'appuyant sur la méthode d'authentification du protocole TLS, tout en restant aussi proche que possible des principes de fonctionnement du standard TLS existant.

En effet, les messages de la machine à états (EAP-TLS) parviennent, au serveur d'authentification (ou au poste client), encapsulés dans des paquets EAP. Selon le cas, ces paquets sont interprétés, formatés et ensuite envoyés au serveur sous forme de messages purement TLS, de telle sorte que la communication entre deux entités respecte les normes du protocole TLS et que l'utilisation du protocole EAP-TLS embarqué dans la carte à puce soit transparente pour un serveur TLS normal.

Nous avons implémenté plusieurs couches applicatives pour établir une liaison logique entre la carte à puce et le serveur d'authentification (ou le poste client), mais aussi entre des services Web et la carte à puce. Ces couches applicatives, que nous appellerons "Proxy", doivent s'exécuter en tâche de fond dans une ressource physique ou virtuelle, permettant ainsi le déroulement de la méthode d'authentification TLS embarquée dans la carte à puce au lieu d'exécuter celle implémentée dans la ressource.

Les Proxys jouent donc un rôle crucial dans le fonctionnement du protocole TLS embarqué. Ils sont implémentés dans l'esprit de respecter le protocole TLS pour établir une session sécurisée avec n'importe quel type de serveur TLS qui comporte les trois procédures principales suivantes :

- ✓ ***SSL_connect ()*** : réalise les opérations de la phase I, d'abord une authentification est effectuée entre le client et le serveur (Handshake), puis un bloc de clefs (appelée KEY-BLOCK) est calculé et associé à un ensemble d'algorithmes cryptographiques (identifiés par l'étiquette Cipher-Suite) utilisé par la couche enregistrement (Record Layer) en mode chiffré. Les sessions TLS peuvent être ouvertes de deux manières (Full Mode ou Resume Mode). Dès que la phase I se termine, un canal TLS sécurisé est mis en place et permet d'échanger des quantités de données chiffrées (c'est le début de la Phase II).
- ✓ ***SSL_write () et SSL_read ()*** : ces procédures sont utilisées dans la Phase II. La première procédure chiffre et envoie des données et la deuxième procédure lit et décrypte des données.

Ces proxys agissent comme un pont logiciel entre la carte EAP-TLS et le serveur TLS distant pour acheminer les messages d'authentification entre deux entités. Cette partie sera détaillée au Chapitre VI.

Notre architecture globale (Figure 30), se compose de plusieurs parties distinctes (infrastructure et plateforme) ainsi que des composants matériels et logiciels ; la communication avec la GoSE est réalisée avec une interface PC/SC (§6.3.1).

La partie infrastructure est constituée de ressources matérielles (serveurs, réseaux, GoSE, etc.) et de ressources logicielles (un hyperviseur en l'occurrence XEN version 4.0), un pilote pour gérer la grille de test constituée d'un ensemble de cartes à puce, une console de contrôle des ressources de XEN et un "Proxy Server", lequel est constitué de plusieurs composants (§6.3.2.1). L'hyperviseur est associé à un microcontrôleur sécurisé qui stocke son certificat et sa clé privée. Le certificat incarne donc son identité.

La partie plateforme se compose de ressources virtuelles (Machine Virtuelle, Routeur Virtuel, etc.). Les VMs sont exécutées dans le domaine de privilège (Domaine U). Chaque VM est associée à un microcontrôleur sécurisé qui stocke son certificat et sa clé privée. Le microcontrôleur est une carte à puce déportée qui se trouve dans la grille d'éléments sécurisés.

Pour accéder à son microcontrôleur, la VM doit établir dans un premier temps un tunnel sécurisé TLS avec le Proxy Server, par l'intermédiaire de sa couche logicielle nommée : "Proxy VM" (§6.3.2.2). Ce Proxy joue le rôle de médiateur avec un utilisateur qui veut par exemple s'identifier au niveau de la VM afin de consommer un de ses services. Pour ce faire, le Proxy VM redirige tous les messages d'authentification vers la grille qui héberge la carte à puce de la VM via le Proxy Server, lequel va interagir avec la carte associée à la VM pour négocier, avec l'utilisateur, toutes les clés cryptographiques qui seront stockées dans celle-ci et nécessaire pour l'établissement d'une session TLS.

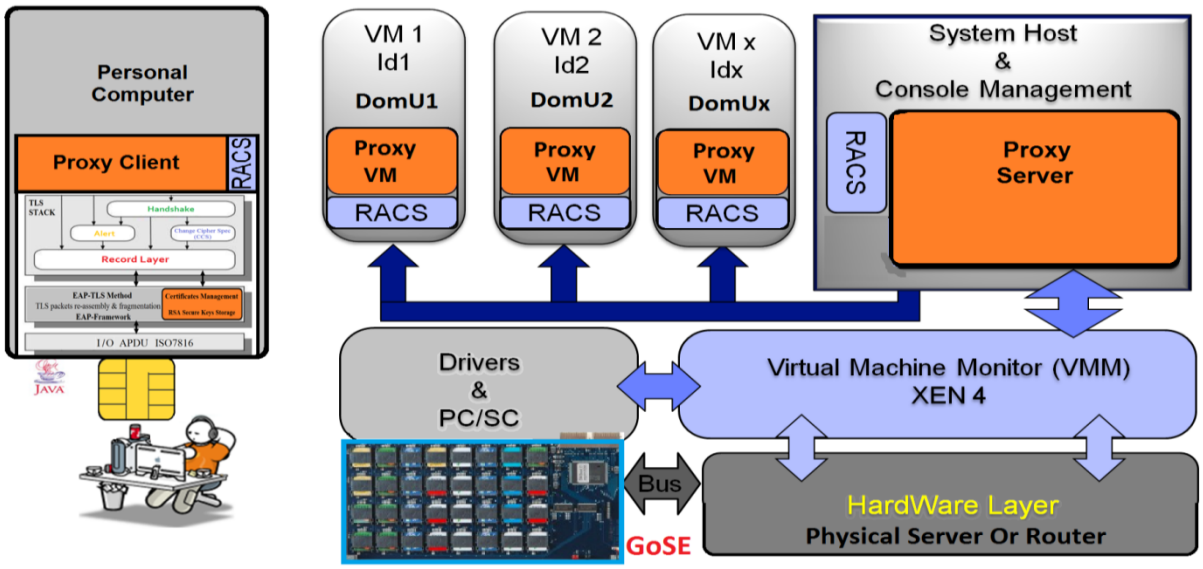

Figure 30 : L'Architecture logicielle globale de SecFuNet

Au niveau client nous avons implémenté une autre couche logicielle nommée "Proxy Client" similaire au "Proxy Server", laquelle interagit avec la carte à puce de l'utilisateur et les ressources SecFuNet. Le Proxy Client est alors capable d'envoyer des instructions à la carte à puce pour dérouler une connexion TLS avec une ressource.

L'acheminement des protocoles d'authentification entre les ressources, se fait de deux manières, soit en utilisant des procédures implémentées dans ces Proxys, soit en utilisant un nouveau protocole, récemment introduit, nommé : (RACS : Remote APDU Call Secure Protocol) [24] [90]. Ce protocole fonctionne au-dessus de la pile TLS, fournissant des commandes de base et faciles pour accéder à une GoSE pilotée par des protocoles propriétaires. Ce protocole sera détaillé dans le Chapitre VI (§6.4).

4.5) La notion organisationnelle d'un domaine administratif de SecFuNet

Un des objectifs de SecFuNet est de démontrer et d'expérimenter ces propositions de gestion des identités afin de parvenir à une solution standard pour identifier les utilisateurs et les nœuds dans l'architecture SecFuNet. L'utilisation des microcontrôleurs sécurisés et les Proxys permettent de construire des domaines administratifs, indépendamment de la couche où se trouve une ressource (Figure 31).

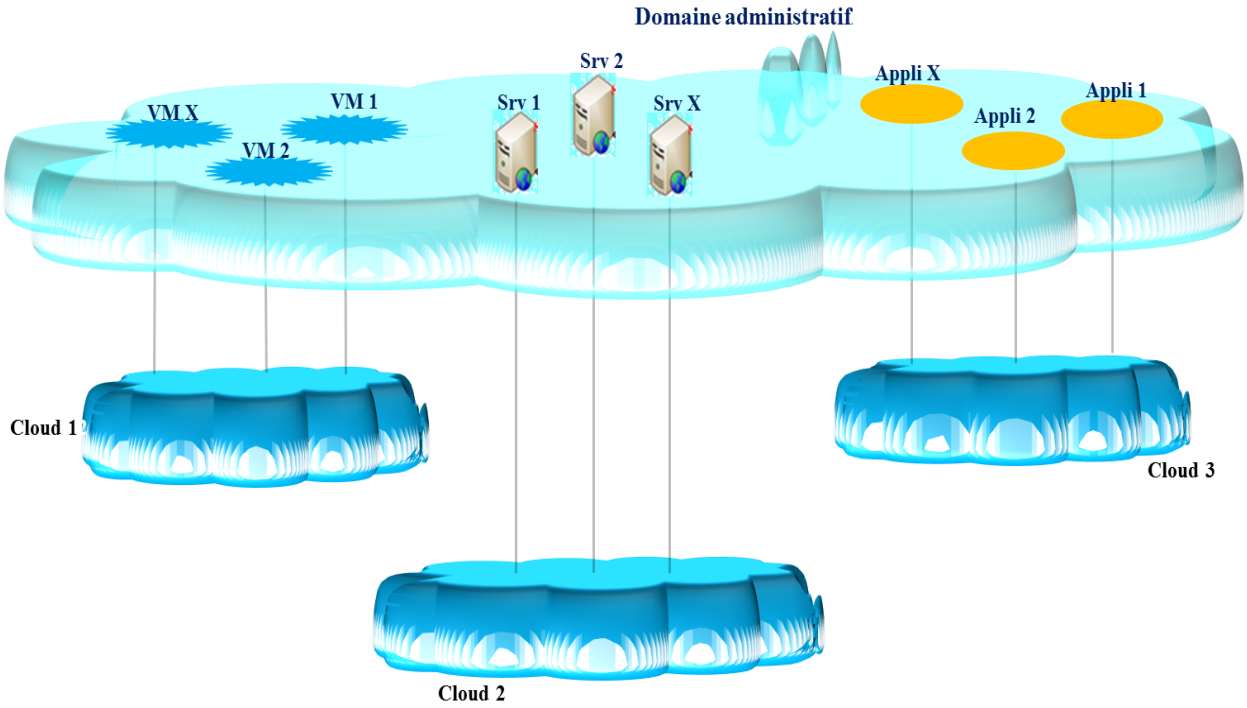

Figure 31 : Notion d'un domaine administratif dans SecFuNet

En d'autres termes, un domaine administratif est un système de gestion des identités qui regroupe plusieurs ressources (physiques ou virtuelles) de plusieurs Clouds, délivre des certificats à chaque élément de sécurité associé à une ressource SecFuNet. Pour construire le lien logique entre ses ressources, il s'appuie sur des Proxys (§4.4). Ces derniers assurent la transversalité entre les niveaux de visibilité et délimitent la zone d'application de la politique de sécurité. Par conséquent, l'administrateur système d'un domaine administratif, n'a de pouvoirs que sur ce dernier, il est considéré comme simple utilisateur dans les autres.

4.6) Conclusion

Après avoir analysé le contexte, et les principaux résultats de l'existant, nous avons consigné, dans ce chapitre, nos propositions qui ont pour objectives de répondre aux besoins

d'identification des ressources dans notre infrastructure indépendamment de leur couche de visibilité, la gestion de leurs sessions de bout en bout et celles des utilisateurs. Nos propositions visent donc à offrir des services d'identification et d'authentification sécurisés. Ces propositions ont nécessité la prise en compte de tous les niveaux : utilisateurs, équipements, VMs et applications.

En effet, pour faciliter la gestion de nos ressources avec un maximum de dynamicité et de flexibilité, nous avons fait des propositions d'ordre organisationnel, architectural et protocolaire. La nouvelle organisation subdivisée en quatre niveaux de visibilité, permet d'illustrer de manière simple et générique tous les éléments qui composent chaque niveau. Elle permet ainsi de faciliter l'identification des ressources dans des architectures distribuées, favoriser les approches de coopération ou de fédération entre les acteurs de ces architectures selon leur rôle.

La gestion de l'authentification repose ainsi sur le protocole TLS embarqué dans les éléments sécurisés et les briques architecturales qui pilotent la médiation et la cinématique des messages d'authentification entre les ressources et la GoSE.

Ces briques architecturales et protocolaires nous offrent plusieurs possibilités de déploiement de ces composants de sécurité dans notre infrastructure et nous permettent de délimiter nos domaines administratifs pour appliquer une politique de sécurité ou offrir des services de confiance. Les grilles peuvent être associées à des domaines administratifs différents, supervisées par une autorité de confiance indépendante pour offrir des services TaaS. Ces grilles peuvent être facilement intégrées dans un modèle d'identité pour établir des relations de coopérations ou de fédérations à l'image du standard OpenID.

L'intégration de ces éléments de sécurité n'a de sens qu'avec un déploiement des briques architecturales dans chaque composant de l'architecture OpenID. Cet aspect sera détaillé dans le chapitre V, pour finaliser la description de notre infrastructure globale de gestion des identités et fournir ainsi une preuve de concept de notre modèle de gestion des identités.

En outre, nous avons proposé une couche protocolaire RACS (§6.4) pour assurer une interaction plus flexible et sécurisée entre les différentes ressources de l'infrastructure et faciliter la gestion à distance et la personnalisation d'un élément de sécurité associé à une entité. Ce protocole permet l'approvisionnement d'un élément de sécurité distant (Identifiant, Certificat, Clés symétrique, Jeton, etc.) et la négociation d'une authentification d'une manière

transparente entre deux ressources pendant la phase d'initialisation d'une session (Full Mode), ou aussi la renégociation d'authentification lors de la phase de consommation d'un service (Resume Mode).

Chapitre V : Gestion de l'authentification dans SecFuNet

5.1) Introduction

Il existe plusieurs façons de réaliser une authentification entre l'utilisateur et le serveur OpenID. La plus populaire est l'utilisation d'un login/mot de passe simple. Malheureusement ce mécanisme comporte plusieurs risques de sécurité : attaque de phishing (hameçonnage), dans lequel l'utilisateur est redirigé vers un SP malveillant qui l'invite à entrer son mot de passe sur un faux formulaire imitant son IdP [25] [45].

Une autre vulnérabilité particulière du standard OpenID est celle dans laquelle le RP procède à une redirection du client vers le serveur OpenID. En l'absence de relation digne de confiance entre l'OP et le RP, ce dernier pourrait être un site malveillant et redirigerait le terminal automatiquement vers un OP pour soustraire le mot de passe de l'utilisateur.

Pour limiter ces risques, plusieurs techniques sont à l'étude. L'une d'elles et la technique anti-hameçonnage qui consiste à demander à l'utilisateur de choisir une image, au moment de la création de son compte auprès d'un serveur OpenID. À chaque tentative d'authentification, ce dernier devra montrer cette image secrète, partagée entre l'utilisateur et son OP. Une autre technique consisterait à supprimer ou à limiter la redirection, dans ce cas l'utilisateur sera obligé de saisir l'URL de son OpenID. Malheureusement, ces techniques rendraient le déploiement d'OpenID moins attractif et son utilisation plus difficile.

Pour réaliser une authentification mutuelle, notre modèle de gestion des identités proposé dans le cadre de SecFuNet s'appuie sur un serveur OpenID pour construire ces relations de confiance entre les utilisateurs, les IdPs et les SPs, ainsi que sur des clés cryptographiques asymétriques stockées dans des microcontrôleurs, en particulier les cartes à puce EAP-TLS.

Ce chapitre est consacré à l'intégration de ces cartes à puce dans une architecture OpenID, comme une alternative pour remplacer le couple (Login/Mot de passe) et pour faire barrière aux attaques de phishing, en se basant sur la technologie AJAX.

À présent nous allons nous attarder, dans cette section, sur un rappel du fonctionnement de la technologie AJAX (§5.2) et l'imbrication des éléments de sécurité avec OpenID pour authentifier un utilisateur (§5.3). Le téléchargement des jetons dans une carte à puce (§5.3.5) sera illustré à travers un scénario. Ensuite, nous allons dérouler des opérations

d'authentifications avec le serveur OpenID et la grille d'un fournisseur de serveur (§5.4) pour démontrer notre concept (Trust as a Service) dans l'infrastructure SecFuNet.

5.2) Rappel sur la technologie AJAX

AJAX (Asynchronous JavaScript and XML) [46], est un ensemble de technologies pour exécuter des applications sur le navigateur WEB, déclenché par les interactions de l'utilisateur, typiquement, des clics de souris associés à des formulaires HTML et traités par des procédures JavaScript. Les fonctionnalités AJAX (Figure 32) sont indispensables pour échanger des données avec un serveur distant.

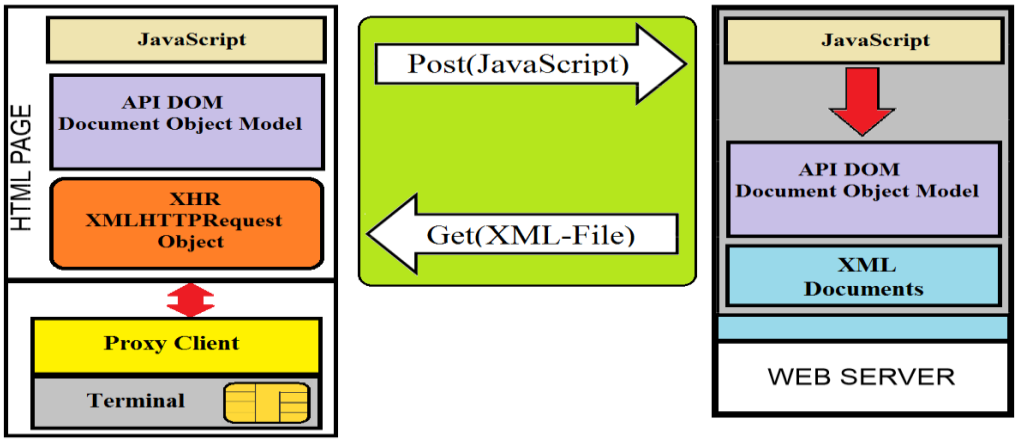

Figure 32 : Architecture d'AJAX et le logiciel du Proxy Client

En effet, les requêtes JavaScript asynchrones peuvent être envoyées vers un serveur Web pour modifier dynamiquement des valeurs dans un formulaire Web courant sans que la page Web soit rechargée entièrement ou redirigée vers une autre. Ceci est un avantage qui offre un gain de temps dans les réponses et une meilleure interaction entre le client et le serveur. Ces requêtes JavaScript asynchrones sont assurées par l'objet XMLHttpRequest (XHR). Cet objet a été implémenté avec les mêmes méthodes que celle d'XMLHTTP de Microsoft. XMLHttpRequest est un objet JavaScript qui permet de récupérer des informations au format XML, en particulier, à l'aide de requêtes HTTP. La version actuelle de l'objet XMLHttpRequest, présente quelques inconvénients qui résident, en particulier, dans le fait qu'il ne peut offrir qu'une seule connexion à un serveur Web distant qui utilise le protocole HTTP (ou HTTPS).

Par conséquent, cet objet s'avère intéressant pour effectuer des opérations de téléchargement de jetons dans une carte à puce. Les pages AJAX sont téléchargées par le navigateur du poste client, lequel exécute notre Proxy Client (Figure 32), qui est exécuté en tâche de fond sur le terminal de l'utilisateur, pour assurer la médiation entre un serveur Web et la carte à puce de l'utilisateur.

Le Proxy Client utilise donc un port spécifique, généralement le port 8080, ouvert sur l'adresse (Loopback : 127.0.0.1) sous forme de : *http ://127.0.0.1:80*80. L'interface du Proxy Client est composée de deux types de classe d'adresses URL, ces classes sont utilisées pour les échanges réseaux entre un serveur Web et l'élément sécurisé EAP-TLS.

La **Classe I** - http ://127.0.0.1:8080/~url = www.server.com/path/file.html, elle ouvre une session HTTPS au serveur distant, équivaut à : https ://www.server.com/path/file.html. La session TLS est démarrée à partir de l'élément sécurisé EAP-TLS, puis transférée vers le démon. En conséquence, le navigateur ouvre une session HTTPS avec le serveur distant qui est authentifié via la carte à puce.

La **Classe II** : http ://127.0.0.1:8080/reader/apdu.xml, elle envoie des commandes APDUs ISO7816 à l'élément sécurisé EAP-TLS. Ces commandes sont localisées dans le corps d'une requête HTTP POST. Elles sont codées comme contenu en entrée dans un formulaire XML sous le format suivant : *= APDU1 & = APDU2 & = APDUi*.

Ces commandes, sous forme de requêtes AJAX, sont envoyées simultanément par le navigateur de l'utilisateur après un traitement par le Proxy. Effectivement, les requêtes AJAX visent à atteindre la carte à puce du poste client et non le serveur d'identités, par l'intermédiaire du Proxy Client, qui écoute sur l'adresse et le port (127.0.0.1 : 8080), lequel redirigera les requêtes vers la carte à puce. La carte traitera ces ARDU et ensuite toutes les réponses seront transmises au Proxy Client qui les formatera sous forme de fichier XML ou HTML. Le Proxy s'appuie sur une API DOM (Document Object Model), utilisée dans les formulaires HTML et XML, pour fournir une structure de représentation de la page, qui permettra de modifier le contenu ou la présentation visuelle de celle-ci. En d'autres termes, l'API permet de relier les pages Web à des programmes JavaScript.

5.3) OpenID et la carte EAP-TLS

Comme détaillé (Annexe B.3), OpenID est constitué de trois entités, le site de consommation (SP) nécessitant l'authentification de l'utilisateur, le serveur d'authentification (OpenID) pour authentifier l'utilisateur, et un terminal IP de l'utilisateur. L'identifiant OpenID est une URL qui comprend deux parties : le nom du serveur OpenID et l'alias de l'utilisateur. Une association de sécurité est statiquement ou dynamiquement établie entre le client et le serveur d'authentification en signant numériquement une preuve de l'identité de l'utilisateur, mais à condition que ce dernier autorise le serveur OpenID à dévoiler son identité. Cette preuve est une clé secrète partagée entre ces deux entités. Elle est utilisée pour l'authentification du message. Les mécanismes d'authentification utilisés dans OpenID ne sont pas spécifiés par le standard. Il peut utiliser soit les traditionnels mots de passe, soit des certificats X509.

Dans le contexte SecuFuNet, le modèle de gestion des identités sera composé d'OpenID et une GoSE (e.g. EAP-TLS) afin d'établir des authentifications mutuelles fortes basées sur des sessions TLS exécutées dans ce périphérique. Le client et le serveur seront ainsi identifiés par leur certificat X509. L'interaction, entre les serveurs (OpenID, SP) et le poste client, est basée sur la technologie AJAX (Asynchronous JavaScript and XML). Par ailleurs, il est nécessaire d'interagir avec la carte à puce via le poste client, en particulier quand il s'agit d'une demande de connexion sécurisée avec un serveur Web distant, en utilisant le navigateur de l'utilisateur.

5.3.1) Définition d'un conteneur d'Identité ou TLS-Identity

La carte à puce EAP-TLS embarque une pile complète TLS, un module de gestion des certificats X509, un environnement sécurisé pour le stockage des clés et l'exécution des procédures cryptographiques.

Schématiquement une application EAP-TLS offre quelques services de base : la gestion des identités multiples stockées dans des "Conteneurs d'Identité", la personnalisation et l'association d'une identité à la carte, en fonction du réseau visité ou du domaine administratif d'affectation, le traitement des messages EAP, le calcul des clés unicast et Key-block. À la fin d'une session d'authentification TLS, le poste client dispose, à chaque instant,

grâce à cette application, de ces clés durant une session, pour qu'il puisse ensuite les dériver en d'autres clés de session.

Un Conteneur d'Identité définit un format d'identité noté "*TLS-Id : TLS-Identity*" pour personnaliser la carte en lui associant des données correspondant à une d'identité (cliente, et/ou serveur) et un nom à ces identités.

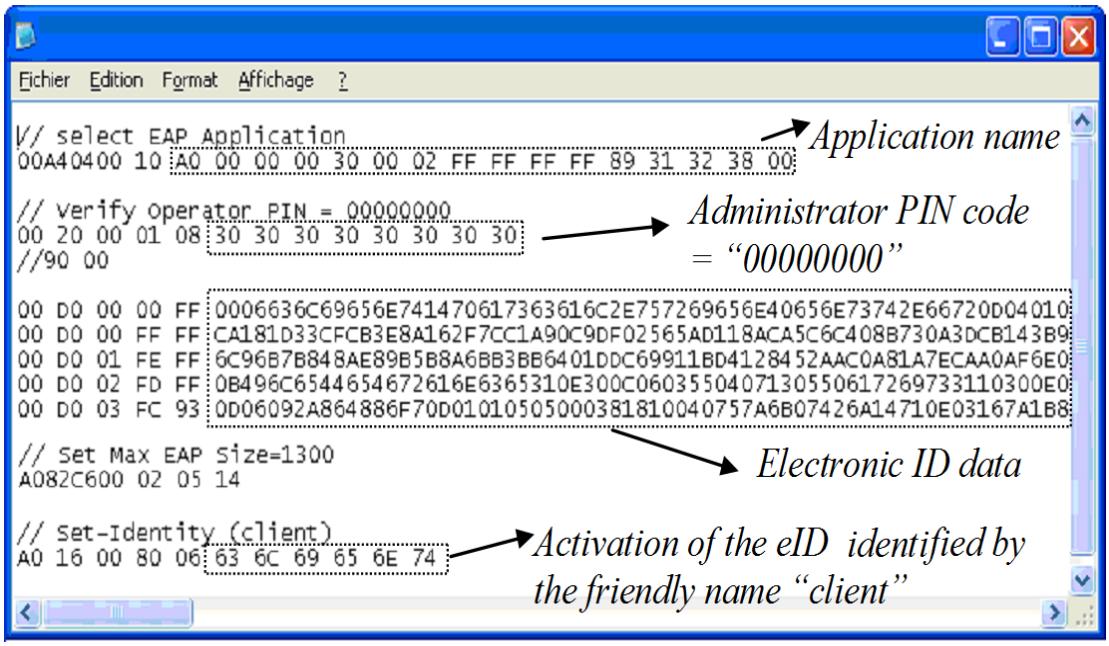

Figure 33 : L'identité TLS (TLS-Identity)

Comme illustré (Figure 33), ces paramètres sont stockés dans la mémoire non volatile E²PROM du microcontrôleur sécurisé au cours du processus de personnalisation, et maintenus par l'administrateur ou l'utilisateur SecFuNet. Chaque TLS-Id nécessite un ensemble de paramètres d'authentification pour définir un format d'identité, qui est constitué des champs suivants :

- ✓ Le type de l'identité (Client et/ou Serveur) et le nom associé à cette identité,
- ✓ EAP-ID : est le nom associé à EAP-Identity, c'est une valeur délivrée dans le message "EAP-Response. Identity",
- ✓ EAP-Type : est le type de protocole d'authentification supporté par le réseau (EAP-MD5, EAP-SIM, EAP-TLS, etc.),
- ✓ L'ensemble des clés cryptographiques utilisé par la méthode d'authentification, c'est-à-dire l'ensemble des paramètres nécessaires à l'établissement d'une session TLS (secret partagé, certificat X509 et sa clé privée associée à l'identité, etc.),

- ✓ Le certificat de l'autorité de certification (CA), utilisés par un protocole particulier (EAP-SIM, EAP-TLS, etc.),
- ✓ Un pseudonyme, utilisé pour identifier et activer un TLS-Id donné.

5.3.2) Principe d'authentification d'un utilisateur dans OpenID

Dans le contexte OpenID, une demande d'authentification est alors envoyée par le SP au navigateur. Grâce au mécanisme de redirection HTTP, ce message est transmis au fournisseur OpenID. La figure 34 détaille une requête d'authentification délivrée par la SP. Il s'agit d'un ensemble de paramètres d'une requête, laquelle est incluse dans un format codé en HTML. La requête contient des éléments qui identifient l'association de sécurité précédemment établie entre le SP et le serveur OpenID.

```
<html><head><title>OpenId transaction in progress</title></head>
<body onload='document.forms[0].submit();'>
<form accept-charset="UTF-8" enctype="application/x-www-form-urlencoded" id="openid_message"
action="http://192.168.2.44/openid/examples/server/server.php" method="post">
<input type="hidden" name="openid.ns" value="http://specs.openid.net/auth/2.0" />
<input type="hidden" name="openid.ns.sreg" value="http://openid.net/extensions/sreg/1.1" />
<input type="hidden" name="openid.ns.pape" value="http://specs.openid.net/extensions/pape/1.0" />
<input type="hidden" name="openid.sreg.required" value="nickname" />
<input type="hidden" name="openid.sreg.optional" value="fullname,email" />
<input type="hidden" name="openid.pape.preferred_auth_policies" value="" />
<input type="hidden" name="openid.realm" value="http://192.168.2.35:80/openid/examples/consumer/" />
<input type="hidden" name="openid.mode" value="checkid_setup" />
<input type="hidden" name="openid.return_to" value="http://192.168.2.35:80/openid/examples/consumer/finish_auth.php?
janrain_nonce=2013-03-17T09:07:14ZGoN7wY" />
<input type="hidden" name="openid.identity" value="http://192.168.2.44/openid/examples/server/server.php/idpage?user=pascal" />
<input type="hidden" name="openid.claimed_id" value="http://192.168.2.44/openid/examples/server/server.php/idpage?user=pascal" />
<input type="hidden" name="openid.assoc_handle"
value="{HMAC-SHA1}{49bf5e64}{Va0OCQ==}" />
<input type="submit" value="Continue" />
</form>
<script>var elements = document.forms[0].elements;for (var i = 0; i < elements.length; i++) { elements[i].style.display = "none";}
</script></body></html>
```

Figure 34 : Une requête d'authentification OpenID

Le SP ajoute quelques paramètres standards : "*openid.identity*" ou "*openid.return_to*". Ce dernier doit être protégé contre le rejeu, en incorporant un identificateur de session unique : *"janrain_nonce = 2013-03-17T09 : 07 : 14ZGoN7wY".*

Selon la terminologie des normes Web-Services le message de demande d'authentification OpenID est interprété comme une demande de jeton. Dans le contexte SecFuNet, il est utilisé pour l'authentification et pour la génération des jetons [1].

5.3.3) Authentification de l'utilisateur dans SecFuNet

Dans OpenID, l'identité d'un utilisateur est associée à une adresse URL, l'utilisateur se connecte au fournisseur de service (SP) d'un site Web, où il décline son identité et indique le service sollicité. Le SP détermine à partir de l'identité de l'utilisateur, le nom du serveur OpenID auquel il est rattaché, démarre une procédure, compatible au standard OpenID, de découverte (XRI) avec le fournisseur d'identité et collecte l'adresse XRDS (Extensible Resource Descriptor Sequence) du serveur.

Ces deux entités échangent leurs descripteurs de ressources encodés selon le format XRDS. Ensuite le serveur et le consommateur (SP) exécutent le protocole classique de Diffie-Hellman (DH) [26] avec leur clé publique, afin de calculer un secret symétrique partagé, utilisé pour renforcer l'intégrité du message grâce à une procédure HMAC (RFC 2104, 1999).

Le serveur d'authentification OpenID reçoit la demande d'authentification, comme un message POST de HTML émis par le terminal de l'utilisateur. Il retourne par la suite un formulaire HTML (Figure 35). À ce stade, cette page de connexion est protégée par une session TLS et la session est gérée par la carte EAP-TLS, car chaque utilisateur SecFuNet est équipé d'un élément sécurisé.

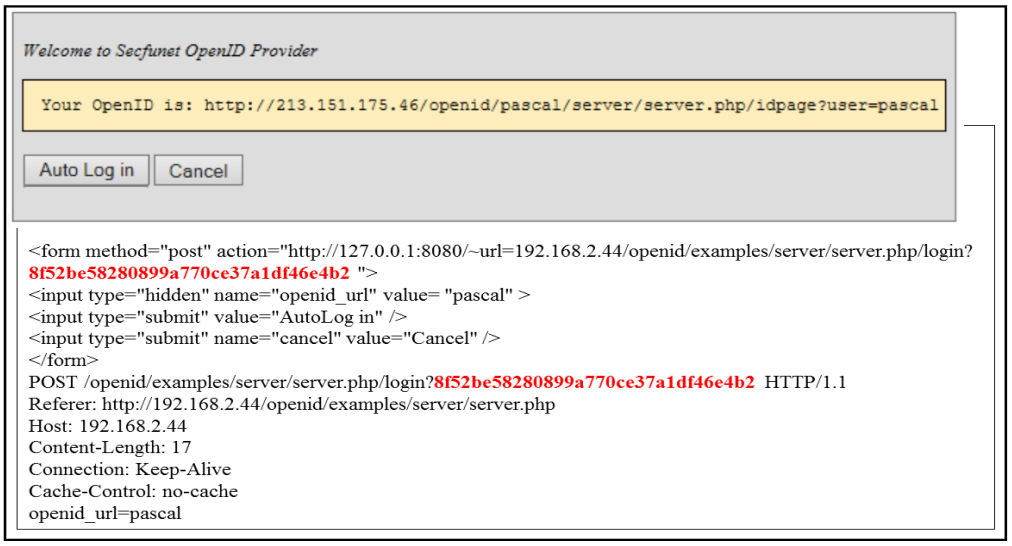

Figure 35 : Formulaire OpenID d'authentification d'un utilisateur

En effet, dans une implémentation classique d'OpenID, l'utilisateur doit saisir son mot de passe dans un formulaire pour être identifié, dans le contexte SecFuNet ce mot de passe, est remplacé par une session TLS avec authentification mutuelle, initiée par l'élément sécurisé

EAP-TLS. La page de connexion est associée à un en-tête HTTP, incluant un cookie, dont la valeur est l'identificateur de session (*sid : session identifier*), et l'URL associé comprend également la valeur suivante : *http ://127.0.0.1:8080/~url = server.com/login.php?sid*

Lorsque l'utilisateur clique sur le bouton AutoLogin (Figure 35), Il ouvre via son Proxy, le lien : *https//server.com/login.php?sid*. Par conséquent, l'utilisateur est authentifié par son certificat X509. La réponse HTTP à cette requête fixera le cookie (*sid*) pour l'adresse du Proxy (qui fonctionne en tâche de fond avec l'adresse « loopback » : 127.0.0.1 : 8080). Le *sid* dans notre exemple est égal à la valeur suivante :

Sid = 8f52be58280899a770ce37a1df46e4b2.

Un deuxième échange optionnel (Figure 36), se produit entre le fournisseur et le navigateur, sécurisé avec le jeton EAP-TLS, pour confirmer l'association avec le site du consommateur. Cet échange est utilisé dans le contexte SecFuNet pour télécharger des attributs dans l'élément sécurisé EAP-TLS [1].

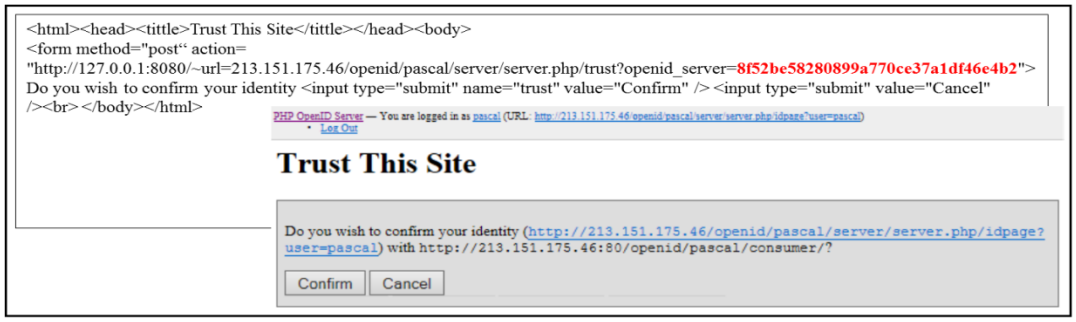

Figure 36 : Page optionnelle de confiance

5.3.4) La réponse d'une Authentification

Après une authentification réussie, la réponse d'authentification est renvoyée par le serveur OpenID vers le terminal de l'utilisateur. Elle est ensuite redirigée vers le site de consommation, qui délivre enfin une page d'accueil. La Figure 37 détaille le message de la réponse d'une authentification délivré par le SP. Il s'agit d'un ensemble de paramètres de réponse inclus dans l'emplacement d'en-tête MIME. L'intégrité des données est renforcée par le champ : (HMAC-SHA1 = *{**49bf5e64**}{Va0OCQ ==}*), associé à la clé secrète OpenID, calculée à partir de l'échange DH précédent.

```
HTTP/1.1 302 Found
Date: Sun, 17 Mar 2013 09:07:28 GMT
Server: Apache/2.2.6 (Win32) DAV/2 mod_ssl/2.2.6 OpenSSL/0.9.8g mod_autoindex_color PHP/5.2.5
X-Powered-By: PHP/5.2.5
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Thu, 19 Nov 1981 08:52:00 GMT
location: http://192.168.2.35:80/openid/examples/consumer/finish_auth.php?
janrain_nonce=2013-03-17T09:07:14ZGoN7wY
&openid.assoc_handle={HMAC-SHA1}{49bf5e64}{Va0OCQ==}
&openid.claimed_id=http://192.168.2.44/openid/examples/server/server.php/idpage/user=pascal
&openid.identity=http://192.168.2.44/openid/examples/server/server.php/idpage/user=pascal
&openid.mode=id_res
&openid.ns=http://specs.openid.net/auth/2.0
&openid.ns.sreg=http://openid.net/extensions/sreg/1.1
&openid.op_endpoint=http://192.168.2.44/openid/examples/server/server.php
&openid.response_nonce=2013-03-17T09:07:28Z2Z6LCL
&openid.return_to=http://192.168.2.35:80/openid/examples/consumer/finish_auth.php/
janrain_nonce=2013-03-17T09:07:14ZGoN7wY
&openid.sig=HUBr4K7Ff51FHCywZFux2IcZ9Xg=
&openid.signed=assoc_handle,claimed_id,identity,mode,ns,ns.sreg,op_endpoint,response_nonce,return_to,signed,sreg.email,sreg.fullname,sr
eg.nickname
&openid.sreg.email=pascal@secfunet.com
&openid.sreg.fullname=Pascal+Urien
&openid.sreg.nickname=pascal
Connection: close
Content-Length: 0
Content-Type: text/html
```

Figure 37 : Une réponse d'authentification OpenID

5.3.5) Téléchargement du Token Cryptographique dans la carte

Dans une approche « User-Centric Identity » [1], les attributs sont représentés par les jetons stockés dans la carte à puce. Le but de cette section est de détailler comment un SP dédié peut être utilisé pour télécharger des jetons dans la carte à puce EAP-TLS ?

Comme décrit dans les sections précédentes, l'utilisateur SecFuNet est équipé d'un élément de sécurité EAP-TLS (Figure 38). Afin de collecter un jeton, il effectue une connexion avec le SP. Ce dernier le redirige vers son serveur OpenID, lequel lui envoie une page de connexion. L'utilisateur s'authentifie avec son élément sécurisé. L'interaction, entre le serveur OpenID et ce dernier, est basée sur la technologie AJAX et le logiciel Proxy Client.

Les réponses retournées sont codées selon un format XML par le Proxy. Les pages AJAX retournées par le serveur OpenID contiennent les objets XMLHttpRequest qui envoient, au Proxy Client, des requêtes HTTP de Classe II (§5.2). Ces requêtes contiennent des commandes ISO7816, qui seront formatées, et acheminées vers l'élément sécurisé, puis elles seront exécutées par ce dernier.

En effet, durant la procédure d'authentification, le serveur OpenID collecte le certificat de l'élément de sécurité, ou la clé publique pour créer un conteneur et négocier des clés cryptographiques. Ces clés sont utilisées pour transmettre en toute sécurité un jeton ou pour le charger dans l'élément sécurisé.

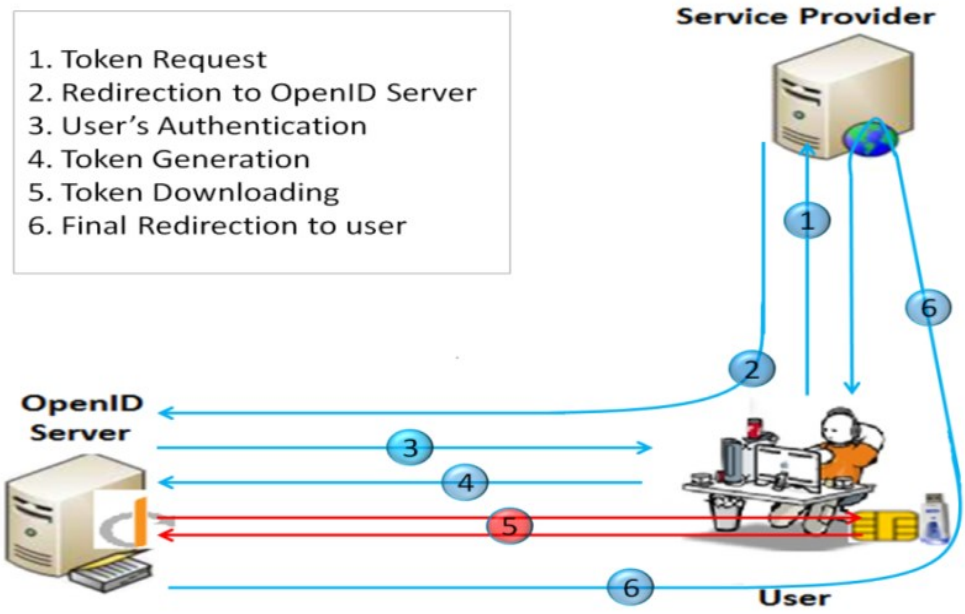

Figure 38 : La plateforme OpenID pour générer et télécharger les jetons

Grâce à ces mécanismes, le serveur OpenID charge le jeton dans l'élément de sécurité en utilisant un ensemble de commandes ISO7816 intégré dans les pages AJAX, et réalise des opérations nécessaires pour le téléchargement du conteneur (Figure 39).

Ensuite, la signature du conteneur est vérifiée ; la clé AES est récupérée à partir de la clé privée EAP-TLS, le jeton est ensuite déchiffré et stocké dans la carte à puce [27].

Ce conteneur se compose de trois parties :

- ✓ Header : un en-tête qui est la valeur chiffrée de la clé symétrique AES, avec la clé publique de l'élément sécurisé EAP-TLS, selon la norme PKCS#1.
- ✓ Body : un corps qui est la valeur chiffrée du jeton, selon une procédure AES-CBC.
- ✓ Trailer : c'est une valeur signée par une autorité de confiance, de l'en-tête (Header), concaténée au corps (Body), selon la norme PKCS#1, cette autorité est identifiée par sa clé publique.

Dans certains cas, le SP peut être autorisé à générer des jetons pour une famille d'éléments sécurisés. Dans ce cas de figure, l'utilisateur SecFuNet est directement authentifié par une session TLS avec une authentification mutuelle forte avec le site Web du fournisseur de services. L'utilisateur interagit avec le serveur du SP en téléchargeant des pages Web sécurisées par des sessions TLS. Le mécanisme décrit précédemment, basé sur des pages

AJAX, est utilisé pour télécharger des informations (stockées dans le conteneur) de l'élément sécurisé.

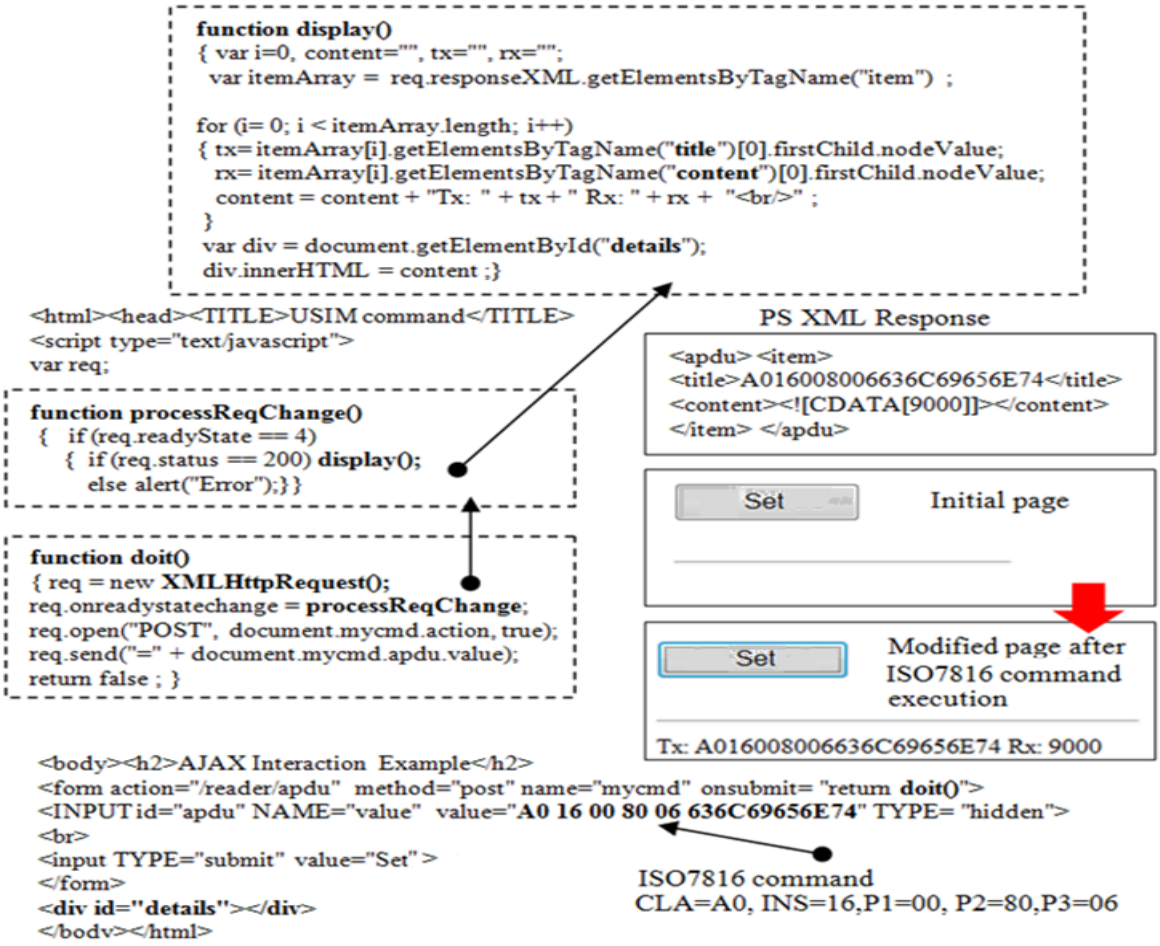

Figure 39 : Interaction entre EAP-TLS et OpenID en utilisant AJAX

5.4) Opérations entre OpenID et la GoSE

Comme mentionné précédemment dans le Chapitre IV, chaque VM est associée à un élément de sécurité, intégré dans une grille distante, qui stocke en toute sécurité la clé privée de la VM ; celle-ci possède un identifiant à savoir un certificat (ID_{VM}), et un jeton établissant le lien avec l'hyperviseur qui l'héberge. Ce dernier est identifié lui aussi par un identifiant ID_{HV}. Afin d'utiliser sa clé privée à distance, la VM doit interagir avec le SP qui s'interface avec la grille pour s'authentifier en utilisant le certificat du ID_{HV}, la clé privée de l'hyperviseur et le jeton (*VM-Auth-Token*).

Deux scénarios, sont pris en considération, pour réaliser des opérations avec la grille : l'utilisation d'un serveur d'authentification OpenID (§5.4.1), l'utilisation de l'interface Web classique (§5.4.2).

5.4.1) Le serveur d'authentification OpenID comme TaaS

La machine virtuelle peut accéder à la clé privée de l'hyperviseur stockée dans un élément sécurisé (Figure 40). Elle établit une session avec le fournisseur de services (étape 1) et fournit : (*ID_{HV}, ID_{VM} et VM-Auth-Token*). Elle peut également demander un ensemble d'opérations cryptographiques (chiffrement ou déchiffrement) avec sa clé privée.

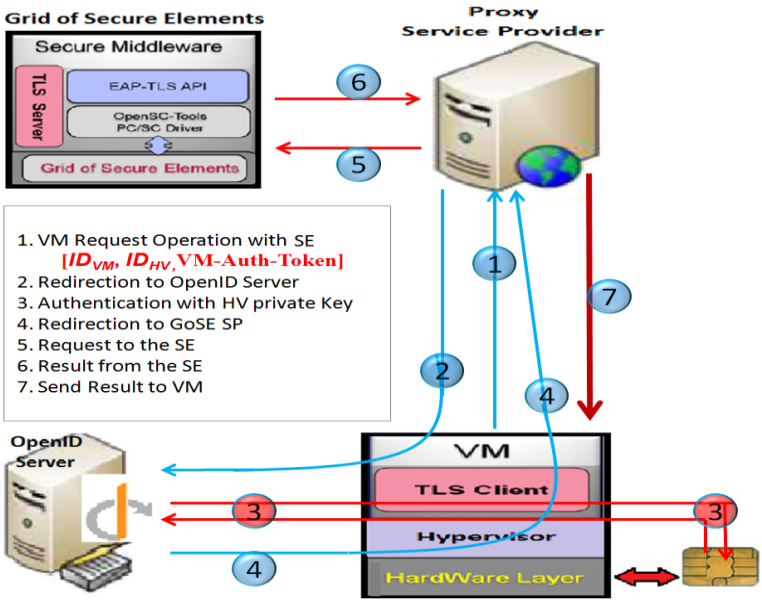

Figure 40 : Le TaaS dans le contexte OpenID

Elle est ensuite redirigée vers le serveur OpenID (étape 2). Le processus d'authentification avec l'élément sécurisé de l'hyperviseur (étape 3) s'exécute. Le serveur OpenID, authentifie l'hyperviseur et redirige les informations d'authentification vers le SP (étape 4). À cette étape, le fournisseur de services (SP) sait que l'entité dispose d'un accès à distance à la clé privée HV, grâce aux informations d'identification appropriées dans le jeton (*VM-Auth-Token*).

La VM demande par exemple, via le SP, un ensemble d'opérations cryptographiques (chiffrement ou déchiffrement) avec sa clé privée (étape 5). La carte à puce de la VM réalise les opérations demandées (étape 6) et le SP retourne le résultat à l'étape 7.

Afin d'obtenir un niveau de sécurité plus élevé, la session entre la machine virtuelle et le fournisseur de services doit être une session TLS. Ce fournisseur peut encapsuler le résultat dans un conteneur (§5.3.5). Certains problèmes de confiance peuvent exister entre le SP qui contrôle la grille et la VM, puisqu'ils ne partagent pas nécessairement la même autorité de confiance. Dans ce cas le conteneur pourrait être directement généré par l'élément sécurisé de la VM, précédemment lié à un ID_{HV}. Cette opération est effectuée par un élément sécurisé de l'administrateur grâce à des mécanismes de la plateforme globale.

5.4.2) Authentification avec une grille SP

Le serveur OpenID n'est pas sollicité quand le SP est capable de vérifier l'identité de l'hyperviseur (Figure 41). En d'autres termes une session TLS avec une authentification mutuelle forte est réalisée entre la grille SP et l'hyperviseur distant.

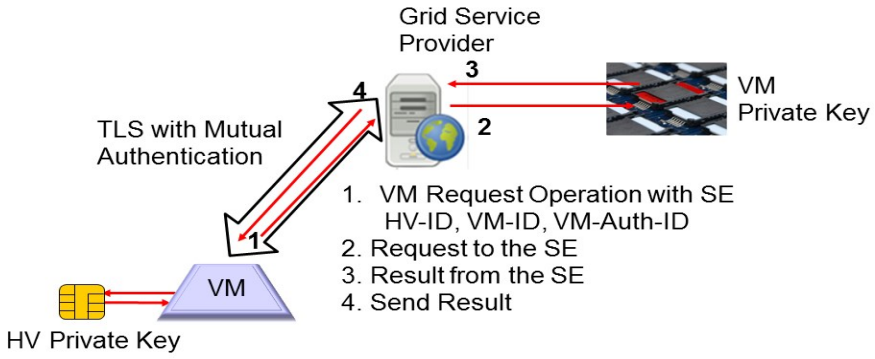

Figure 41 : Fournisseur de service de grille avec un serveur Web classique

La requête HTTP contient des paramètres chiffrés (ID_{HV}, ID_{VM} *et VM-Auth-Token*), et l'opération qui doit être exécutée par l'élément sécurisé. Les informations d'autorisation de la VM sont vérifiées. En cas de succès, l'opération est exécutée par l'élément sécurisé qui retourne un résultat.

Selon le niveau de confiance (LoT) avec le SP qui contrôle la grille, ce résultat peut être dans un format clair ou encapsulé par l'élément sécurisé dans un conteneur sécurisé, comme décrit dans la section précédente.

5.5) Conclusion

Dans l'optique d'offrir une solution crédible aux défauts relevés dans l'état actuel du standard OpenID et des modèles de gestion des identités, la carte EAP-TLS s'impose comme modèle alternatif au couple (Login/mots de passe) pour parer certaines vulnérabilités.

Elle réunit de nombreux avantages, grâce à son architecture relativement simple, qui s'intègre facilement dans le standard OpenID, lequel tire le meilleur parti de l'environnement de confiance offert par ce composant sécurisé. Ce composant embarque un protocole d'authentification mutuelle, capable de réaliser des calculs cryptographiques, de gérer des identités multiples, de stocker des clés cryptographiques ou des jetons dans des conteneurs sécurisés. Ces conteneurs d'identités définissent un format d'identité pour personnaliser la carte à puce associée à un utilisateur. L'utilisateur dispose ainsi de ses identifiants et de ses attributs pour les utiliser selon son désir dans un modèle de gestion des identités afin d'établir par exemple une authentification mutuelle forte.

La cinématique d'authentification entre la carte à puce et le standard OpenID est assurée, d'une part, grâce à l'utilisation de la technologie AJAX et le Proxy Client pour éviter les contraintes d'implantation de cette carte au niveau du poste client, et d'autre part grâce à un des standards extrêmement répandu qu'est le protocole TLS embarqué sous forme de machine à états dans la carte à puce.

Il nous reste alors à implémenter nos briques logicielles et à les intégrer de manière plus globale dans notre architecture de gestion des identités pour gérer le cycle de vie de ces composants, stocker les clés cryptographiques et les différentes identités.

La troisième partie de ce mémoire est consacrée précisément, au développement de ces logiciels architecturaux pour répondre aux besoins de notre système de gestion tout en démontrant que les propositions sont viables et que les tests de nos solutions dans des situations réelles ont des performances acceptables.

Ces briques architecturales seront détaillées dans le chapitre suivant, afin d'aboutir à une implémentation de l'architecture complète fournissant une preuve de concept de notre modèle architectural de gestion des identités.

PARTIE III : Implémentations.

Chapitre VI : Implémentations des différentes couches applicatives.

6.1) Introduction

Les serveurs d'authentification constituent généralement, un point de faiblesse dans la chaîne de sécurité d'un système d'information. Notre objectif est de développer un serveur d'authentification hautement sécurisé basé sur une grille de microcontrôleurs sécurisés. Les équipes du sous-projet (Infrastructure of the authentication Server), ont spécifié un serveur d'authentification relié à plusieurs solutions matérielles, selon deux approches basées sur une grille de microcontrôleurs sécurisés et sur une grille de TPMs ou de TPMs virtuels.

Cependant, pour atteindre cet objectif un ensemble de fonctionnalités, nécessaires pour gérer le cycle de vie de ces composants doivent être implémentées. Par exemple l'administration des cartes à puce de la grille distante nécessite différents types de fonctionnalités pour gérer les identités, charger des jetons, personnaliser ces cartes à puce avec des clés cryptographiques, installer de nouvelles applications Java Card, etc.

Ce chapitre est consacré aux développements et aux implémentations des briques architecturales (§6.3), pour gérer la cinématique des messages d'authentification entre les ressources (utilisateurs, virtuelles, serveurs, etc.) et un élément de sécurité qui embarque la machine à états (EAP-TLS), lequel nous assurera des opérations d'authentification mutuelle forte, durant tout le processus d'échanges de communication de bout en bout. Nous démontrerons ainsi l'utilité de l'ensemble de nos propositions et leur viabilité à travers des illustrations sous forme de scénarios (§6.4), dans un contexte opérationnel. Ensuite nous testerons les performances de cette technologie (§6.4.3), pour démontrer que les temps de réponse sont acceptables pour une utilisation régulière par l'utilisateur final. Ensuite, nous aborderons l'implémentation du protocole RACS (§6.5) qui nous offre toutes les fonctionnalités nécessaires pour un usage distant des éléments sécurisés. À présent, nous allons détailler la norme PC/SC (§6.2), sur laquelle se base notre architecture logicielle pour communiquer avec les éléments sécurisés.

6.2) Les APIs OpenSC et la norme PC/SC

Chaque grille de cartes à puce s'appuie sur un protocole dédié pour offrir des fonctionnalités permettant le transport des commandes ISO7816. Les principales commandes

prises en charge par ce protocole dans la grille de cartes à puce (IMPLEMENTA) sont listées au Chapitre IV (§4.2.1).

Ces commandes sont constituées d'un en-tête de cinq octets (nommé CLA, INS, P1, P2, P3) et d'un corps en option. Ces commandes APDUs sont écrites en caractères ASCII pour s'interfacer avec des applications comme par exemple EAP-TLS ou GP.

Le serveur de cartes SIM IMPLEMENTA possède un pare-feu d'APDUs. Ce dernier bloque ceux non autorisées pour des raisons de sécurité. La Figure 42 montre une configuration typique du pare-feu dans le contexte SecFuNet : le préfixe (r) indique les opérations en lecture et le préfixe (w) indique les opérations en écriture.

EAP-TLS Application	Global Platform Facilities
cla A0 ins 18 r 'get current identity' cla A0 ins 17 r 'get next identity' cla A0 ins 16 w 'set identity' cla A0 ins 19 w 'reset' cla A0 ins 80 w 'process EAP' cla A0 ins A6 r 'get session key' cla F0 ins 2A w 'verify key' cla 80 ins E4 w 'delete application' cla 80 ins E6 w 'install for load' cla 80 ins E8 w 'load' cla 80 ins D0 w 'write' cla A0 ins 82 w 'set size'	cla 00 ins A4 w 'SELECT' cla 80 ins 50 w 'INITIALIZE UPDATE' cla 84 ins 82 w 'EXTERNAL AUTHENTICATE ' cla 80 ins F2 w 'GET STATUS' cla 80 ins E4 w 'DELETE' cla 80 ins E6 w 'INSTALL' cla 80 ins E8 w 'LOAD, cla 84 ins E4 w 'DELETE' cla 84 ins E6 w 'INSTALL' cla 84 ins E8 w 'LOAD'

Figure 42 : Le pare-feu des APDUs du serveur SIM

Le cycle de vie des applications embarquées dans ces cartes à puce est géré selon la norme GP, pour gérer les opérations de téléchargement, d'activation et/ou de suppression. Il existe six commandes élémentaires :

- ✓ **SELECT** : cette commande permet par exemple de sélectionner l'AID de l'application Java Card (i.e. EAP-TLS).
- ✓ **INITIALIZE or UPDATE** : cette commande permet d'initialiser ou de mettre à jour une procédure d'authentification mutuelle.
- ✓ **EXTERNAL-AUTHENTICATE** : cette commande met fin à une procédure d'authentification mutuelle et confirme la création d'un canal sécurisé.
- ✓ **DELETE** : cette commande supprime une application ou un paquet.
- ✓ **INSTALL** : cette commande permet d'activer une application.
- ✓ **LOAD** : cette commande charge l'image binaire d'une application.

L'acheminement des APDUs s'appuie sur la norme PC/SC (Personal Computer/Smart Card) [47]. Cette norme spécifie comment intégrer des lecteurs de cartes à puce et les cartes à puce dans un environnement informatique, ainsi que la façon d'offrir à plusieurs applications la possibilité de partager des périphériques de cartes à puce. PC/SC est une bibliothèque spécifiée par des grands fabricants de carte à puce ou d'ordinateurs (PC/SC Workgroup). Le but de ce middleware est de normaliser des commandes pour assurer une meilleure interopérabilité entre PC, cartes à puce et lecteurs de ces cartes.

Le projet OpenSC (Open Source) [48] supporte cette norme PC/SC qui fournit un ensemble de bibliothèques et de services conçus pour fonctionner et dialoguer avec les cartes à puce (Figure 43). C'est un middleware qui se focalise sur des cartes à puce qui supportent des opérations cryptographiques, pour faciliter leur intégration dans des applications de sécurité comme par exemple : l'authentification, le chiffrement des messages, la signature numérique, etc.

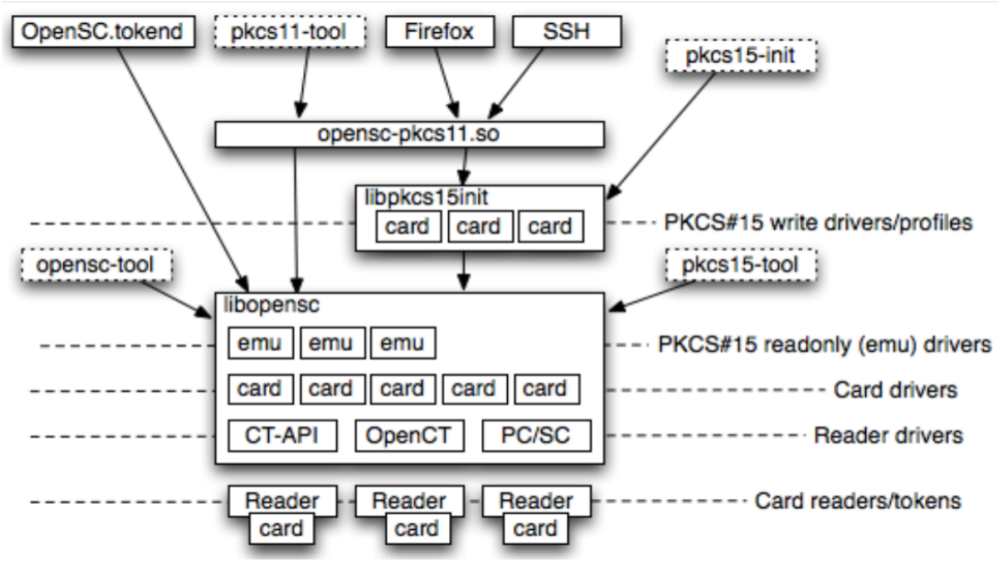

Figure 43 : Architecture globale d'OpenSC et ses interfaces externes

OpenSC présente plusieurs avantages : multiplateforme (Linux, Windows et Mac OS), supporte plusieurs cartes à puce les plus courantes ainsi que des lecteurs de cartes. OpenSC présente un autre avantage important, car il implémente la norme PKCS#15 pour être compatible avec toutes les cartes et l'API PKCS#11, qui est supportée par les applications (Mozilla Firefox et Thunderbird).

Pour dialoguer avec le lecteur de carte, l'API PC/SC s'appuie sur des pilotes (drivers) au format "ifdHANDLER". La communication est assurée par un protocole en mode liaison série (port DB9 ou USB). La norme PC/SC est supportée par plusieurs systèmes d'exploitation (Linux, xBSD, Mac OS, Windows, etc). Sous Windows, PC/SC est le standard de facto pour l'accès à la carte à puce, il est implémenté nativement dans le système d'exploitation avec le support de plusieurs lecteurs de cartes à puce. Linux et Mac OS X utilisent la suite "*pcsc-lite*", car la plupart des lecteurs de cartes à puce modernes (USB : Universal Serial Bus) prennent en charge les spécifications CCID (Chip/Smart Card Interface Devices) [100]. Sous Linux, OpenSC utilise plusieurs options (CT-API, OpenCT, PC/SC) (Figure 43), pour piloter les lecteurs de cartes. En procédant à la configuration de certains fichiers spécifiques, ces options peuvent être activées et/ou désactivées pour conserver celles dont nous avons besoin.

6.3) Les briques logicielles et architecturales

Comme mentionné dans le Chapitre IV, l'infrastructure SecFuNet est constituée de domaines administratifs et de plusieurs couches de visibilité principales : infrastructure, plateforme et application. Chaque domaine est associé à un administrateur système ou réseau. Les ressources (poste client, VM, serveur XEN, etc.) sont associées à des éléments de sécurité qui incarnent leur identité physique dans un Cloud, tout en restant aussi proches que possible des principes du fonctionnement du protocole d'authentification TLS.

Pour s'interfacer avec ces éléments de sécurité, plusieurs commandes (ISO7816) sous forme de procédures primitives qui s'appuient sur la norme PC/SC ont été programmées pour d'une part différencier les différents types d'APDUs spécifiques, devant être envoyés à ces cartes embarquant la machine à états EAP-TLS et d'autre part pour pouvoir gérer pleinement les opérations de (lecture/écriture) de la mémoire E²PROM.

Ces primitives ont été organisées sous forme de briques architecturales pour pouvoir établir des liaisons logiques entre les cartes à puce et les ressources informatiques. Ces briques, que nous appellerons "Proxys", doivent s'exécuter en tâche de fond dans une ressource physique ou virtuelle pour dérouler le protocole TLS embarqué dans la carte à puce, au lieu d'exécuter celui implémenté dans la ressource.

Les Proxys jouent donc, un rôle architectural crucial pour établir cette liaison logique et respecter le protocole TLS pour établir une session sécurisée avec n'importe quel type de serveur TLS. Les Proxys sont de trois types (Figure 44) : client, VM, serveur.

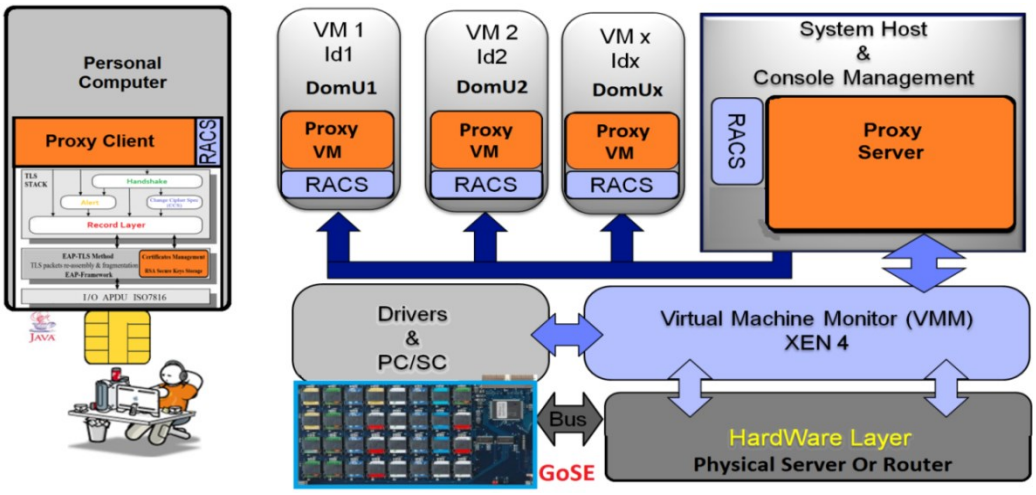

Figure 44 : Architecture logicielle globale de SecFuNet

6.3.1) Implémentation du Proxy Client

Le Proxy Client est une API qui fait office de médiateur entre le poste client et le serveur d'authentification ou le Proxy Server installé dans un hyperviseur XEN. C'est un middleware subdivisé aussi en plusieurs couches logicielles intégrées dans un poste client. Ce middleware est considéré comme un "Couteau Suisse", car il se comporte de différentes manières selon le contexte de l'architecture (Chapitre V), mais le principe de fonctionnement reste le même. Il est constitué de plusieurs composants (Figure 45).

A) **Le pilote (driver)** : le pilote natif est compatible avec la norme PC/SC pour gérer le lecteur de carte à puce. Il s'appuie sur des outils (OpenSC et OpenCT) qui permettent la programmation de nouvelles fonctionnalités pour s'interfacer avec les cartes à puce et leur lecteur. En effet, dans le cadre du projet SecFuNet, certains modules d'OpenSC ont été réutilisés pour assurer l'acheminement du processus d'authentification, ainsi que le cryptage des données et la signature numérique. Malheureusement, aucun pilote intégré dans OpenSC sous Linux, ne supportait nos lecteurs de cartes à puce du démonstrateur, bien qu'ils soient compatibles avec la norme PC/SC.

Pour résoudre ce problème de pilote, nous avons procédé dans un premier temps à des ajustements et des modifications de certains fichiers, pour enfin personnaliser et pouvoir

compiler un pilote pour notre plateforme basée sur le système d'exploitation Fedora version 20 et un autre middleware : OpenCT. C'est un projet Open Source implémentant une grande variété de pilotes sous forme d'une suite de logiciels libres. Il permet de réécrire, sous Linux, des pilotes pour des lecteurs spécifiques de carte à puce, au format ifdHANDLER requis pour la suite pcsc-lite. OpenCT offre ainsi une API directement utilisable par OpenSC et par notre lecteur de cartes compatible avec la norme PC/SC. Notre Proxy Client peut donc s'appuyer directement sur ce pilote pour s'interfacer avec la carte à puce.

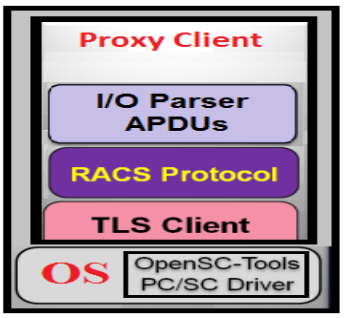

Figure 45 : Les composants du Proxy Client

B) **Le Client TLS** : ce client est utilisé pour sécuriser les connexions et les transactions entre le poste client et une ressource virtuelle, ou pour sécuriser les connexions et les transactions en utilisant le protocole RACS (§6.6). Cette couche intervient généralement, pendant l'initiation d'une session TLS, pour gérer les opérations TCP/IP et l'acheminement des messages d'authentification entre deux entités, en utilisant un ensemble de ressources fournies par les composants socket : « connect (), send (), recv () et shutdown ()». À la fin de la Phase I, le Client TLS offre deux procédures de haut niveau pour assurer le chiffrement et le déchiffrement des données dans la Phase II.

C) **L'I/O APDU Parser** : cette couche logicielle réalise les fonctions principales suivantes : envoyer ou recevoir des commandes ISO7816 de la carte à puce via le lecteur de cartes à puce, relié à un poste client, formater les messages provenant d'une carte à puce et les rediriger vers un serveur d'authentification via le Client TLS.

Elle se comporte comme un lecteur logiciel de carte à puce, utilisée pour formater les messages d'authentification. Elle permet d'une part d'interpréter toutes les requêtes provenant d'un serveur TLS et d'autre part de les formater pour ensuite les transférer à la carte à puce du poste client sous forme d'APDUs. Une fois ces requêtes traitées par la machine à états embarquée dans la carte de l'utilisateur, cette couche logicielle récupère les réponses de celle-

ci, de nouveau elle les formate et les transfère au serveur TLS, sous forme de message purement TLS en utilisant le Client TLS.

Pour assurer ce lien logique entre la carte à puce et le serveur d'authentification nous avons développé un ensemble de fonctionnalités qui a permis de réaliser une gestion efficace de la machine à états EAP-TLS ; parmi celles-ci, citons notamment :

- *SCard_init_remote_reader* : initialise le lecteur de carte à puce. Cette fonction retourne le nombre de lecteurs de cartes détectés, sélectionne l'AID de l'application Java Card (i.e. EAP-TLS) (§4.2.2 C) installée dans la carte à puce, définit l'identité du client ou du serveur et vérifie le code PIN, démarre une session avec la carte à puce identifiée par son index (premier indice fixé à zéro), et initialise le T-protocol utilisé par la carte à puce.

- *SCard_start_remote_eap :* initialise la machine à état courante EAP-TLS et démarre un processus EAP.

- *SCard_connect_remote_reader* : ouvre une connexion avec un lecteur de carte.

- *SCard_disconnect_remote_reader* : déconnecte une session établie avec une carte à puce, identifiée par son index.

- *SCard_list_remote_reader* : liste l'ensemble des lecteurs de cartes locaux ou distants.

- *SCard_send_remote_apdu_cmd* : envoie des commandes ISO7816 à une carte à puce.

- *SCard_get_remote_cipher_suite* : collecte les paramètres (CipherSuite) négociés avec une carte à puce, et associés à une session TLS ouverte.

- *SCard_get_remote_key_block* : collecte la clé éphémère (KEY_BLOCK) à partir de la carte à puce, associée à une session TLS. Cette fonction peut être considérée comme dangereuse du point de vue sécurité, mais l'accès à cette clé est protégé par un code PIN.

- *SCard_parse_rw_message* : permet de lire les réponses à partir de la carte à puce, en les analysant et en les envoyant vers la ressource en question.

- *SCard_process_eap_tls_packet* : reçoit des instructions d'une ressource et construit un paquet EAP-TLS pour l'envoyer à la carte à puce.

6.3.2) Implémentation du Proxy VM

Le Proxy VM est une API qui fait office de pont entre le poste client et le serveur d'authentification distant ou le Proxy Server installé dans l'hyperviseur XEN. C'est un middleware, subdivisé aussi en plusieurs couches logicielles, intégrées dans une machine virtuelle (Figure 46).

A) **Le Client TLS** permet d'établir un canal sécurisé entre la machine virtuelle et le

serveur TLS de la GoSE hébergeant la carte à puce associée à cette VM. Ce canal est utilisé pour protéger les messages d'authentification entre le poste client qui souhaite s'authentifier au niveau d'un service offert par une VM et la carte à puce associée à cette VM ou ce service. Il est aussi utilisé pour sécuriser le protocole RACS (§6.6).

B) **Le Serveur TLS** intervient généralement, pendant l'initiation d'une session TLS, pour gérer les opérations TCP/IP et l'acheminement des messages d'authentification entre le poste client et la VM, en utilisant aussi un ensemble de ressources fournies par les composants socket : « *connect (), send (), recv () et shutdown ()*». À la fin de la Phase I (authentification) le serveur TLS offre deux procédures de haut niveau pour assurer le chiffrement et le déchiffrement des données (Phase II).

C) **Le Forwarding Agent** ne fait que rediriger les requêtes et/ou les réponses, pendant la Phase I, entre un poste client et la carte à puce de la VM hébergée dans la grille distante. Ce composant n'interprète ni les requêtes ni les réponses, il ne fait qu'extraire les messages d'authentification et les rediriger soit vers le poste client ou vers la carte à puce associée à la VM via le canal sécurisé par la couche Client TLS. Une fois la Phase I est terminée, l'agent collecte les paramètres (CipherSuite) négociés avec la carte à puce associée à la VM et la clé (KEY_BLOCK) associée à une session TLS. Il commence ensuite la Phase II avec le client.

6.3.3) Implémentation du Proxy Server

Le Proxy Server, est subdivisé en plusieurs couches logicielles (Figure 46), intégrées dans l'hyperviseur XEN ou dans un serveur d'authentification relié à une GoSE.

Ce middleware permet d'une part d'interpréter toutes les requêtes provenant d'une VM ou d'un poste client et d'autre part de les formater pour les transférer sous forme d'APDUs à une carte à puce dans une grille distante. Celle-ci est équipée d'un ensemble de cartes à puce personnalisables qui embarquent la machine à états (i.e. EAP-TLS Server). Ce Proxy est constitué :

- d'un pilote natif (driver) pour gérer la grille de cartes à puce, lequel a été adapté à nos lecteurs spécifiques de cartes pour les besoins de notre plateforme de démonstration.
- d'un serveur TLS pour sécuriser les connexions et les transactions entre la grille et les machines virtuelles ou un poste client.
- d'un protocole RACS (Remote APDU Call Secure Protocol) qui a fait l'objet d'un draft à l'IETF en Juillet 2014, permettant de gérer des modules matériels de sécurité (§6.6).

Ce protocole fournit toutes les fonctionnalités nécessaires pour gérer à distance des éléments sécurisés.

- d'un composant logiciel très important nommé (API EAP-TLS), qui permet de s'interfacer avec les cartes à puce, pour jouer le rôle de médiateur entre les machines virtuelles et leur carte associée. Le but de cette interface logicielle est de définir un sous-ensemble de base de plusieurs fonctionnalités pour la grille de cartes à puce, afin d'être en conformité avec la norme PC/SC. Elle fait abstraction des commandes ISO7816 élémentaires (codées en format hexadécimal) pour obtenir des opérations de gestion de la grille de haut niveau.

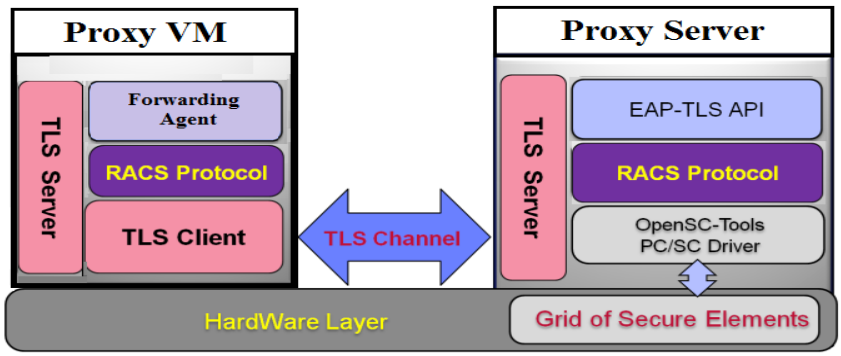

Figure 46 : Le Proxy VM et le Proxy Serveur

Parmi ces fonctionnalités, citons :

- **SELECT** : la carte peut stocker plusieurs applications identifiées par un ensemble de caractères ASCII (i.e. AID : Application Identifier). Cette commande sélectionne l'AID de l'application EAP-TLS qui va être exécutée.
- **GET-CURRENT-IDENTITY** : cette commande demande à la carte de fournir l'identité courante.
- **GET-NEXT-IDENTITY** : cette commande retourne l'identité qui a été chargée précédemment, dans le cas où plusieurs identités ont été chargées dans la carte.
- **SET-IDENTITY** : cette commande sélectionne l'identité qui sera utilisée dans le processus EAP-TLS.
- **VERIFY-PIN** : cette commande permet de vérifier le code PIN d'un utilisateur qui souhaite accéder à des informations dans une carte à puce.
- **CONTENT :** cette commande permet de retourner le nombre de cartes à puce disponibles dans une grille distante.

- **RESET** : cette commande permet pendant une session EAP-TLS de forcer la carte à se mettre dans l'état "AUTHENTIFICATION" (i.e. initialise la machine à état EAP-TLS sans changer l'identité courante).
- **PROCESS-EAP** : cette commande forge des paquets EAP pour les transmettre à la carte qui doit répondre avec d'autres paquets du même type (i.e. en-tête EAP) ou un message de réponse approprié sous forme de mots 'SW1SW2'.
- **GET-RESPONSE** (**MORE** or **FETCH**) : cette commande doit être exécutée à chaque fois pour récupérer des réponses provenant de la carte suite à une requête précédente, tant que les mots d'état suivants : 'SW1 = 61 ou SW1 = 9F et SW2 = XX', et où XX est le nombre d'octets (en format hexadécimal) qui devra être demandé par la commande **FETCH**.
- **SET-SESSION-ID** : cette commande permet de fournir la variable *Session-ID* de l'identité actuelle, par le protocole EAP-TLS. Elle est principalement utilisée pour forcer une session TLS à se dérouler en FULL Mode.
- **GET-CIPHER-SUITE** : cette commande permet de récupérer la CipherSuite déjà négociée au cours du Handshake. La valeur retournée correspond aux valeurs CipherSuite définies par le RFC 5246.
- **GET-KEYS-BLOCK** : cette commande permet de récupérer la clé de session (KEY-BLOCK), qui sera dérivée et ensuite utilisée pour protéger la session TLS.
- **INITIALIZEGRID** : cette commande permet d'initialiser une grille distante identifiée par le nom de son serveur, et retourne le nombre de cartes à puce hébergées par la grille (retourne -1 en cas d'erreur). Elle s'appuie sur deux commandes : RESET et CONTENT.
- **CONNECTGRIDSC** : cette commande permet d'activer une session avec une carte à puce qui se trouve dans la grille ; le microcontrôleur sécurisé est identifié par un indice. En cas de réussite cet indice est retourné, ainsi qu'une valeur qui pointe vers une référence socket TCP/IP.
- **DECONNECTGRIDSC :** cette commande permet de clôturer une session avec une carte à puce distante identifiée par son indice et une référence de socket TCP/IP, précédemment initiée par la fonction *CONNECTGRIDSC.*
- **SENDGRIDSC** : cette commande permet d'envoyer des APDUs à une carte à puce, identifiée par son indice, associée à un socket TCP/IP.

6.3.4) Exemples de quelques fonctions implémentées

Pour s'interfacer avec les cartes à puce nous avons implémenté des fonctionnalités élémentaires en langage ANSI C, de haut niveau, pour pouvoir les réutiliser dans d'autres programmes. Celles-ci permettent d'envoyer des instructions plus évoluées aux cartes et à la grille, au lieu d'envoyer des commandes ISO7816 élémentaires (Figure 47).

```
/** SCard EAP TLS Commands / Some ISO7816 commands used in this project **/

#define SCARD_FILE_EAPTLSCMD   0x00, 0xA4, 0x04, 0x00, 0x10
#define SCARD_VERIFY_PINCMD    0xA0, 0x20, 0x00, 0x00, 0x04
#define SCARD_SET_ID_CMD       0xA0, 0x16, 0x00, 0x80, 0x06
#define SCARD_RESET_EAPTLS     0xA0, 0x19, 0x10, 0x00, 0x00
#define SCARD_PROCESS_EAPTLS   0xA0, 0x80, 0x00, 0x00, 0x0A

/** Get all cipher suite and key block **/
#define SCARD_GET_CIPHER_SUITE     0xA0, 0x82, 0xCC, 0x00, 0x00
#define SCARD_GET_KEY_BLOCK        0xA0, 0x82, 0xCA, 0x00, 0x00
#define SCARD_GET_CERT_BLOCK       0xA0, 0x60, 0x00, 0x00, 0x00

typedef enum {
    SCARD_GSM_SIM_ONLY,
    SCARD_USIM_ONLY,
    SCARD_TRY_BOTH,
    SCARD_EAPTLS_ONLY
} scard_sim_type;
typedef enum {
    SCARD_GSM_SIM, SCARD_USIM, SCARD_EAPTLS
} sim_types;

/* SCard EAP TLS File = AID */
unsigned char SCARD_FILE_EAPTLS[] = {0xA0, 0x00, 0x00, 0x00, 0x30, 0x00, 0x02, 0xFF,
                                     0xFF, 0xFF, 0xFF, 0x89, 0x31, 0x32, 0x38, 0x00};
char SCARD_VERIFY_PIN[] = {0x30, 0x30, 0x30, 0x30};
char SCARD_ID_CLIENT[] = {0x63, 0x6C, 0x69, 0x65, 0x6E, 0x74};
unsigned char SCARD_ID_SERVER[] = {0x73, 0x65, 0x72, 0x76, 0x65, 0x72};
unsigned char SCARD_GET_RESP[] = {0xA0, 0xC0, 0x00, 0x00, 0x00};
unsigned char SCARD_START_EAPTLS[] = {0x01,0x14,0x00,0x06,0x0D,0x20,0x00,0x00,0x00,0x00};
```

Figure 47 : Illustration des commandes ISO 7816 en ASCII

L'ensemble de ces fonctionnalités, a permis d'affiner la gestion de la machine à état (sous forme de protocole EAP-TLS), embarquée dans les microcontrôleurs de la grille.

La fonction (*scard_reset_eaptls*) représente le code ASCII de la commande ISO7816 (SCARD_RESET_EAPTLS) pour initialiser la machine à état EAP-TLS sans modifier l'identité en cours d'utilisation.

La fonction (*scard_set_identity_eaptls*) représente le code ASCII de la commande ISO7816 (SCARD_SET_ID_CMD) qui permet de choisir une identité d'un utilisateur dans le conteneur de la carte à puce pour l'utiliser dans une session TLS.

La fonction (*scard_verify_pin_eaptls*) représente le code ASCII de la commande ISO7816 (SCARD_VERIFY_PINCMD) ; elle est utilisée lorsque l'utilisateur souhaite accéder à une application ou à des données protégées dans la carte à puce.

6.4) Les Scénarios d'authentification dans l'architecture globale

Dans cette section, nous présentons deux scénarios qui illustrent deux cas d'usage de déploiement des Proxys dans une architecture impliquant les différents éléments que sont les cartes à puce, le poste client ou la VM et le serveur d'authentification distant. Le premier scénario (§6.4.1) s'attache à décrire l'établissement d'une session sécurisée entre un poste client, en utilisant une carte EAP-TLS (en mode client), et un serveur TLS SecFuNet. Le deuxième scénario (§6.4.2), met en évidence l'établissement d'une connexion (M2M) entre deux cartes à puce (EAP-TLS mode Client et EAP-TLS mode Server), dont le principe de fonctionnement reste similaire au premier scénario.

6.4.1) Scénario I : Authentification d'un client SecFuNet

La Figure 48 représente des échanges entre la carte à puce, le Proxy installé dans le poste client, et le serveur d'authentification distant TLS. Le scénario de fonctionnement est alors le suivant : une fois la carte à puce EAP-TLS introduite dans le lecteur de carte du poste client, le Proxy Client déjà démarré en tâche de fond la détecte grâce à la fonction (***SCard_init_remote_reader)*** et demande à l'utilisateur de saisir son code PIN. Si aucune identité n'est retrouvée, ou si aucune carte n'a été détectée, ou si le code PIN est erroné, toutes les opérations d'initialisation d'une session TLS seront refusées.

Quand l'utilisateur souhaite établir une session sécurisée avec un serveur TLS, le Proxy Client (i.e. *I/O APDU Parser*) débute les opérations de la première phase de TLS (Phase I), fixe l'identité TLS associée à cette carte (***Set-Identity***), et lui envoie ensuite un message (***EAP-TLS-start***). La carte génère et envoie un *ClientHello* encapsulé dans du protocole EAP. Le Proxy Client assure l'interprétation et la traduction des messages qui transitent entre les deux entités. Il récupère le message *ClientHello*, supprime les paquets EAP puis l'envoie au serveur TLS distant, en utilisant *socket.send*. Après traitement de ce message par le serveur TLS distant, le Proxy Client récupère, par l'intermédiaire de *socket.recv* plusieurs messages selon les cas "*Resume Mode*" ou "*Full Mode*" :

1) Si le message « *ServerHello* » suivi par les deux messages "*ChangeCipherSpec*" et "*Finished*", il s'agit dans ce cas d'une reprise de session "*Resume Mode*". La carte génère

alors une réponse qui sera traitée par le Proxy Client et transmise au serveur TLS. Cette réponse met fin aux échanges de la Phase I et passe directement à la Phase II.

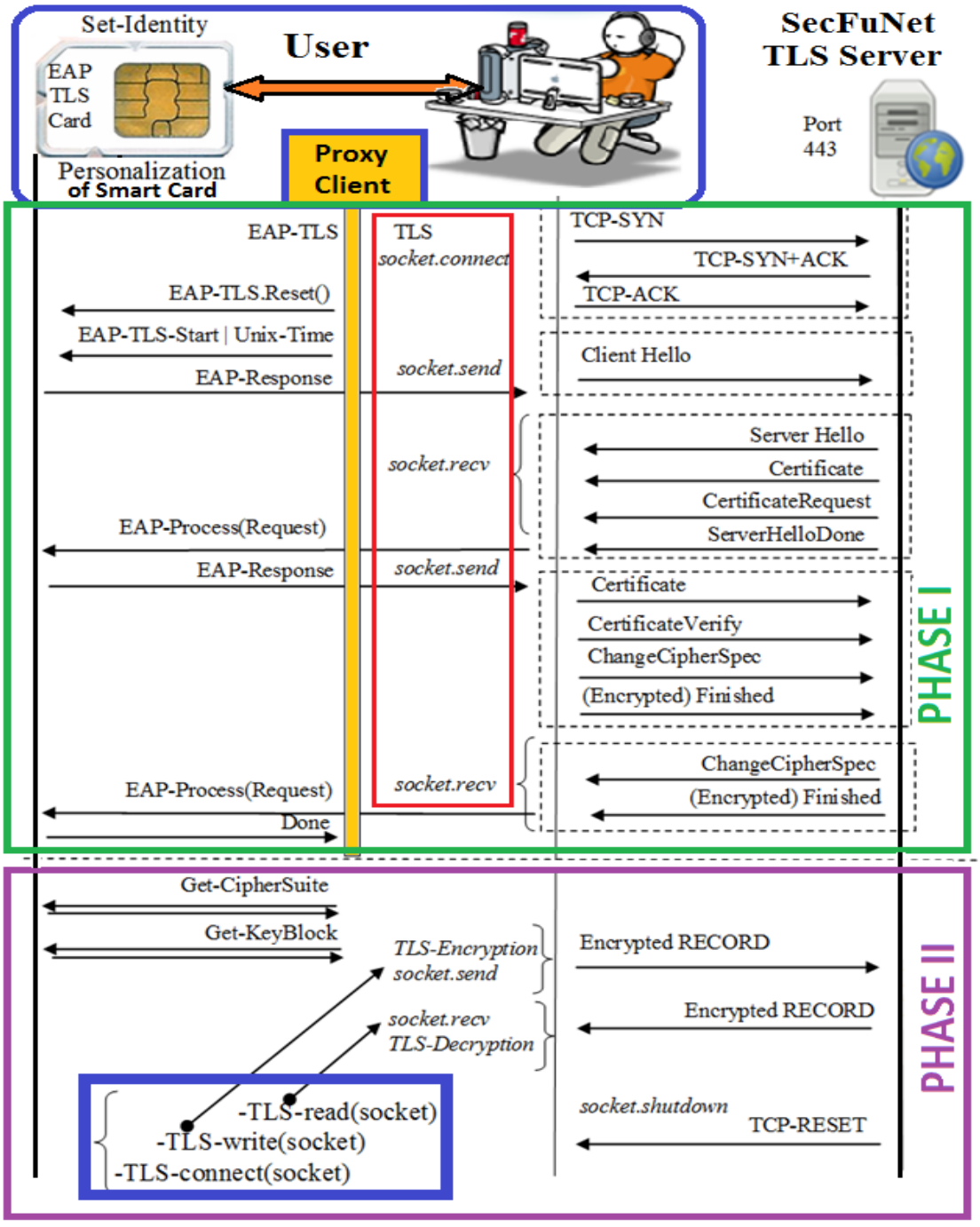

Figure 48 : Illustration d'une connexion entre le Proxy Client et le serveur TLS

2) Si le message « *ServerHello* » est suivi des messages "*Certificate*", "*CertificateRequest*" et "*ServerHelloDone*", il s'agit d'une session complète "*Full Mode*". Dans les deux cas, ces messages seront fragmentés par la fonction (***SCard_process_eap_tls_packet),*** et envoyés vers la machine à états embarquée dans la carte.

Le mécanisme de cette fonction consiste à découper en petits fragments (240 octets), les messages TLS provenant du serveur TLS distant. Ensuite, chaque fragment est encapsulé dans un paquet EAP-TLS, et puis dans une commande APDU au format ISO7816. L'ensemble de ces commandes APDUs est alors envoyé vers la carte à puce. Pour chaque envoi d'une commande APDU, le Proxy Client reçoit une réponse d'acquittement de la carte sous forme de deux octets (SW1SW2 = 9000). L'ensemble des échanges du Handshake se poursuit ainsi, entre le serveur TLS distant et la carte à puce, jusqu'à l'obtention des messages *ChangeCipherSpec et Finished*.

À l'issue de la Phase I, le Proxy Client collecte deux paramètres à partir de la carte à puce : " *CipherSuite* " et "*Key-block*". Cette dernière clé sera dérivée en un trousseau de clés (§2.3.2.3 C) pour chiffrer et/ou déchiffrer les données envoyées et/ou reçues du serveur distant dans la Phase II (Data Applications). Ces opérations sont effectuées via les fonctions TLS-read () et TLS-write (). La carte à puce n'est plus sollicitée, sauf si nous avons besoin de récupérer de nouveau ces deux paramètres.

Ces opérations de chiffrement et de déchiffrement des données peuvent être réalisées de deux façons différentes : la première consiste à chiffrer et/ou déchiffrer les données au sein même de la carte à puce, de sorte à ce que les clés de session restent en sécurité dans celle-ci. Cette solution est très séduisante, mais dégrade largement les performances pendant le traitement de données applicatives importantes. La seconde, que nous avons adoptée, consiste à exporter les clés de session dans le poste client, les opérations de chiffrement et de déchiffrement des données applicatives sont alors déléguées à ce dernier, ce qui permet de tirer avantage de sa puissance de calcul.

Cette dernière solution peut avoir des impacts du point de vue de la sécurité, car elle présente un léger risque potentiel qui consiste à subtiliser la clé "key-block" par un attaquant avant qu'elle soit détruite. C'est une clé éphémère, contrairement à la clé "Master KEY" qui est stockée en permanence dans la carte à puce pendant une session TLS.

En effet, la récupération de cette clé éphémère par un attaquant lui permettrait de déchiffrer une petite partie des données pendant la session en cours, mais ne l'autoriserait pas à récupérer les autres clés ("Master KEY" ou la clé privée qui sont stockées dans la carte). Par contre, la compromission de ces dernières permettrait à un intrus d'ouvrir de nouvelles sessions TLS ou de dériver de nouvelles clés temporaires pour déchiffrer la totalité des données d'une session.

6.4.2) Scénario II : Connexion à un service web dans SecFuNet

Dans le scénario précédent, nous avons décrit les échanges de messages d'authentification entre un Proxy Client (i.e. carte à puce EAP-TLS) et un serveur TLS distant, en utilisant la machine à états en mode client (elle supporte deux modes d'authentification : EAP-TLS mode Client et EAP-TLS mode Server).

Dans ce scénario, nous allons détailler les échanges entre deux cartes à puce (M2M) pour s'authentifier mutuellement, en passant par une VM et son Proxy, exécutée dans le domaine U. Cette VM est associée à une carte à puce qui se trouve dans une grille distante.

Pour accéder à sa carte à puce distante, un canal sécurisé est établi entre la VM et le serveur TLS de l'hyperviseur XEN, en utilisant le Client TLS du Proxy VM.

Un utilisateur SecFuNet tente de consommer un service Web HTTPS, offert par cette VM (Figure 49). Les échanges entre le Proxy Client, le Proxy VM, le Proxy Serveur et la carte à puce associée à la VM, se déroulent de la manière suivante :

1) L'utilisateur souhaite établir une session sécurisée avec le serveur Apache SecFuNet installé dans la VM. Le Proxy Client débute les opérations de la première Phase (Handshake) en envoyant le message *ClientHello* généré par la carte du poste client.

2) Ce message *ClientHello* est récupéré par le Proxy VM, lequel le redirige vers le Proxy Server.

3) Ce dernier fragmente ce message, l'encapsule dans un paquet EAP-TLS, ensuite il l'encapsule dans une commande APDU au format ISO7816. Celle-ci est alors envoyée vers la carte à puce associée à la VM, laquelle traite le message et retourne au Proxy Server les messages *ServerHello, Certificate, CertificateRequest et ServerHelloDone*.

4) Ces messages sont traités par le Proxy Server sous forme de messages purement TLS, puis transmis au Proxy VM, qui les redirige vers le Proxy Client.

5) Ce dernier récupère les messages *ServerHello, Certificate, CertificateRequest et ServerHelloDone,* les fragmente, les encapsule dans des paquets EAP-TLS, puis les encapsule dans des commandes APDUs au format ISO7816. L'ensemble de ces APDUs est alors envoyé à la carte à puce du poste client. Ainsi, les échanges du Handshake se poursuivent entre les deux cartes à puce, jusqu'à l'obtention des messages *ChangeCipherSpec et Finished*. À la fin de la Phase I du Handshake, commence la Phase II de chiffrement et de déchiffrement des

données applicatives, en utilisant les clés temporaires *KEY BLOCK* et la *CipherSuite* associées à cette session et partagées par le poste client et le serveur Apache de la VM.

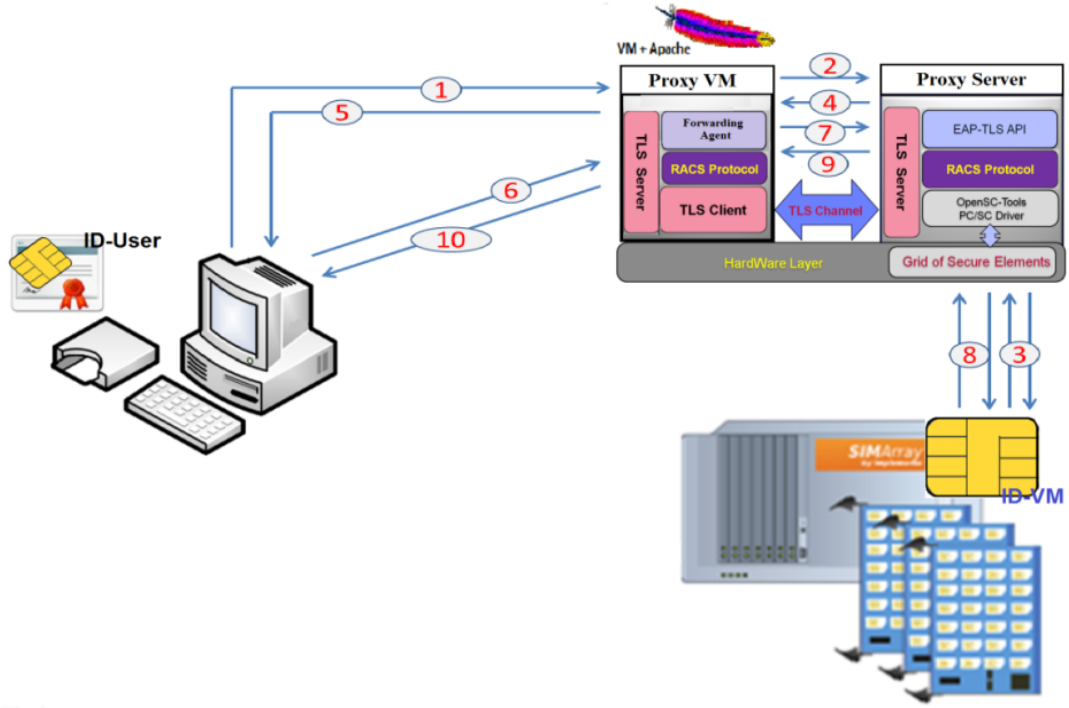

Figure 49 : Illustration des interactions entre le Client et la carte à puce de la VM

Pendant ces échanges, nous pouvons noter que le Proxy VM n'a pas modifié les paquets, mais qu'il avait simplement pour rôle de les rediriger vers les deux cartes à puce. Le déploiement de ces briques architecturales permet de donner une meilleure vision globale de la compatibilité avec un modèle de gestion des identités classique.

Ces briques permettent d'établir des liens logiques pour acheminer les messages d'authentification en toute sécurité entre les ressources. Ainsi, la VM délègue la fonction d'authentification à un élément sécurisé déporté sur une grille distante ; l'idée est d'établir des sessions TLS entre deux cartes (M2M).

L'architecture présentée dans le Chapitre V pourrait facilement être adaptée pour se calquer sur le modèle architectural que nous venons de décrire. Les problèmes d'authentification mutuelle trouvent leurs solutions pour le côté client et le côté VM dans l'environnement de confiance que représente l'élément sécurisé.

De ce point de vue, un tel modèle architectural, que ce soit dans le contexte du chapitre précédent ou dans d'autres contextes, s'inscrit pleinement dans le paradigme du Cloud Computing en proposant un nouveau service (TaaS : Trust as a Service).

6.4.3) Analyse des performances de la carte EAP-TLS

Pour chaque opération, la carte à puce sélectionnée pour les besoins du projet SecFuNet consomme un temps T_x exprimé en en millisecondes. Ces caractéristiques temporelles sont illustrées sur le Tableau 3.

De manière générale, les performances cryptographiques restent acceptables et varient entre 0,50 ms et 2,60 ms, à l'exception du temps consommé par la carte pour traiter des opérations avec les clés RSA publique ou privée qui sont respectivement de l'ordre 25 ms et 550 ms.

Tableau 3: Les caractéristiques temporelles cryptographiques du microcontrôleur

	T_{RSAPub}	$T_{RSAPriv}$	T_{MD5}	T_{SHA1}	T_{3xDES}	T_{AES}	T_{RC4}
Size of Key or Bloc	RSA-Pub 1024	RSA-Priv 1024	MD5 / Bloc 64B	SHA1 / Bloc 64B	3xDES / Bloc 8B	AES / Bloc 16B	RC4 / byte
Timings in ms	25	550	0,50	0,90	2,10	2,60	0,50

En outre, on note que :

- T_{RW} : c'est le temps mesuré par l'hôte pour écrire ou lire des données stockées dans la mémoire non-volatile, il est de l'ordre de 0,15 ms/octets.
- T_{MD5} ou T_{SHA1} : c'est le temps nécessaire pour traiter des fonctions de hachage (e.g. MD5 et SHA1) pour calculer par exemple le haché d'un bloc de 64 octets la carte consommera : 0,50 ms en utilisant MD5 et 0,90 ms en utilisant SHA1.
- T_{3xDES}, T_{AES}, T_{RC4} : c'est le temps nécessaire pour traiter des opérations de chiffrement ou déchiffrement symétrique : l'algorithme 3xDES utilise des blocs de 8 octets, le traitement de ces blocs par la carte consommera 2,10 ms, tandis que l'algorithme AES utilise des blocs de 16 octets, le traitement de ces blocs par la carte consommera 2,10 ms. Il convient de noter que RC4 est un algorithme de chiffrement par flot qui consomme 0,50 ms par octet.
- T_{RSAPub}, $T_{RSAPriv}$: c'est le temps nécessaire pour que la carte à puce chiffre ou déchiffre avec des clés RSA (publiques ou privées) dont la taille est de 1024 bits. Le traitement par la carte coûtera 25 ms en utilisant une clé publique RSA et 550 ms en utilisant une clé privée RSA.

A) Phase I : Full et Resume Mode

L'ouverture d'une session complète basée sur une cryptographie RSA de 1024 bits, nécessite 2000 ms et échange 2500 octets dont le transfert coûte 400 ms. Elle effectue environ 230 opérations MD5 et SHA1, portant sur des blocs de 64 octets ; ces opérations consomment 320 ms.

Une opération RSA avec la clé privée et deux procédures RSA avec des clés publiques consomment 600 ms. La somme des opérations précédemment citées est 1320 ms, le temps restant (680 ms = 2000 ms — 1320 ms) est consommé par l'exécution de code Java Card. L'ouverture d'une session de reprise « Resume Mode » coûte 500 ms. Elle nécessite l'échange de 250 octets, soit 40 ms, et le calcul de 75 (MD5 et SHA1) ; ceux-ci consomment environ 110 ms. Le temps restant (350 ms) est le temps consommé pour exécuter le code Java Card.

B) Phase II : Chiffrement/Déchiffrement des données

Une fois le canal TLS établi, les messages échangés par le client et le serveur sont chiffrés avec l'algorithme choisi et l'intégrité est renforcée par l'application d'une procédure HMAC. La dérivation des clés temporaires de la Phase II peut être calculée soit par la carte à puce EAP-TLS, soit par le terminal d'accueil.

Dans le cas où la phase II de chiffrement de données est traitée par le dispositif EAP-TLS, on en déduit (Tableau 3) que le temps de calcul est généralement consommé par les opérations de chiffrement et de déchiffrement.

Les calculs en utilisant des fonctions de hachage sont secondaires. Le coût de traitement par exemple d'un bloc de 1000 octets, en utilisant une CipherSuite classique (RC4 et HMAC-MD5) est de l'ordre de 500 ms pour effectuer des opérations de chiffrement avec une clé symétrique, puis l'utilisation de la fonction MD5 pour calculer le haché consommera 16 ms et enfin la transmission de ces données coûtera 170 ms. Par conséquent, il est obligatoire de transférer la session TLS du microcontrôleur sécurisé vers le terminal auquel il est connecté, afin d'augmenter le débit opérationnel de données.

6.4.4) Analyse des performances de la GoSE

Il est possible de démarrer simultanément un ensemble de tests de connexion, avec une ou plusieurs cartes à puce. Les performances mesurées sont calculées entre les deux

procédures de socket (*socket.send* et *socket.recv*) (§6.4.1) utilisées pour acheminer une requête ISO7816 à la grille et pour ensuite recueillir la réponse associée à une carte à puce.

Nous constatons que chaque machine à états EAP-TLS, consomme environ T_{Server} = 2000 ms par session, reçoit environ 800 octets et transmet environ 1700 octets par session. Sur une grille, le lien série avec chaque carte à puce a une vitesse de transmission de 9600 bauds. Par conséquent, le délai de transfert sur le lien série ($T_{Transfer}$) est d'environ 2500 ms. Ce lien série de réception est commun à toutes les cartes à puce. Par contre chaque carte possède un lien de transmission série.

Le Proxy Client transmet environ 32 requêtes ISO7816. Chaque requête nécessite deux paquets TCP (une pour envoyer et l'autre pour recevoir), qui coûtent environ un RTT (Round Time Trip) de 100ms. Le temps nécessaire pour établir une seule session TLS devrait coûter environ : $T_{Session} = T_{Server} + T_{Transfer} + 32 \times RTT = 7700$ ms, ce qui est compatible avec les valeurs observées sur le marché des HSM. Théoriquement, pour n connexions simultanées en mode complet (Full Mode), nous pouvons en déduire la formule suivante :

$$TnxSE_{Session} = B + n*A, \text{ où } B = T_{Session}, A = 800ms.$$

Ce délai (A) correspond principalement au fait que le lien série de réception n'est pas parallélisable. Donc, lorsque le nombre de connexions est supérieur ou égal à un nombre de cartes à puce donné, certaines sessions devraient être interrompues pour des raisons de délai d'attente de la grille. Les Figures (50 et 51) illustrent les résultats de tests simultanés de 8 à 16 connexions, avec des cartes EAP-TLS identifiées par un SEID-X (Secure Element Identifier) (§4.2.1).

T en ms	SEID-1	SEID-2	SEID-3	SEID-4	SEID-5	SEID-6	SEID-7	SEID-8	SEID-9
Test avec 1 SE	8543								
Test avec 2 SE	9032	9043							
Test avec 3 SE	11821	11654	11637						
Test avec 8 SE	13035	14035	13991	14167	KO	13496	KO	14165	
Test avec 9 SE	17235	17355	KO	17189	19396	KO	19253	KO	17191

Figure 50 : Test de performances des connexions simultanées avec 9 cartes à puce

Voici l'ensemble des mesures, exprimées en millisecondes, pour 16 sessions TLS simultanées (Figure 51) :

Test-1: (21668, 21133, 21664,21464,21447,21453, KO, KO, KO, 20153, 19920, KO, 20125, 20371, 19828, KO),

Test-2: (22464, 21447, 24453, KO, KO, KO, 24056, KO, KO, 24276, KO, KO, KO, 20253, 19990, 21364),

Test-3: (23947, 24151, 24368, 24193, KO, 24303, KO, 24369, KO, 24272, 24056, KO, 22321, 22558, KO, 22213),

Figure 51 : Test de performances de connexions simultanées avec 16 cartes à puce

6.5) Implémentation du protocole RACS

Le protocole RACS (Figures 52) fonctionne au-dessus de la couche de transport TCP et sécurisé par le protocole TLS. Un client TLS doit être authentifié par un certificat pour envoyer des commandes ISO7816 encapsulées dans le protocole RACS.

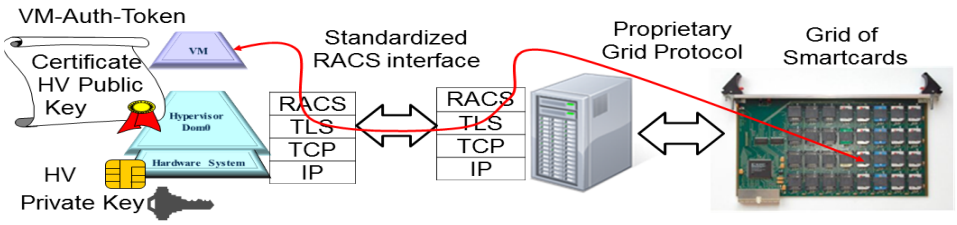

Figure 52 : Intégration du protocole RACS dans SecFuNet

Une requête RACS comprend typiquement un préfixe composé de multiples requêtes ISO7816 et un suffixe qui porte le résultat d'une procédure cryptographique. L'utilisation de TLS avec une authentification mutuelle basée sur un certificat fournit un moyen simple et élégant pour déterminer les pouvoirs d'un client RACS sans passer par la GoSE. Il permet également une séparation facile entre les privilèges des utilisateurs et des administrateurs.

Certains fournisseurs de Cloud Computing (par exemple Amazon Web Services) supportent déjà des serveurs HSM dans leurs infrastructures. Les sessions avec HSM sont protégées par le protocole TLS avec authentification mutuelle basée sur des certificats. Les clients utilisent les services HSM grâce à des bibliothèques "PKCS#11" ou "Cryptoki".

Le protocole RACS transporte des commandes ISO7816, dont les règles d'encodage binaire sont normalisées. Il offre des services cryptographiques basés sur le Framework Java Card et améliorés par des processeurs cryptographiques.

Une ressource distante (i.e. poste client, VM, etc.) peut répertorier et/ou échanger des informations avec les éléments sécurisés grâce au protocole RACS intégré dans le Proxy. Une VM peut donc envoyer des instructions à la carte à puce pour effectuer des signatures

numériques, des opérations cryptographiques ou accéder à la clé privée associée. Cet accès n'est accordé que pour l'utilisateur ou la VM authentifiée.

Le RACS Client et le RACS Server sont deux composants logiciels intégrés dans les Proxys, qui s'interfacent pour jouer le rôle de médiateur entre le poste client ou les machines virtuelles et leur carte associée. Ces deux composants logiciels se basent sur les fonctionnalités que nous avons implémentées précédemment (§6.3), afin d'obtenir des opérations de gestion de haut niveau de la grille.

6.6.1) Structure du protocole RACS

A) **La structure d'une requête RACS** :

C'est un ensemble de commandes en ligne, codé selon le format ASCII. Chaque ligne se termine par les caractères : Cr (retour chariot) et Lf (saut de ligne). Le protocole RACS est sensible à la casse (case sensitive). Chaque commande est un ensemble de mots séparée par des espaces.

Le premier mot de chaque ligne est la commande à exécuter. Une ligne de commande peut comprendre d'autres mots, appelés les arguments de la commande. Une requête RACS doit commencer par une commande **BEGIN** et doit se terminer par une commande **END**.

Chaque ligne de commande est associée à un numéro de ligne implicite. La ligne **BEGIN** est associée au numéro de ligne zéro. Le traitement d'une requête RACS s'arrête après la première erreur. Dans ce cas, la réponse renvoyée contient l'état d'erreur (**STATUS**) induit par la dernière commande exécutée.

B) **La structure d'une réponse RACS** :

La Figure 53 illustre la structure des commandes RACS. C'est un ensemble de lignes codé selon le format ASCII. Chaque ligne se termine par un Cr (retour chariot) et un saut de ligne (Lf). C'est un ensemble de mots séparés par des espaces (0x20). La réponse RACS comprend une ou plusieurs lignes sous forme d'états, elle commence par un en-tête **BEGIN** et finit par **END.** Le premier mot de chaque ligne est donc un en-tête. Chaque ligne commence soit par le caractère "+ " quand il s'agit d'une réponse réussie, soit par le caractère "-" dans le cas contraire.

Ce caractère est suivi par une valeur qui correspond au statut de la réponse. Le deuxième mot de chaque ligne correspond au numéro de la requête. Une ligne de réponse peut comporter d'autres mots, qui sont appelés les arguments de réponse.

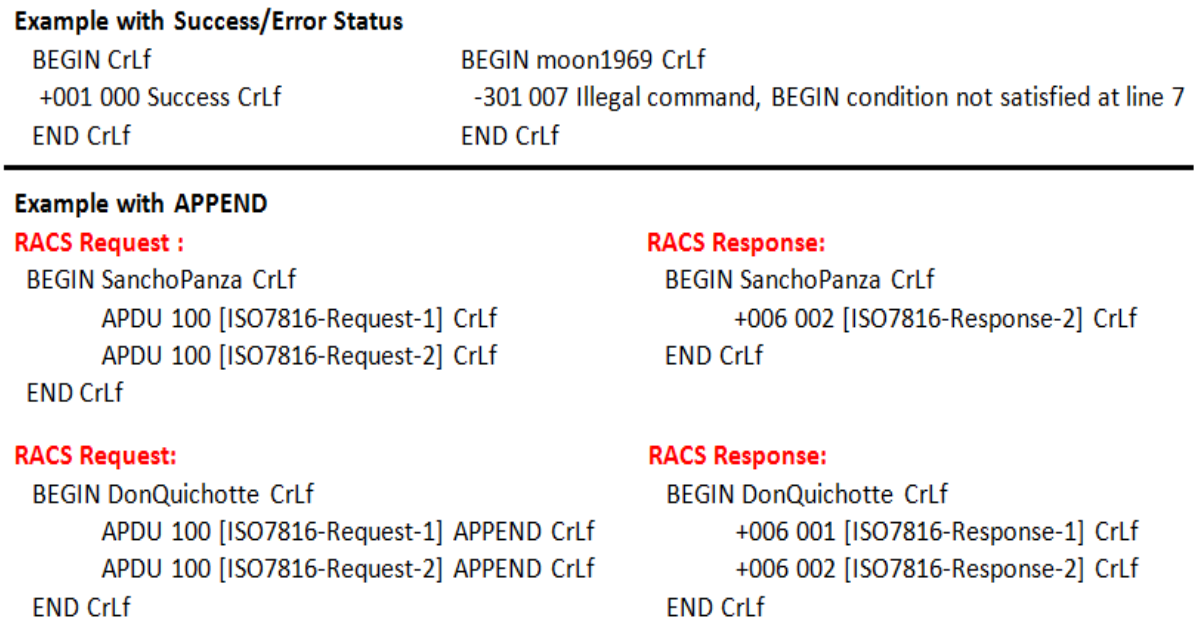

Figure 53 : Exemples de Requêtes/Réponses de RACS

En plus de **BEGIN** et **END**, plusieurs classes d'en-têtes ont été définies. **APPEND** peut être utilisé dans toutes les lignes de commande, sauf devant **BEGIN** ou **END**.

Quand un utilisateur envoie plusieurs requêtes en même temps, la réponse par défaut contient uniquement une ligne de réponse qui correspond à la dernière ligne de requête. Lorsque **APPEND** est inséré, les lignes de commande, si elles sont exécutées, produisent une ligne de réponse correspondant à chaque requête.

6.6.2) Les commandes RACS Client

Le RACS Client envoie une requête ISO7816 pour un élément sécurisé ou un ensemble de commandes ISO7816 sous la forme suivante (Figure 54) :

```
Request:

  BEGIN CrLf
    APDU SEID ISO7816-REQUEST CrLf
  END CrLf

Response:

  BEGIN CrLf
    +006 001 ISO7816-RESPONSE CrLf
  END CrLf
```

Figure 54 : Exemples de Requête/Réponse du RACS Client/Server

- **APDU** : indique au RACS Server qu'il s'agit d'une commande ISO7816. Le SEID (Secure Element Identifier) spécifie l'identifiant unique indiquant la position de l'élément sécurisé dans un slot de la grille, et désigne implicitement l'emplacement du module physique (SlotID) dans lequel la carte à puce est branchée. ISO7816-**REQUEST** est la commande à exécuter. Trois paramètres sont disponibles en option. Ils doivent être situés après la commande ISO7816-REQUEST :
  - **CONTINUE = value :** indique que la commande RACS suivante ne sera exécutée que si le mot d'état ISO7816 (SW) est égal à la valeur affectée à **CONTINUE**. Sinon, un statut d'erreur est retourné.
  - **KETCH = valeur** est une requête qui permet de récupérer la réponse d'un élément de sécurité quand celle-ci est de taille supérieure à 240 octets. À chaque requête, elle fixe les quatre premiers octets de la requête APDU ISO7816 (i.e. CLA INS P1 P2) pour demander le reste de la réponse. La valeur par défaut de l'APDU, lorsque la commande Fetch est omise, est CLA = 00, INS = C0, P1 = 00, P2 = 00.
  - **MORE = value** indique qu'une commande FETCH sera exécutée (i.e. une nouvelle requête ISO7816 sera envoyée) à condition que le premier octet du mot d'état ISO7816 (SW1) soit égal à la valeur affectée à **MORE**.

6.6.3) Utilisation des fonctionnalités de base avec RACS

Un ensemble de fonctionnalités de base a été implémenté dans le Proxy Server. La couche logicielle RACS Server est un parseur du protocole RACS. Elle permet de traiter et d'extraire les commandes ISO7816 du RACS Client ; elles sont ensuite récupérées par la couche API EAP-TLS, laquelle les transmettra à la carte à puce. Parmi ces fonctionnalités, citons les exemples suivants (Figure 55) :

- **GET-VERSION** : C'est une requête pour demander la version actuelle du protocole RACS. La réponse renvoyée est codée sur deux entiers séparés par le caractère '.'. Le premier entier indique la version majeure et le second indique la version mineure.
- **SET-VERSION** : Cette commande définit la version à utiliser pour RACS. En cas d'erreur, un statut d'erreur est renvoyé dans la réponse.
- **RESET** : Cette commande réinitialise un élément sécurisé (Figure 55). Le premier paramètre indique l'identifiant de l'élément sécurisé (SEID = device#45). Si un second paramètre (WARM) optionnel est spécifié, il force l'élément sécurisé à se réinitialiser à chaud. Le comportement par défaut est une réinitialisation à froid quand l'élément sécurisé n'est pas occupé. Le statut de réponse indique le succès ou l'échec de cette opération.

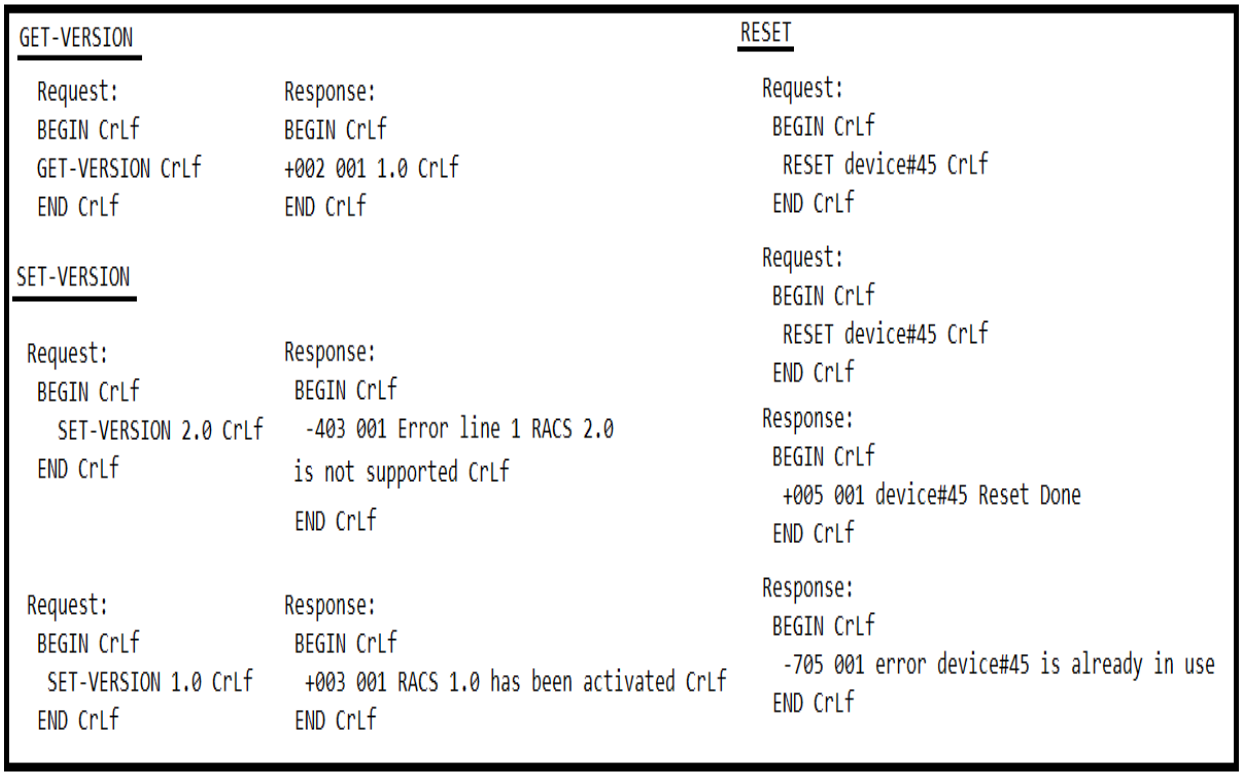

Figure 55 : Exemples de commandes de base avec RACS

6.6.4) Illustration d'une session avec la GoSE

La Figure 56 illustre le début d'une session avec une carte à puce déportée sur une grille distante et un Proxy Client exécuté sur une ressource distante. Le but de ces échanges

est de démarrer une application embarquée dans une carte à puce distante, identifiée par son AID (Application Identifier) selon la norme ISO7816. Cette carte est située dans le slot 02, SEID 702, de la grille IMPLEMENTA.

```
Connected to the Grid Server !!!
001 implementa SIM Server 3.5 on simarray (Revision 5.4) ready.
USE 702
011 3B7D96000080318065B08311C0C883009000
Smartcard Inserted...Protocol T=0
ATR: 3B 07 47 52 49 44 30 30 32
Reader> 00 A4 04 00 10 A0 00 00 00 30 00 02 FF FF FF FF 89 31 32 38 00
Tx      : 00 A4 04 00 10 A0 00 00 00 30 00 02 FF FF FF FF 89 31 32 38 00
GridTx : apdu 702 00A4040010A0000000300002FFFFFFFF8931323800
>> 156 ms
GridRx : 013 9000
Rx      : 90 00
Reader>
```

Figure 56 : Illustration d'une session avec la Grille IMPLEMENTA

6.6.5) Administration de la grille

L'administration de la grille est un sujet sensible, qui a suscité des discussions dans le consortium SecFuNet, car les éléments sécurisés ne sont pas nécessairement gérés par le prestataire de services de la grille.

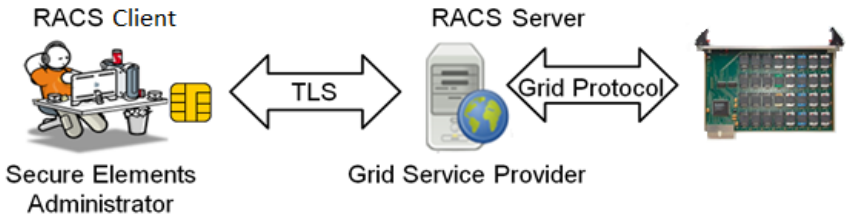

Figure 57 : Administration de la grille des éléments sécurisés

Actuellement, les éléments sécurisés sont équipés de clés secrètes symétriques utilisées à des fins d'authentification mutuelle. L'application embarquée dans les cartes à puce permet de charger, d'activer ou de supprimer des outils cryptographiques à distance, grâce à des canaux sécurisés. Ces fonctionnalités intégrées dans ces éléments sécurisés peuvent lier une identité (comme ID_{VM}, ou ID_{HV}) à une carte à puce et protègent les informations qu'elles produisent dans des conteneurs, qui ne peuvent être décryptés que par l'entité autorisée (i.e. XEN ou VM). Cependant, un accès spécifique est nécessaire pour permettre ces opérations. Notre Proxy pourrait être utilisé avec le protocole RACS pour transporter les commandes

ISO7816 en toute transparence et les convertir dans un format compatible avec le protocole de la grille (Figure 57).

6.6) Conclusion

Dans ce chapitre, nous avons implémenté quelques briques qui visent à répondre aux besoins d'établir des liens logiques pour gérer les cartes à puce. Cette implémentation architecturale tient compte des besoins fonctionnels du Cloud Computing, mettant en avant les avantages d'une authentification mutuelle forte de bout en bout, en utilisant le protocole TLS, pour offrir des relations de confiance par un tiers qui pourrait se décliner sous la forme d'un nouveau service (i.e. TaaS).

En effet, une grille de cartes à puce reliée à un serveur d'authentification peut alors être déportée ou administrée par une autorité tierce pour offrir des services TaaS, mais elle peut également être gérée par le client lui-même, s'il souhaite garder le contrôle de la sécurité de son infrastructure.

Grâce à la carte EAP-TLS qui réunit de nombreux avantages et en s'appuyant sur notre modèle architectural relativement simple, nous pouvons imaginer une multitude de possibilités de déploiement de cette architecture. L'objectif est d'offrir une solution crédible pour relever certains défis relatifs aux problèmes de gestion des identités, relevés dans l'état actuel des environnements virtuels.

Il est donc possible d'intégrer ces briques architecturales et ces composants sécurisés dans des infrastructures d'authentification avancées pour préserver la vie privée des utilisateurs en s'appuyant sur des modèles de gestion des identités centrés sur l'utilisateur et des modèles de contrôle des attributs "User-Centric Identity/Attributes Control" [1].

Nous avons construit notre travail autour de ces briques architecturales, proposé un nouveau modèle architectural modulaire pour faire face aux problèmes de gestion des identités des ressources décentralisées dans une architecture transorganisationnelle et favorisé le principe de coopération entre les acteurs du Cloud selon leur rôle. Ces acteurs seront capables de répondre aux besoins d'authentification de leurs partenaires en toute confiance.

Ce modèle architectural constitue une couche d'abstraction pour piloter d'une manière transparente le système de gestion des identités indépendamment du niveau de visibilité. Nos briques architecturales s'articulent autour d'un ensemble de composants logiciels (Proxys) qui

exploite de manière efficace les avantages sécuritaires de ces technologies (i.e. cartes à puce, GoSE), les méthodes et les protocoles d'authentification qui sont initialement indépendants.

Chaque Proxy est exécuté en tâche de fond, contrôle en temps réel, réagit dynamiquement et de manière autonome à chaque demande d'une session, pour garantir à deux entités une authentification mutuelle forte. En outre, nous avons proposé et implémenté une couche protocolaire RACS au-dessus du protocole TLS. Celle-ci a pour rôle d'encapsuler des requêtes ISO7816 (APDUs), de les acheminer efficacement d'une ressource vers un élément sécurisé pour qu'il réalise des opérations cryptographiques indépendamment de la technologie de la grille.

Les tests réalisés avec les cartes à puce montrent bien que les temps de réponses affichés sont acceptables par rapport aux contraintes d'utilisations courantes. Une session en mode complet (*Full mode*) engendre certes un surcoût au niveau de la carte à puce, mais une fois que l'utilisateur est connecté à un serveur les requêtes envoyées vers les services hébergés par ce dernier sont moins coûteuses car généralement les connexions utilisées sont en mode reprise.

Chapitre VII : Kit de développement logiciel, supportant les cartes à puce dans OpenSSL

7.1) Introduction

L'adoption généralisée de l'Internet comme moyen de communication et l'utilisation de la cryptographie comme une composante essentielle des systèmes d'information modernes pour établir des relations de confiance et sécuriser les transactions commerciales, sont devenues des axes de recherche pour concevoir l'Internet de demain. La carte à puce est de plus en plus populaire comme un moyen d'identification personnel et d'authentification dans de nombreux domaines d'application sécurisés (banques, commerces, téléphonie, etc.). Pour répondre à ces besoins de sécurité, le projet de SecFuNet vise à généraliser l'usage des éléments sécurisés, afin d'améliorer la fiabilité des offres de services Internet.

Le projet OpenSSL [68] offre plusieurs possibilités pour développer des applications sécurisées en utilisant des bibliothèques et des fonctionnalités cryptographiques. Malheureusement, le stockage des clés privées sur un disque dur pose un vrai problème de vulnérabilité, car il y a de fortes chances qu'un administrateur (ou un Root-Kit avec des privilèges élevés) puisse facilement dérober une clé privée.

En revanche, si cette clé est stockée dans une carte à puce, il devient extrêmement difficile de la récupérer ou d'attaquer physiquement une carte à puce. Le vol de la carte ne représente donc qu'un risque très limité dans le temps, car il faut que l'attaquant vole aussi le code PIN et même si c'est le cas, les certificats seront immédiatement révoqués et les données stockées seront déclarées corrompues.

L'objectif principal du projet SecFuNet est de développer une infrastructure sécurisée ouverte (basée sur l'Open Source) pour les environnements virtualisés et les Clouds. Cette infrastructure fournit non seulement une authentification mutuelle forte, mais fournit également un ensemble de fonctionnalités sous forme de bibliothèques pour s'interfacer avec des éléments de sécurité.

Ce chapitre a pour vocation de décrire l'intégration de la norme PC/SC dans OpenSSL et l'implémentation de nouvelles bibliothèques pour supporter les échanges avec les cartes à puce. L'objectif de ces fonctionnalités ajoutées et intégrées à OpenSSL, est de renforcer la sécurité des applications sensibles et de banaliser ou démystifier l'usage des cartes à puce dans le développement de celles-ci. Ces nouvelles bibliothèques sont sous forme d'un kit de

développement logiciel dédié, en particulier à notre machine à états (i.e. carte EAP-TLS) et de plusieurs commandes en ligne pour s'interfacer avec cette carte (§7.3).

À présent, nous allons décrire le projet OpenSSL, ses APIs et ses outils cryptographiques.

7.2) Le projet OpenSSL

OpenSSL [68] est un projet collaboratif qui offre des outils cryptographiques pour développer des applications robustes. Le projet est géré par une communauté de volontaires, qui développe les outils OpenSSL et la documentation associée.

OpenSSL est basé sur la bibliothèque SSLeay développée par Éric A. Young et Tim J. Hudson. Il est distribué sous licence (Apache-style License), ce qui signifie essentiellement que vous êtes libre de l'utiliser à des fins commerciales et non commerciales.

OpenSSL est un ensemble d'outils cryptographiques qui est utilisé pour le chiffrement/déchiffrement des données, le calcul des hachés, etc. Son code source est libre (Open Source) et met en œuvre les protocoles d'authentification SSL v2/v3 et TLS V1.x, ainsi qu'une bibliothèque de cryptographie à usage général et des standards cryptographiques connexes. Cette bibliothèque cryptographique est très complète et très répandue dans des applications sensibles (i.e. e-commerce, e-banque, etc.). Elle est utilisée dans plusieurs protocoles et applications pour sécuriser les connexions : OpenSSH, IPSEC, HTTPS, IMAPS, POPS, SFTP...

OpenSSL offre aussi des commandes en ligne pour utiliser ses différentes fonctions cryptographiques à partir du Shell, pour créer et gérer des clés asymétriques et/ou symétriques, des certificats X509, des CRLs (listes de révocation de certificats), des opérations cryptographiques à clé asymétrique. Il offre en particulier les deux commandes en ligne *s_client et s_server* qui constituent deux applications Client/Serveur pour tester le protocole TLS ou SSL.

7.3) Kit de développement logiciel pour cartes à puce

OpenSSL offre également la possibilité de faire des opérations cryptographiques dans des modules spécifiques, habituellement, pour des raisons de performance et de sécurité. La première raison (performance) est assez évidente : un matériel spécialisé peut faire du calcul

cryptographique beaucoup plus rapidement qu'un ordinateur à usage général. La seconde raison est généralement pour préserver les clés cryptographiques.

Il est donc essentiel de conserver ces clés de manière sécurisée pour en rendre difficile l'accès et en interdire la copie. Une solution consiste à lier OpenSSL à des dispositifs de sécurité comme : un TPM [49], une carte à puce, un jeton USB, ou un HSM. La clé est alors stockée à l'intérieur d'un tel dispositif et les opérations de chiffrement sont également effectuées au sein de certains éléments de sécurité. Ainsi, de nombreuses applications ont été écrites pour supporter ce type de périphérique à des fins diverses, en utilisant les bibliothèques OpenSSL.

Or, OpenSSL n'offre pas de bibliothèques pour prendre en charge des cartes à puce. Par conséquent, les développeurs d'applications doivent consacrer du temps à la création de leur propre kit de développement logiciel (ou *SDK : Software Development Kit*) pour accéder aux APIs PC/SC. Un travail fastidieux et une perte de temps, qui serait mieux utilisé pour le développement de l'application elle-même.

Notre approche consiste à intégrer un SDK dans OpenSSL pour supporter les cartes à puce et à définir une interface applicative de programmation (API) pour ces éléments sécurisés. L'objectif de ce SDK (nommé Smart Card-SDK-API : SC-SDK-API) est de rester indépendant de la technologie utilisée et de permettre le partage de ces périphériques par plusieurs applications. Il permet de faire une abstraction entre les couches applicatives et les opérations liées au périphérique et d'éviter de maintenir ses propres mises à jour, de les modifier et de les recompiler à chaque fois, avec le risque de casser certaines dépendances d'OpenSSL ou du système d'exploitation.

Ce SC-SDK-API met en œuvre des fonctions pour encapsuler les commandes ISO7816 et le protocole RACS afin d'accéder à plusieurs types de cartes par le biais des APIs PC/SC. Il propose également aux programmeurs une logique commune de gestion des cartes à puce. Cette logique consiste en l'implémentation d'un ensemble de fonctionnalités primitives pour manipuler ces périphériques et faciliter le développement d'applications sensibles.

Un développeur n'a plus besoin de créer sa propre bibliothèque, il peut donc réutiliser ce SDK commun à la plupart des cartes à puce de façon homogène car il lui fournit un niveau d'abstraction par rapport aux commandes ISO7816.

Pour l'instant, nous avons développé le SDK pour supporter en particulier la carte EAP-TLS, mais de nouvelles fonctionnalités ou algorithmes peuvent être rajoutés sans pour autant en changer la structure.

7.3.1) Structure du kit de développement logiciel

SC-SDK-API respecte la norme PC/SC et fournit un cadre qui peut être étendu pour supporter toutes les authentifications par cartes à puce, en utilisant, par exemple une carte SIM 2G (RFC 4186) ou UMTS (RFC 4187). Il suffit que le lecteur de carte à puce soit compatible avec la norme PC/SC. Notre SDK est disponible sous Linux, sous forme de primitives implémenté en langage de programmation ANSI C, développées et déjà utilisées dans le projet SecFuNet, situé dans l'arborescence du code source OpenSSL (Figure 58).

Les changements sont mineurs dans OpenSSL pour ne pas influer sa structure. En effet, les modifications concernent uniquement les fichiers *Makefile (Annexes E-1*) pour permettre au compilateur de construire OpenSSL avec le kit SC-SDK-API.

Les autres modifications se situent dans le sous-répertoire (/openssl/scard_pcsc_api) qui contient tous les fichiers de notre SDK. Nous avons développé deux applications (Client/Server) supplémentaires pour effectuer des tests (*s_scard et s_hypervisor*) qui se trouvent dans le sous-répertoire (*/openssl/apps*). Ces deux applications s'appuient sur un ensemble de procédures se trouvant dans le sous-répertoire (*/openssl/ssl*) pour gérer les deux phases du protocole TLS.

Le programme*/openssl/apps/s_scard.c* est une commande en ligne ; dérivée à partir de la commande *s_client*, elle permet de s'interfacer avec la machine à états EAP-TLS, pour tester des connexions avec des serveurs TLS classiques (§7.3.2). Cette commande peut jouer le même rôle que le Proxy Client évoqué au Chapitre VI (§6.3.1). Elle appelle des procédures implémentées dans le fichier*/openssl/ssl/s3_scard_ctnt.c* pour négocier des authentifications mutuelles.

Le programme*/openssl/apps/s_hypervisor.c* est une autre application en ligne, basée sur la commande *s_server*. Cette commande joue le rôle de Proxy Server et/ou Proxy RACS pour gérer et négocier également des sessions TLS avec la grille de cartes à puce. Elle appelle des procédures implémentées dans les fichiers :*/openssl/ssl/s3_vm_io_xen_ctnt.c*, et*/openssl/ssl/s3_racs_ctnt.c*.

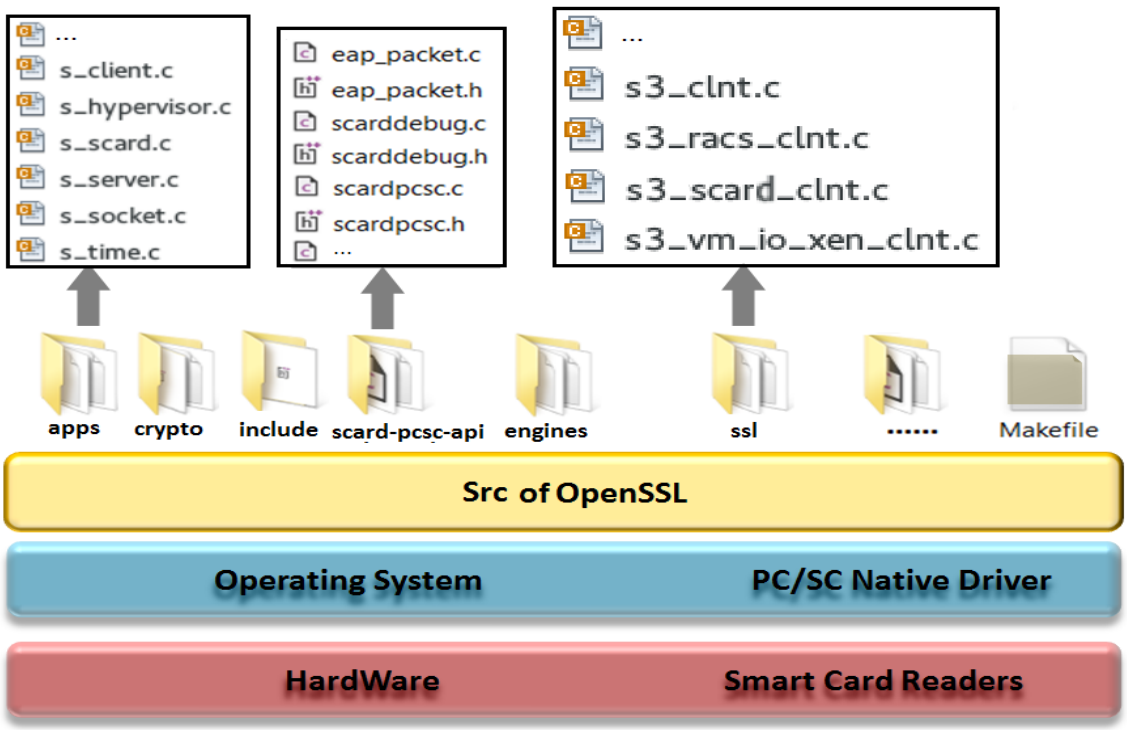

Figure 58 : Architecture logicielle du SDK dans OpenSSL

Ces deux programmes (*Client/Serveur*) spécifiques aux cartes à puce EAP-TLS, permettent de dérouler intégralement les échanges Handshake (Phase I).

Ce SDK dispose de nombreuses fonctionnalités avancées. Le sous-répertoire*/openssl/scard_pcsc_api*, contient plusieurs fichiers dans lesquels nous avons implémenté toutes les procédures de gestion des cartes à puce.

Le fichier *scarddebug.c* par exemple, contient les procédures de gestion des événements et de débogage. Le fichier *scardpcsc.c*, contient toutes les procédures et méthodes qui permettent de gérer les lecteurs et de se connecter à une carte à puce et, donne la liste des lecteurs et des cartes à puce disponibles.

Par exemple, la procédure *scard_make_eaptls_msg_output*, est utilisée pour initier une authentification en mode complet (*Full Mode*) avec la machine à états EAP-TLS, comme défini dans la RFC 4137.

7.3.2) Authentification avec la commande en ligne "*s_scard*"

Les commandes en ligne *s_scard et s_hypervisor* sont implémentées de telle sorte à être facilement personnalisables et/ou modifiables en fonction des besoins de chaque

développeur, en particulier grâce à l'énumération *scard_sim_type* qui permet de choisir le type de carte à puce à utiliser dans le processus d'authentification TLS :

typedef enum {SCARD_GSM_SIM, SCARD_USIM, SCARD_EAPTLS} scard_sim_type ;

Pour tester un nouveau périphérique de sécurité compatible avec la norme PC/SC, les développeurs peuvent, en fonction de leurs exigences et spécifications, récrire un nouveau code en se basant sur les fonctionnalités et les algorithmes de gestion de la carte implémentés dans le SDK.

Par ailleurs, l'utilisation en ligne de commande, des deux applications *s_scard et s_hypervisor*, spécifiques à la machine à états EAP-TLS, permettent une authentification mutuelle entre la carte à puce d'un client et un serveur TLS classique ou une authentification carte à carte (M2M) entre une carte à puce d'un client et une carte à puce dans une grille gérée par l'application *s_hypervisor*. Dans tous les cas, le déroulement des échanges du Handshake est transparent pour un utilisateur, sauf au moment de saisir son code PIN.

La Figure 59 illustre une connexion avec un serveur TLS classique et la commande en ligne *s_scard*. La plateforme se compose des éléments suivants : un serveur TLS classique, une carte à puce EAP-TLS reliée à un poste client.

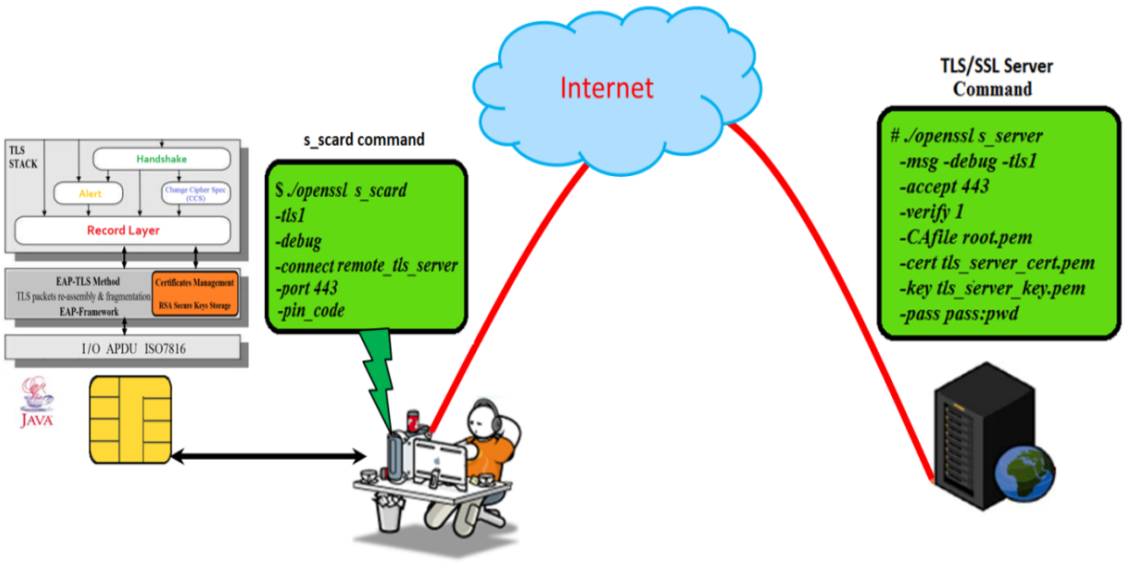

Figure 59 : La Plateforme de test de la commande *s_scard* avec *s_server* d'OpenSSL

Pour lancer le test dans notre cas d'usage, il faut exécuter, en tâche de fond, un serveur TLS. Par exemple la commande en ligne *s_server* des API OpenSSL, avec les arguments

suivants, pour ouvrir le port 443 sur l'adresse IP locale du serveur TLS et les certificats associés pour établir la connexion :

#./openssl s_server -msg -debug -tls1 -accept 443 -verify 1 -CAfile root.pem -cert tls_server_cert.pem -key tls_server_key.pem -pass pass : pwd

La commande en ligne *s_scard* doit être exécutée avec les arguments suivants, pour ouvrir une session TLS avec le serveur OpenSSL (remote_tls_server) sur le port 443 :

#./openssl s_scard -tls1-debug -connect remote_tls_server -port 443 -pin_code

L'utilisateur a juste besoin du nom ou de l'adresse du server TLS, et de son code PIN. Les autres arguments supportés par la commande OpenSSL *s_client* sont aussi intégrés dans *s_scard* et peuvent être fixés dans le programme par le développeur.

7.3.3) Déroulement d'une authentification avec "*s_scard*"

L'ensemble des bibliothèques du SDK a permis de développer la commande *s_scard/s_hypervisor* et de gérer efficacement la machine à états embarquée dans la carte à puce. Nous allons maintenant détailler l'implémentation de la commande *s_scard* puis illustrer le scénario de son fonctionnement (Figure 60) :

✓ Smart Card Initialization : cette étape permet de choisir le type de la carte, en initialisant la variable *sim_type*, de détecter la carte à puce du client, de démarrer l'application Java Card identifiée par son AID (i.e. EAP-TLS) et de charger ensuite l'identité de l'utilisateur et de demander son code PIN. Ces tâches sont assurées par la fonction *scard_init_eap_tls ()*. En cas de succès celle-ci retourne le mot "S*W1 SW2 = 90 00*". Dans le cas contraire la commande *s_scard* échoue.

✓ Setup TCP/IP Socket : création et initialisation, au niveau du client SSL, du Socket (TCP/IP) pour se connecter au serveur SSL afin d'échanger les messages d'authentification.

✓ Handshake : après ouverture du port de connexion avec le serveur, c'est la fonction SSL_connect qui rentre en jeu pour assurer le processus d'échange du *Handshake*, lequel prend quant à lui en charge les tâches de gestion de la connexion :

- Négociation et activation des paramètres algorithmiques (cryptographie et compression),

- Choix du CCS (Change Cipher Spec), pour la synchronisation cryptographique (passage des paramètres cryptographiques courants aux paramètres futurs négociés par le Handshake protocol),

✓ Data Communication : une fois la phase du Handshake terminée, la phase II de chiffrement/déchiffrement des données applicatives commence après dérivation de la clé éphémère "*KEY BLOCK*".

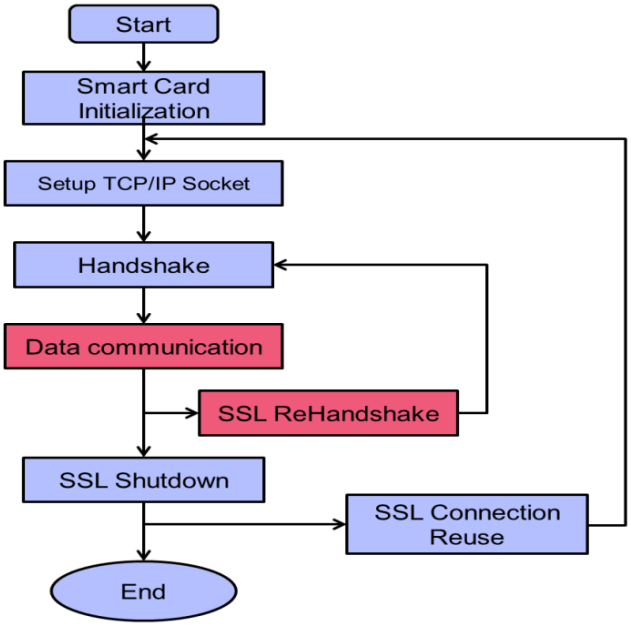

Figure 60 : Scénario de fonctionnement de la commande *s_scard*

Nous allons à présent nous intéresser au fonctionnement de *SSL_connect* qui fait appel à plusieurs procédures pour s'interfacer avec la carte à puce afin de respecter les échanges Handshake :

- *scard_send_client_hello ()* : génère 4 octets qui représentent le "*GMT Unix Time*" (RFC 2246), ensuite ils sont envoyés sous forme d'un message EAP-TLS (Tx) à la carte à puce, car celle-ci ne possède pas d'horloge interne (Figure 61).

Une fois ce Tx est reçu, la carte génère un nombre aléatoire de 28 octets le concatène avec les 4 octets et retourne un message (Rx) "*Client Hello*" encapsulé dans un message EAP-TLS. Le paquet Rx (Figure 62) est une réponse de la carte à puce qui se compose d'une trame EAP (en vert) d'une taille de dix octets, suivie de l'en-tête "*Record Layer*" du message "*Client*

Hello", et de deux octets en fin de trame, correspondant au mot (SW1SW2 = 90 00), indiquant le succès de la réponse de la carte à puce.

	Commande APDU					EAP : Extensible Authentication Protocol						GMT Unix Time			
	Class	INS	P1	P2	Lc	Code	ID	Length		Type	Flag				
Tx	A0	80	00	00	0A	01	14	00	06	TLS = 0D	20	55	1A	C8	B2

Figure 61 : Le paquet Tx avec le GMT Unix Time

Rx : 02 14 00 5C 0D 80 00 00 00 52 16 03 01 00 4D 01 00 00 49 03 01 55 1A C8 B2 F9 73 74 C2 78 E8 87 BF 30 49 40 31 90 47 2C 2B 1E F5 82 0B 6F 93 85 80 4A 15 CF 37 20 16 D2 0C BD 82 77 BA 8F FC 1D 3E D2 8D 8C B9 31 2F 7B D3 9F 31 0A D5 60 52 19 21 FE 9D 36 A7 EF 00 02 00 04 01 00 90 00

Figure 62 : Le paquet Rx : réponse de la carte à puce

Ensuite, la trame EAP et le mot SW1SW2 sont retirés pour n'envoyer que le *Record Layer* et le message *Client Hello* (Figure 63) au serveur TLS distant en utilisant le Socket déjà négocié, après initialisation de la structure SSL du client TLS.

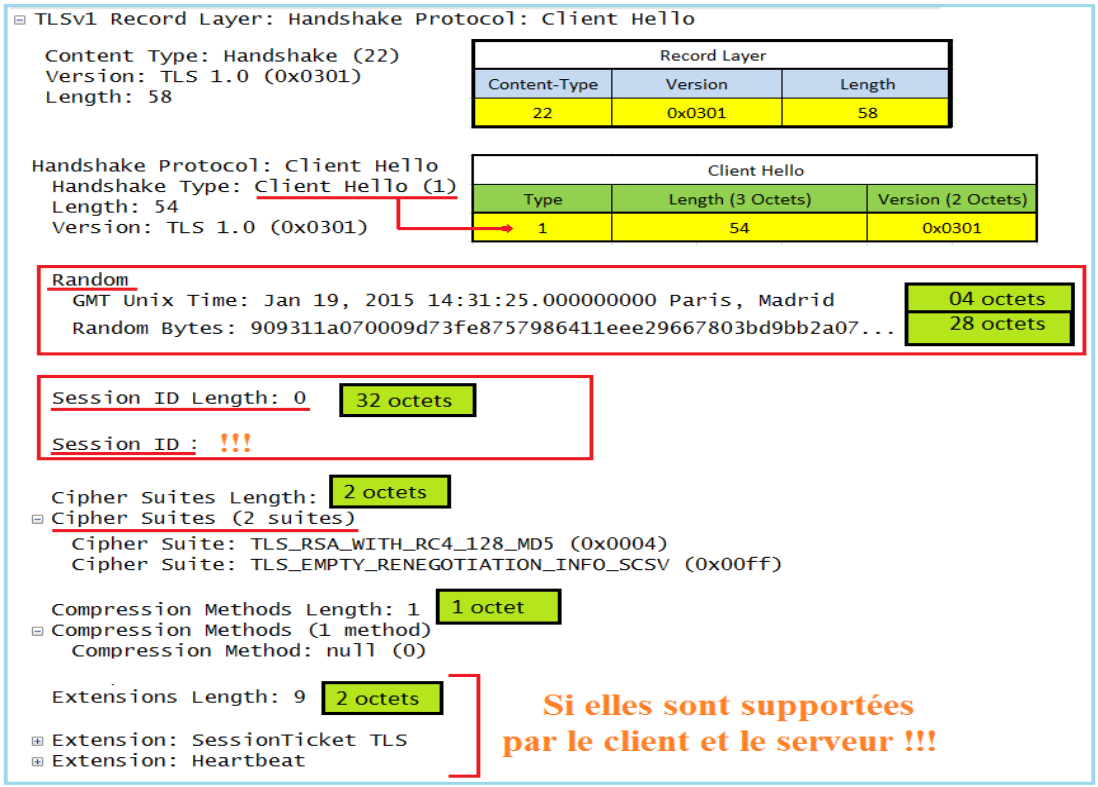

Figure 63 : Détail du message Client Hello

- *scard_get_server_hello ()* : cette fonction récupère le message *Server Hello*, initialise la structure SSL du client TLS, et fixe les paramètres d'état (Figure 64) : *Random, Session_ID, Cipher Suite,* et la méthode de compression.

Les fonctions suivantes permettent de récupérer les autres messages envoyés par le serveur TLS et d'initialiser la structure SSL du client TLS :

- *scard_get_server_certificate ()* : récupère le message *ServerCertificate*.
- *scard_get_key_exchange ()* : récupère, éventuellement le message *Server Key Exchange*, si les messages précédents ne contiennent pas suffisamment d'information, pour que le client puisse échanger le Pré-Master Secret.
- *scard_get_certificate_request ()* : le serveur envoie le message *Certificate Request* qui comprend une liste des types de certificats pris en charge et les "Distinguished Names" acceptables par les Autorités de Certification (CA).

- *scard_get_server_done ()* : récupère le message *ServerHelloDone* qui indique au client TLS que le message *ServerHello* est terminé et que le serveur est en attente d'une réponse du client. Ce message n'a pas de contenu.

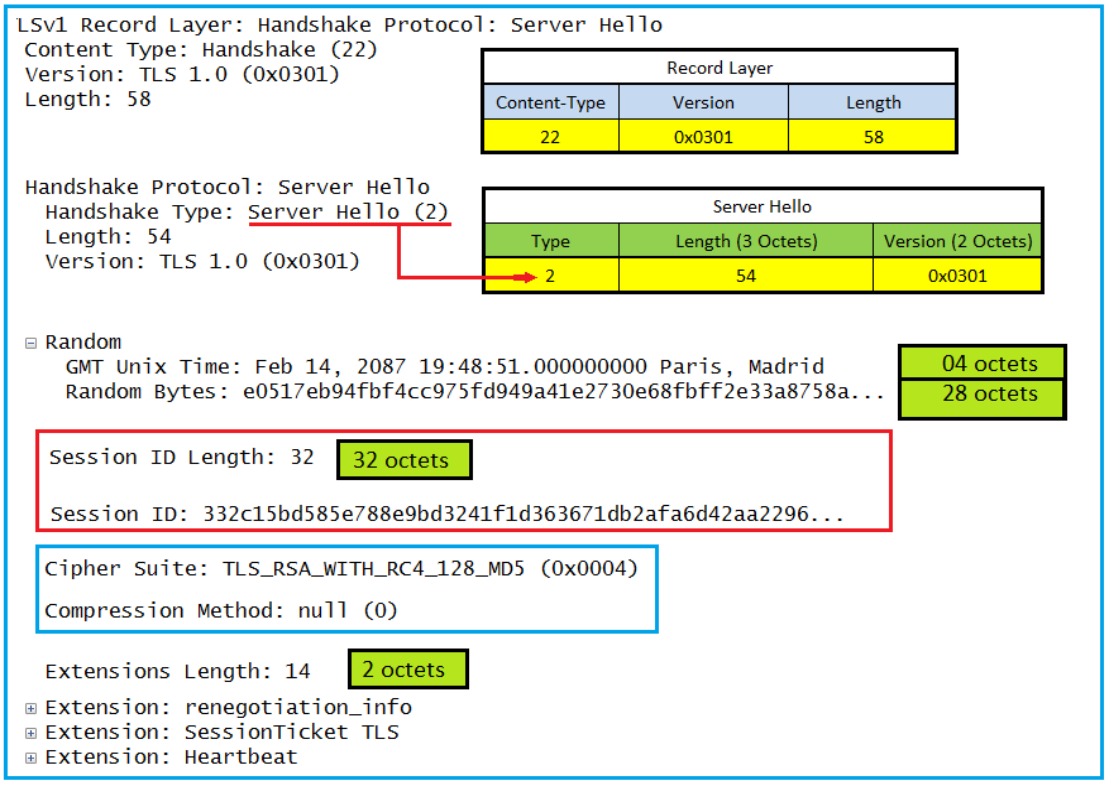

Figure 64 : Détail du message Server Hello

La fonction *scard_save_serverhello_resp (SSL *s, struct scard_data *scard)*, est utilisée pour récupérer tous les messages du serveur TLS, et initialise la structure *scard*.

Une fois que le message *ServerHelloDone* est reçu par le client TLS, la fonction *scard_send_client_certificate ()* fragmente les messages sauvegardés par la fonction précédente et les transmet à la carte à puce sous forme de paquets EAP-TLS de 256 octets (*Figure 65)* en faisant appel à la fonction *scard_make_eap_tls_output_packet ()* :

```
Tx : A0 80 01 00 F0 01 00 07 62 0D 00 16 03 01 00 4A 02 00 00 46 03 01 55 1A A9
     DD F7 28 03 D3 36 FD C9 CF 38 F0 AA D7 66 33 DE 33 E8 9A 5D 04 C1 12 4E
     20 E9 AD 92 5E 20 4A F8 1B A7 9A 95 B0 48 6E 01 87 F1 BC 98 15 CA B4 99
     BC 77 7F 4A 14 13 66 55 0E .. .. .. ..
```

Figure 65 : Fragment d'un message

En orange la commande APDU, suivie d'un paquet EAP (en bleu), ensuite l'en-tête *Record Layer* et les messages reçus du serveur TLS. Une fois ces messages acheminés vers la carte à puce, celle-ci les vérifie et en cas de succès, elle retourne les messages suivants encapsulés dans des paquets EAP :

- *ClientCertificate* : Ce message se compose, comme le certificat du serveur, d'une séquence de certificats X509 v3, codée au format ASN.1 (Abstract Syntax Notation One), un standard international spécifiant une notation destinée à décrire des structures de données dans le secteur des télécommunications et des réseaux informatiques. La description en ASN.1 d'une structure de données a pour but d'obtenir une spécification de la structure qui est indépendante d'un encodage lié à un matériel particulier.

- *ClientKeyExchange* : contient la clé matérielle nécessaire pour sécuriser la communication. Le format du message dépend des messages précédents et de l'algorithme utilisé par les deux parties. Dans notre contexte, la génération du Pré-Master Secret est une concaténation de deux octets correspondant à la version TLS (0x0301) et une valeur aléatoire de 46 octets chiffrée par la clé publique du serveur. Cette protection est utilisée pour détecter les attaques de régression de version.

- *ClientVerify* : Ce message est utilisé pour fournir une vérification explicite du certificat client. Il indique au serveur que le client est en effet le titulaire de ce certificat. Il se compose d'un haché (Digest) signé de tous les messages Handshake envoyés et reçus depuis le début de la négociation. Seul le client détenteur de la clé privée peut signer ce message, et

c'est là que réside la vérification du certificat client. Dans notre cas (clés RSA), le format de la signature est une concaténation de MD5 et SHA1.

- *scard_parse_certificate_request ()* n'extrait que les messages TLS en provenance de la carte, et les envoie au serveur TLS. Ensuite, l'ensemble des messages du Handshake TLS entre le serveur et la carte à puce a lieu jusqu'aux messages *ChangeCipherSpec et Finished*. La réception du message *ChangeCipherSpec* indique le début de chiffrement des échanges en utilisant les outils cryptographiques négociés dans les phases précédentes. Lorsque ce message est envoyé par le client ou par le serveur, les deux changent d'état pour passer du mode "*Current*" au mode "*Pending*" en activant le mode de chiffrement. Quand le client ou le serveur le réceptionne, il change d'état en activant le mode de déchiffrement.

- *scard_get_ciphersuite_keyblock ()* récupère la suite de chiffrement *CipherSuite* et la clef *Key-block* de la carte à puce.

- *scard_setup_key_block ()* initialise la structure SSL du client TLS, une fois l'authentification mutuelle réussie, les clés de session sont générées automatiquement grâce à la clé temporaire (*KEY_BLOCK*), puis la carte à puce associée à cette session TLS est libérée pour être utilisée dans une autre session. En effet, la clé *key-block* est dérivée en un ensemble de quatre clés pour vérifier l'intégrité et pour chiffrer/déchiffrer les données, Ces clés sont calculées selon la méthode indiquée au chapitre II (§2.3.3.2). Une fois la phase I du Handshake terminée, la phase II de chiffrement et de déchiffrement des données applicatives commence.

La Figure 66 illustre l'établissement d'une session réussie côté client, avec un serveur TLSv1, une Cipher Suite RC4-MD5, et une clé publique de 1024 bits. Le client TLS envoie le message "*Message sent from TLS Client*" au serveur TLS, après l'avoir chiffré avec la fonction *tls1_enc ()*. Ensuite, les paramètres d'état *Write et Read Sequence* sont incrémentés. On constate sur cette figure que la clé *Master-Key* est initialisée à zéro car elle est stockée dans la carte à puce du client.

La Figure 67 illustre d'une part l'ouverture de cette session TLS du côté serveur et le certificat du client, et d'autre part montre les mêmes paramètres d'état (i.e. Cipher Suite RC4-MD5, *Write et Read Sequence)*. Nous constatons que le Sequence Number est incrémenté et que le message "*Message sent from TLS Client*", envoyé par le Client TLS, a bien été reçu chiffré et ensuite déchiffré.

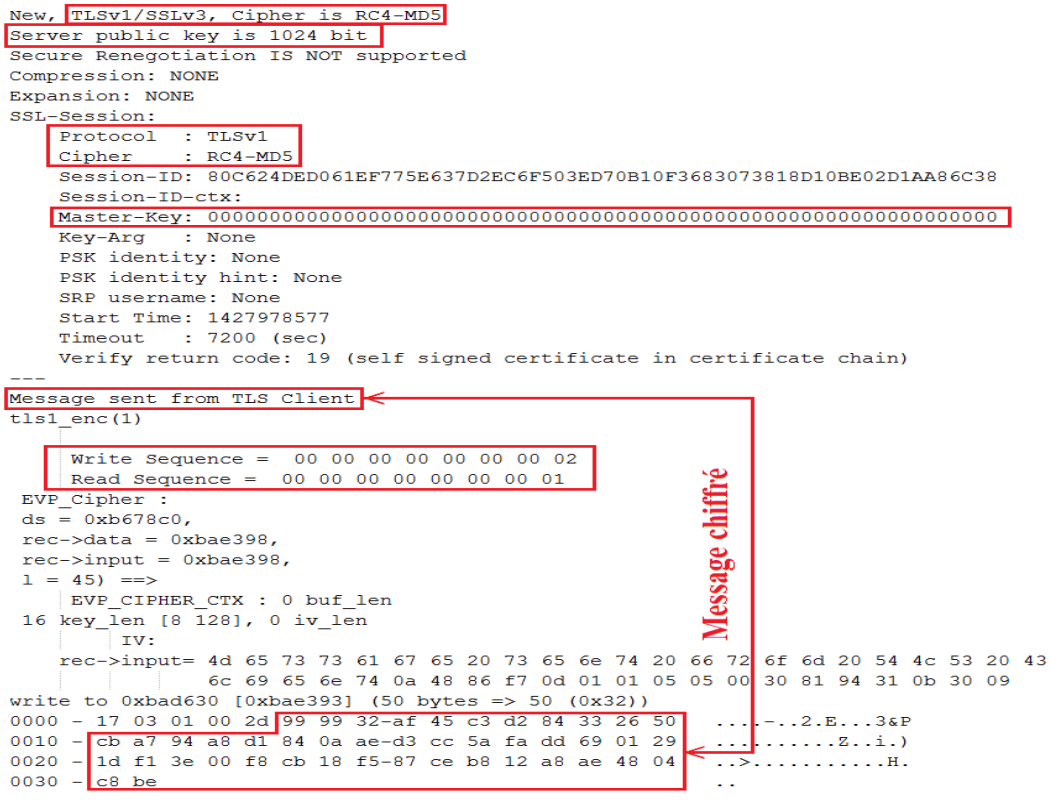

Figure 66 : Les messages de la commande "*s_scard*» côté client

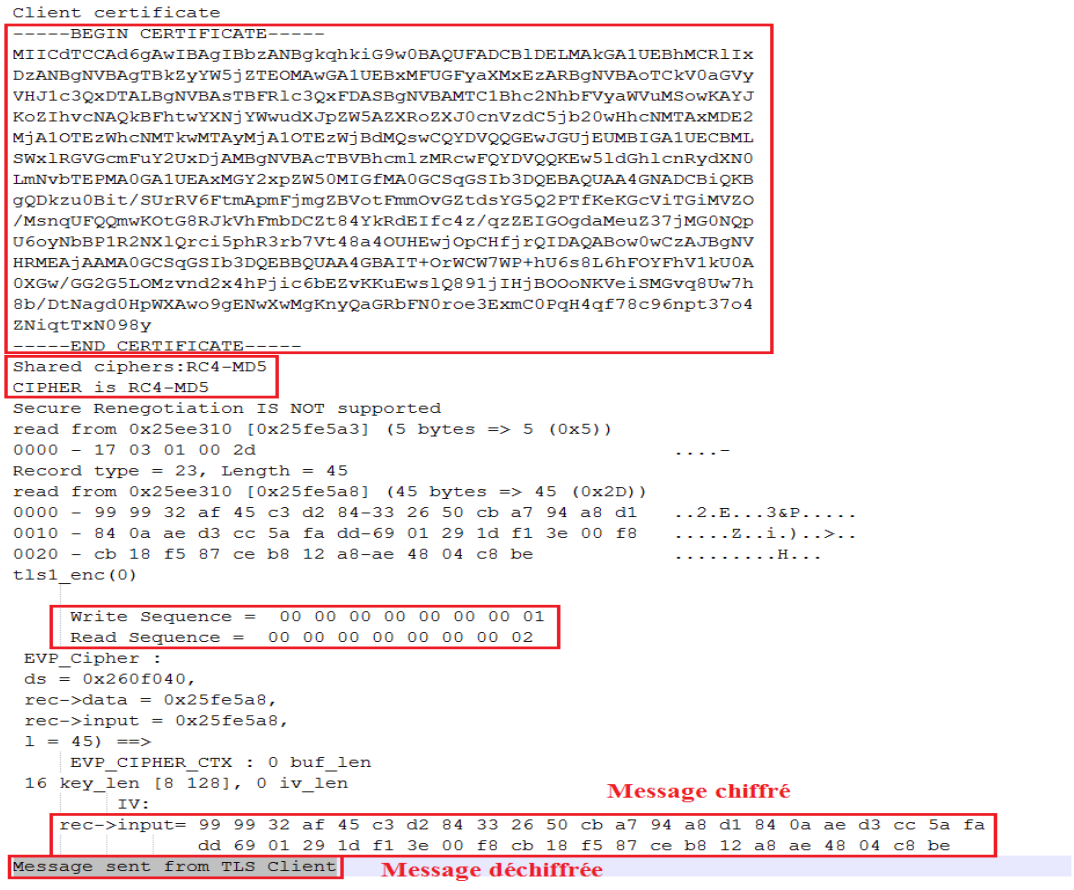

Figure 67 : Les messages du serveur "*s_server*" d'OpenSSL

7.4) Conclusion

L'authentification mutuelle forte est devenue un facteur décisif pour une bonne conception des systèmes d'information décentralisés et sécurisés. La carte à puce est un des moyens pour renforcer la protection de la vie privée des utilisateurs et sécuriser ces systèmes ainsi que les environnements virtuels. Elle dispose d'un ensemble de propriétés indispensables pour la mise en place d'un modèle de gestion des identités avec une authentification mutuelle forte. C'est un atout majeur pour protéger les données personnelles et les attributs des utilisateurs. Un utilisateur de carte à puce peut authentifier des services réseau sans divulguer son identité, ce qui constitue un critère important pour protéger la vie privée.

Notre SDK intégré aux bibliothèques OpenSSL permettra aux développeurs d'implémenter des applications sécurisées et de tirer avantage de l'environnement de confiance qu'elle constitue. Il réduit le temps de développement en éliminant la nécessité de créer ses propres bibliothèques, rend les applications indépendantes des dispositifs de cartes à puce sous-jacentes et facilite le développement d'applications sensibles autour de cette technologie.

C'est un avantage certain puisqu'il leur permet également de se focaliser sur leurs implémentations au lieu de développer eux-mêmes ces fonctions cryptographiques, réservées aux spécialistes de la sécurité.

Ce kit pourrait devenir un standard intégré à OpenSSL à l'image des API Cryptoki pour les cartes à puce, pour fournir aux développeurs des fonctions primitives simples et des bibliothèques pour gérer ces microcontrôleurs.

Conclusion générale et perspectives

Les objectifs de cette thèse étaient de lever les principaux défis et d'apporter de nouvelles contributions pour ce nouveau contexte du "Cloud Computing et Virtualisation", de proposer une nouvelle architecture de gestion des identités et de contrôle des accès aux ressources, dans ces environnements virtuels, avec un haut niveau de sécurité, indispensable pour la protection de la vie privée. Ainsi un utilisateur pourrait s'identifier et s'authentifier une seule fois pour accéder à une multitude de services identifiés qui sont en adéquation avec ses préférences et ses besoins.

Contributions

Pour atteindre ces objectifs, nous nous sommes focalisés sur l'étude et l'analyse des solutions qui serviront à construire notre modèle de gestion des identités basé sur des composants physiques sécurisés. Ils offrent aux utilisateurs des avantages de gestion de leurs attributs et un environnement de confiance, pour garantir une authentification mutuelle forte et unique entre l'utilisateur et les ressources virtuelles distribuées.

Dans la première partie de ce mémoire, nous avons analysé et étudié de manière approfondie, des solutions et des standards de gestion des identités et leurs techniques d'intégration au sein d'une infrastructure. Nos réflexions nous ont conduits à identifier quelques verrous à franchir : la complexité de gestion des fonctions d'authentification et d'autorisation des ressources (physiques ou virtuelles), les différentes méthodes d'authentification, ainsi que les approches relatives au choix de la solution d'interopérabilité à adopter pour résoudre les problèmes de confiance engendrés par l'absence d'éléments sécurisés physiques associés aux ressources virtuelles.

Puis nous avons fait la synthèse et l'analyse en fonction de nos objectifs afin de ne retenir que les fonctions et les propriétés qui semblent nécessaires pour faire des propositions. Ces choix qui ont fondé nos contributions, s'appuient sur des standards ouverts, reconnus et éprouvés, en particulier le protocole d'authentification TLS. Ce protocole est à son tour embarqué dans des éléments de sécurité inviolables avec des contre-mesures pour protéger les informations d'identification des utilisateurs et mettre en œuvre des authentifications mutuelles fortes.

Pour répondre aux différents verrous et défis, nous nous sommes intéressés dans la seconde partie à la description de trois aspects (organisationnel, architectural et protocolaire) qui sont importants pour répondre aux besoins de la nouvelle infrastructure SecFuNet. Ensuite nous avons proposé plusieurs modèles : un modèle de gestion des identités centré sur l'utilisateur "User-Centric" et un modèle global de gestion des identités (Annexe D). Ces modèles sont basés sur des microcontrôleurs sécurisés, en adéquation avec notre proposition organisationnelle et entièrement adaptée aux environnements virtuels, qui constituaient notre cible première.

Ce modèle "User-Centric" basé sur des éléments sécurisés met le pouvoir entre les mains des utilisateurs au lieu qu'il soit en possession totale ou partielle des organismes de gestion des identités. Ils leur permettent de gérer et contrôler la diffusion de leurs données personnelles et de leurs attributs. L'utilisateur construit donc progressivement selon ses préférences ses propres relations de confiance contrairement à d'autres modèles qui exigent des liens préalablement établis. C'est un système totalement décentralisé, qui ne nécessite pas de maintenir de référentiel. Il permet de garantir l'unicité et la continuité d'une session dans un environnement hétérogène et décentralisé. Il peut aussi bien être utilisé pour l'accès à des ressources protégées dans un réseau interne que pour l'accès à des ressources situées dans d'autres domaines administratifs (i.e. dans le cadre d'une fédération d'identité).

Dans le modèle organisationnel nous avons découpé l'architecture actuelle en plusieurs niveaux de visibilité pour enfin pouvoir définir et délimiter les domaines administratifs de chaque entité et pour identifier les ressources virtuelles par un élément sécurisé. En effet chaque niveau est constitué de plusieurs fournisseurs de services associés à une IAM. Celle-ci permet de gérer d'une manière décentralisée, et en Top-Down, les identités de leurs propres ressources, avec un maximum de dynamicité, de flexibilité et de transparence.

Pour établir un lien logique entre les ressources et les éléments sécurisés, nous avons proposé des briques logicielles sous forme de Proxys. Ces briques constituent le socle architectural et jouent un rôle essentiel dans le fonctionnement de la pile protocolaire TLS embarquée dans les microcontrôleurs et facilitent les échanges et la médiation entre les postes clients, les ressources virtuelles, le serveur d'authentification et les éléments sécurisés.

Ces échanges de médiation reposent sur une couche protocolaire pour encapsuler des commandes ISO7816, afin de piloter ces éléments sécurisés distants, faciliter la gestion de leur cycle de vie et rendre les négociations transparentes d'une session d'authentification TLS. Enfin, dans la dernière partie, nous nous sommes efforcés de détailler l'implémentation des différentes briques logicielles proposées dans une perspective d'illustrer leur déploiement sous forme de scénarios de tests, en mettant en évidence l'authentification d'un composant par un autre (M2M) et pour les intégrer dans le démonstrateur pour valider la faisabilité de nos travaux de recherche avec la commission européenne.

Dans le dernier chapitre (Chapitre VII), nous avons proposé un kit pour intégrer la norme PC/SC dans OpenSSL et implémenter de nouvelles bibliothèques pour supporter les échanges avec les cartes à puce. L'objectif de ces fonctionnalités ajoutées et intégrées à OpenSSL, est de renforcer la sécurité des applications sensibles, de réduire leur coût de développement, et ainsi de banaliser ou de démystifier l'usage des cartes à puce, tout en s'adaptant aisément aux services émergents et de tirer parti des avantages fournis par les éléments sécurisés.

Ce kit pourrait devenir un standard intégré à OpenSSL à l'image des API Cryptoki pour les cartes à puce, pour fournir aux développeurs des fonctions primitives simples et des bibliothèques pour gérer ces microcontrôleurs.

- **Perspectives :**

Nous avons développé, dans le cadre de cette thèse, un démonstrateur qui nous a permis d'étudier la faisabilité de nos différentes contributions, nous avons développé des briques logicielles et avons démontré que leur intégration dans des environnements virtuels, permet de créer des liens logiques sécurisés et de faire de la médiation entre deux ressources pour réaliser une authentification mutuelle.

Cette architecture est loin d'avoir la prétention d'être la plus performante et de constituer un rempart absolu aux problèmes de sécurité informatique. Certes, la carte à puce reste quand même un élément inviolable et les données qu'elle contient sont difficilement voire impossibles à extraire ou à falsifier, en comparaison avec les autres composants qui constituent notre architecture (i.e. les Proxys).

En effet, la sécurité d'une architecture repose sur la protection de chaque élément qui la compose. Le Root-Kit est un risque majeur et permanent, pourrait se substituer à un Proxy par exemple, pour tromper une ressource afin de dérober ces attributs.

Il faudrait donc trouver de nouvelles parades pour protéger l'intégrité de ces briques logicielles qui constituent l'articulation logique et principale sur laquelle repose notre infrastructure de confiance. Il faudrait aussi tester la performance de ces contributions dans des applications réelles et des architectures transorganisationnelles, pour améliorer les temps de réponse pendant les authentifications simultanées.

Par ailleurs, le modèle d'infrastructure globale de gestion des identités, proposé dans ce contexte n'est pas dépourvu de cas d'utilisation ; il pourrait se décliner sous la forme d'un tiers de confiance pour offrir le service "Trust as a Service" dans le Cloud Computing pour mettre en avant les mérites d'une authentification mutuelle réalisée dans des éléments sécurisés. Il peut aussi être déployé dans des réseaux virtuels à la demande (type SDN : Software-Defined Networking), pour distribuer des clés de chiffrement au niveau liaison afin de sécuriser cette couche réseau. Ce dernier déploiement pourrait être une des applications majeures de ce modèle.

Références

Bibliographie :

[1] Boger, D. et Al, "User-Centric Identity Management Based on Secure Elements", ISCC´2014 : The 19th IEEE Conference Symposium on Computers and Communications, pages : 1–6, Madeira, Portugal. June 23-26, 2014.

[2] Priem, B. et Al, "Digital Privacy : Privacy and Identity Management for Europe", Heidelberg Dordrecht : Springer, p. 33-51, 2011.

[3] Elisa Bertino et Kenji Takahashi, "Identity Management : Concepts, Technologies, and Systems", Artech House Publishers, 196 p., September 7, 2011.

[4] Damiani, E., di Vimercati, S. D. C., and Samarati, P., "Managing multiple and dependable identities", In IEEE Internet Computing, Volume 7 Issue6, pages 29–37. IEEE, November 2003.

[5] David W. Chadwick, "Federated identity management - Foundations of Security Analysis and Design V", pages 96–120, 2009

[6] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell, "A modular correctness proof of IEEE 802.11i and TLS", The 12th ACM Conference on Computer and Communications Security (CCS'05), November 2005

[7] L.C. Paulson, "Inductive analysis of the Internet protocol TLS", ACM Transactions on Computer and System Security, Vol. 2, Number 3 : 332–351, 1999.

[8] Bhargav-Spantzel, A., Camenisch, J., Gross, T., and Sommer, D. "User centricity : a taxonomy and open issues", Journal of Computer Security, 15(5) : 493–527, 2007.

[9] Jøsang, A. and Pope, S., "User centric identity management", In AusCERT Asia Pacific Information Technology Security Conference 2005.

[10] Jøsang, A., Fabre, J., Hay, B., Dalziel, J., and Pope, S. "Trust requirements in identity management", In CRPIT'44 : Proceedings of the 2005 Australasian workshop on Grid computing and e-research, pages 99–108, Darlinghurst, Australia. Australian Computer Society, Inc. 2005.

[11] T. E. Maliki & J.-M. Seigneur. "A survey of user-centric identity management technologies", The International Conference on Emerging Security Information, Systems, and Technologies, SecureWare 2007, pages 12–17, 2007.

[12] Damiani, E., di Vimercati, S. D. C., and Samarati, P., "Managing multiple and dependable identities". IEEE Internet Computing archive, Volume 7 Issue 6, Pages 29-37, November 2003.

[13] Maler, E. and Reed, D. (2008). "The venn of identity : Options and issues in federated identity management", Security & Privacy, IEEE, Volume 6, Issue : 2, Pages : 16-23, April 2008.

[14] Recordon, D. and Reed, D. (2006). Openid 2.0 : "A platform for user-centric identity management", DIM '06 : Proceedings of the second ACM workshop on Digital identity management, pages 11–16, New York, NY, USA. ACM.

[15] M. Badra, & P. Urien, "Enhancing WLAN Security by Introducing EAP-TLS Smartcards", IADIS WWW/Internet 2004 Conference, Madrid, Spain, 6-9 October 2004.

[16] P., Urien, M., Badra, and M., Dandjinou, "EAP-TLS Smartcards, from Dream to Reality", the fourth IEEE workshop on Applications and Services in Wireless Networks, IEEE ASWN 2004, Boston, Massachusetts, USA.

[17] T.M. Jurgensen et al, "Smart Cards : The Developer's Toolkit", Prentice Hall PTR, ISBN0130937304, 2002

[18] Urien, P., Pujolle, G., "Security and Privacy for the next Wireless Generation", International Journal of Network Management, IJNM, Volume 18 Issue 2 (March/April 2008), WILEY.

[19] R. Couto, M. Campista, and L. H. M. K. Costa, "Xtc : A throughput control mechanism for xen-based virtualized software routers", in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pp. 1–6, 2011.

[20] Yinqian Zhang et Al, "Cross-VM side channels and their use to extract private keys", CCS'12 ACM : Computer and communications security, Pages 305-316, New York, NY, USA, October 16–18, 2012.

[21] Top Threats Working Group, "The Notorious Nine Cloud Computing Top Threats in 2013", Cloud Security Alliance, February 2013.
[22] Florencio, D. and Herley, C., "A large-scale study of web password habits", WWW ’07 : Proceedings of the 16th International Conference on World Wide Web, pp 657–666, New York, NY, USA. ACM, 2007.
[23] Zhiqun Chen, "Java CardTM Technology for Smartcards : Architecture and Programmer's (The Java Series) ", 2002, Addison-Wesley Pub Co 2002, ISBN 020170329.
[24] Urien, P., "Cloud of Secure Elements, Perspectives for Mobile and Cloud Applications Security", First IEEE Conference on Communications and Network Security, IEEE CNS 2013, DC USA, October 2013.
[25] Lee, H., Jeun, I., Chun, K. and Song, J. "A New Anti-Phishing Method in OpenID", In : SECURWARE '08. Second International Conference on, pp.243-247, 2008, IEEE.
[26] W. Diffie and M.E. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory, Volume 22, Numero 6, Pages : 664-654, 1976.
[27] Pascal Urien et al, "A New Convergent Identity System Based on EAP- TLS Smart Cards", Network and Information Systems Security (SAR-SSI), 18-21 May 2011. p 1 - 6.
[28] BARRETO, Luciano ; SIQUEIRA, F. ; FRAGA, J. S. ; FEITOSA, E. “An Intrusion Tolerant Identity Management Infrastructure for Cloud Computing Services”, International Conference on Web Services, 2013, Santa Clara Marriott. International Conference on Web Services, 2013. p. 155-163.
[29] Konstantinos Markantonakis, Keith Mayes, "Secure Smart Embedded Devices, Platforms and Applications", 2014, pp 407-427, 13 Sep 2013
[30] N.J. AlFardan, K.G. Paterson, "Lucky Thirteen : Breaking the TLS and DTLS Record Protocols", Security and Privacy, 2013 IEEE Symposium on, Pages 526-540, May 2013
[31] Serge Vaudenay, "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS, …", Advances in Cryptology EUROCRYPT'02, Amsterdam, Netherland, Lecture Notes in Computer Science No. 2332, pp. 534-545, Springer-Verlag, 2002.
[32] Nadhem J. AlFardan et AL, "On the Security of RC4 in TLS", SEC'13 Proceedings of the 22nd USENIX conference on Security - Pages 305-320, August 2013
[33] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol", The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, pp. 29-40, November 1996.
[34] John Kemp, Scott Cantor, Prateek Mishra, Rob Philpott, Eve Maler, "Authentication Context for the OASIS Security Assertion Markup Language V2.0", OASIS Standard, 15 March 2005.
[35] Chadwick, D. and Inman, G., "Attribute aggregation in federated identity", IEEE Computer, vol.42, pages 44–53, Issue No.05 May 2009.
[36] Ahn, G.-J. and Lam, J., "Managing privacy preferences for federated identity management", DIM ’05 : Proceedings of the 2005 workshop on Digital identity management, pages 28–36, New York, NY, USA. ACM., 2005.
[37] Simon Horman aka Horms, "SSL & TLS An Overview of A Secure Communications Protocol", the Security Mini-Conf at Linux.Conf.Au Canberra, ACT, Australia. April 2005.
[38] David Chisnall, "The Definitive Guide to the Xen Hypervisor : Understanding How Xen Approaches Device Drivers", Mar 21, 2008, Pearson Education, 2007,
[39] Thomas Ristenpart, Eran Tromer, Hovav Shacham, Stefan Savage, "Hey, You, Get Off of My Cloud : Exploring Information Leakage in Third-Party Compute Clouds", Proceeding CCS’09, the 16th ACM conference on Computer and communications security, Pages 199-212, Chicago, IL, USA, November 09 - 13, 2009.

Webographie :

[40] Burr, W. E., Dodson, D. F., and Polk, W. T. (2006). "Electronic authentication guideline". NIST Special Publication, 800:63. http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf.
[41] Joshua Hill, InfoGard Laboratories, "An Analysis of the RADIUS Authentication Protocol", November 2001, http://www.untruth.org/~josh/security/radius/radius-auth.html
[42] CERT-IST, "Faiblesses du protocole d'authentification Radius", Juillet 2005, http://www.cert-ist.com/public/print/fr/SO_detail?code=Faiblesses_Radius

[43] Mishra A., Arbaugh W., "An Initial Security Analysis of the IEEE 802.1X standard", http://www.cs.umd.edu/~waa/1x.pdf, February 2002.
[44] Chappell, D., “Introducing windows cardspace”, Msnd technical articles, Microsoft Corporation, 2006, http://msdn.microsoft.com/en-us/library/aa480189.aspx
[45] K. Cameron, "Integrating OpenID and Infocard". 2007. http://www.identityblog .com/?p=659.
[46] J.J. Garrett, "AJAX : A New Approach to Web Applications", February 2005, http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.
[47] PC/SC Workgroup, “PC/SC Workgroup Specifications Overview, Version 2.01.14”, June 2013, http://www.pcscworkgroup.com/specifications/overview.php.
[48] OpenSC “OpenSC-Tools and Libraries for Smartcard”, https://github.com/OpenSC/OpenSC/wiki
[49] Trusted Computing Group, “TPM Main Part 1 Design Principles. Specification Version 1.2”, Revision 103, July-2007.
[50] MAKANA, June 9, 2011, "Des inquiétudes persistent sur la sécurité du Cloud Computing", 2011, http://www.itnation.lu/news/des-inquietudes-persistent-sur-la-securite-du-cloud-computing/2629/
[51] The Heartbleed Bug, Avril 2014, http://heartbleed.com/
[52] "What Is IAS ?" Updated : March 28-2003, https://technet.microsoft.com/en-us/library/cc737273%28v=ws.10%29.aspx
[53] Marc Goodner, Anthony Nadalin, "Web Services Federation Language (WS-Federation) Version 1.2", OASIS Standard, 22 May 2009, http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html
[54] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", W3C, June 2007, http://www.w3.org/TR/2007/REC-wsdl20-20070626/
[55] Asir S Vedamuthu, et AL., "Web Services Policy 1.5 - Framework", W3C, September 2007, http://www.w3.org/TR/2007/REC-ws-policy-20070904/
[56] Doug Davis, Ashok Malhotra, Katy Warr, Wu Chou, "Web Services Transfer (WS-Transfer)", W3C Working Draft, August 2010, http://www.w3.org/TR/2010/WD-ws-transfer-20100805
[57] Chappell, David, “Introducing windows cardspace. Msnd technical articles, Microsoft Corporation”, April 2006, http://msdn.microsoft.com/en-us/library/aa480189.aspx.
[58] Doug Davis, Ashok Malhotra, Katy Warr, Wu Chou,"Web Services Metadata Exchange (WS-MetadataExchange)", W3C Working Draft, March 2009, http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317.
[59] Eclipse Foundation 2010. "Higgins open source identity framework", http://www.eclipse.org/higgins/.
[60] Tim Moses, "eXtensible Access Control Markup Language (XACMLV2.0), OASIS Standard, Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
[61] John Kemp, et AL, "Authentication Context for OASIS SAMLV2.0", OASIS Standard, 15 March 2005, http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf
[62] Alexsandra Nenadic, Ning Zhang and Mike Jones, "A Gap Analysis of Current LoA Definitions vs. LoA Requirements in e-Science and Grid Context", Open Grid Forum LoA-RG Deliverable : LoA Gap Analysis v.3, April-2008, https://redmine.ogf.org/dmsf_files/46?download=61.
[63] Cyrille Chausson, "Une attaque DDoS suspectée contre Amazon", déc. 2009, http://www.lemagit.fr/actualites/2240196192/Une-attaque-DDoS-suspectee-contre-Amazon/.
[64] "Trojan Bohu", March 2013, http://pro.01net.com/editorial/527253/un-virus-attaque-les-antivirus-en-mode-cloud/.
[65] Gordon Fowler, "Affaire PlayStation Network", Mai 2011, http://www.developpez.com/actu/32010/Affaire-PlayStation-Network-les-attaques-auraient-utilise-le-Cloud-d-Amazon-Sony-commence-a-reactiver-ses-services/.
[66] OpenID (2007). Openid authentication 2.0. OPENID. http://openid.net/specs/openid-authentication-2_0.html.
[67] OpenIMS, "Open IP Multimedia Subsystem Core", 2006, http://www.openimscore.org/
[68] OpenSSL, "The Open Source toolkit for SSL/TLS.", http://www.openssl.org.
[69] T. Scavo & S. Cantor,"Shibboleth Architecture", 2005, http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf.
[70] Jeff Costlow, "BREACH attack", August 2013,https://devcentral.f5.com/articles/breach-attack.

[71] Marc Goodner, Anthony Nadalin, "Web Services Federation Language (WS-Federation) Version 1.2", OASIS Standard, 22 May 2009, http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html.
[72] Russ Cutler, "Liberty Identity Assurance Framework V1.1", Liberty Alliance Project, http://www.projectliberty.org/liberty/content/download/3736/24651/file/liberty-identity-assurance-framework-v1.0.pdf.
[73] Tim Hudson, RSA Laboratories, "PKCS #11 v2.20 : Cryptographic Token Interface Standard", 10-Sep-2009, http://www.cryptsoft.com/pkcs11doc/v220/.
[74] Aarts, R. and Madsen, P. (2006). “Liberty ID-WSF Interaction Service Specification v.2.”, Liberty Alliance Project, http://www.projectliberty.org/liberty/content/download/.
[75] CERT-IST, "Virtualisation des systèmes d'exploitation et sécurité", Août 2007, http://www.cert-ist.com/fra/ressources/Publications_ArticlesBulletins/Autres/virtualisation_0807/.
[76] Thierry DOSTES & Maurice LIBES, "Virtualisation selon Xen", Journées Thématiques SIARS, CESAR, 17-18 Septembre 2009, http://cesar.resinfo.org/IMG/pdf/virtualisation_diapos_xen_v2_1_.pdf.
[77] http://sites.amd.com/fr/business/it-solutions/virtualization/Pages/amd-v.aspx
[78] http://www.intel.com/technology/virtualization/technology.htm]
[79] O.S. Community, "JOIDS (Java OpenID Server) : A multi-domain, multi-user OpenID Provider", https://code.google.com/p/openid-server/, 2014.

RFC, Drafts, Normes & standards

[80] RFC 3748, June 2004, Aboba B., Blunk L., Vollbrecht J., Carlson J., Levkowetz H., “Extensible Authentication Protocol (EAP)”.
[81] RFC 1994, August 1996, Simpson W., PPP Challenge Handshake Authentication Protocol (CHAP).
[82] RFC 2716, October 1999, Aboba B., Simon D., PPP EAP TLS Authentication Protocol.
[83] RFC 2246, January 1999, Dierks T., Allen C., "The TLS Protocol Version 1.0".
[84] RFC 5216, March 2008, "The EAP-TLS Authentication Protocol".
[85] RFC 3748, June 2004, "Extensible Authentication Protocol, (EAP)".
[86] RFC 5246, August 2008, T.Dierks, E. Rescorla, "The TLS Protocol Version 1.2".
[87] IETF DRAFT, October 2002, CALHOUN P.R., JOHANSSON T. et PERKINS C.E., Diameter Mobile IPv4 Application-draft-ietf-aaa-diameter-mobileip-13.
[88] IETF DRAFT, March 2002, Pat R. Calhoun, Stephen Farrell, William Bulley, "Diameter CMS Security Application", Internet-Draft, http://www.ietf.org/archive/id/draft-ietf-aaa-diameter-cms-sec-04.txt.
[89] IETF DRAFT, January 2013, P. Urien, G. Pujolle, "EAP-Support in Smartcard", https://tools.ietf.org/html/draft-urien-eap-smartcard-22.
[90] IETF DRAFT, July 2014, P.Urien, "Remote APDU Call Secure", http://tools.ietf.org/html/draft-urien-core-racs-03.txt.
[91] DRAFT, August 2010, S. Turner, Tim Polk, "Prohibiting SSL Version 2.0", http://tools.ietf.org/html/draft-ietf-tls-ssl2-must-not-04.
[92] IETF DRAFT, June 2010, S. Turner, Tim Polk, "Prohibiting SSL Version 3.0 and Earlier", http://tools.ietf.org/html/draft-turner-ssl-must-not-01.
[93] IETF DRAFT, January, 1997, D. Carrel, Lol Grant Cisco Systems, "The TACACS+ Protocol Version 1.78", https://tools.ietf.org/html/draft-grant-tacacs-02.
[94] Global Platform Standards, " Card Specification Version 2.1.1", March 2003.
[95] IEEE 802.1X, Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks : Port-Based Network Access Control", IEEE Standard 802.1X, September 2001.
[96] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts".
[97] EuroSmart, "Smartcard IC Platform Protection Profile, Version 1.0", July 2001, https://www.commoncriteriaportal.org/files/ppfiles/ssvgpp01.pdf
[98] Federal Information Processing Standards Publication, "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES", Supercedes FIPS PUB 140-1, 1994 January, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[99] "FIPS 140-1 and FIPS 140-2 Vendor List validation of cryptographic module", http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401vend.htm

[100] CCID (Chip/Smart Card Interface Devices) "Universal Serial Bus Device Class Specification for USB", Revision 1.00, March 20, 2001.

Liste des publications dans des Conférences :

1- ACSAC-2014 : Smartcard Support Embedded within OpenSSL Tool Kits, Hassane Aissaoui-Mehrez, Pascal Urien & Guy Pujolle; ACSAC-2014, Annual Computer Security Applications Conference, 8–12, Décembre 2014 New Orleans, Luisiane, USA.

2- MILCOM.2014 : Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements, Hassane Aissaoui-Mehrez, Pascal Urien & Guy Pujolle; IEEE MILCOM-2014, Military Communications Conference, 6-8 Octobre 2014, pages: 282-287, 10.1109/MILCOM.2014.51, Baltimore, Maryland, USA.

3- SECRYPT-2014 : Framework Implementation Based On Grid of Smart Cards To Authenticate Virtual Machine, Hassane Aissaoui-Mehrez, Pascal Urien & Guy Pujolle; SECRYPT-2014, The 11th International Conference on Security and Cryptography, pages: 285-290, Vienne, Autriche, 28-30 Aout 2014.

4- IEEE CITS-2014 : Global Identity Management of Virtual Machines Based on Remote Secure Elements; Hassane Aissaoui-Mehrez, Pascal Urien & Guy Pujolle; IEEE CITS 2014: International Conference on Computer Information and Telecommunication Systems, Pages 1-4; DOI: 10.1109/CITS.2014.6878954 ; 7-9 Juillet 2014, Jeju, Corée du Sud.

5- IEEE ISCC-2014 : User-Centric Identity Management Based on Secure Elements, Boger, D. ; Barreto, L. ; Fraga, J. ; Urien, P. ; Aissaoui-Mehrez, H. ; Santos, A. ; Pujolle, G.; ISCC´2014: The 19th IEEE Conference Symposium on Computers and Communications, pages: 1–6, Year: 2014, DOI: 10.1109/ ISCC.2014.6912541, Madeira, Portugal. June 23-26, 2014.

6- Journal IJRTET : Security for Future Networks: a Prospective Study of AAIs, Aissaoui-Mehrez, H.; Urien, P. ; Pujolle, G.; Fraga, J. ; Boger, D. ; Journal Int. J. on Recent Trends in Engineering and Technology, 10 , no. 1 (2014): 10., January 2014, The Third International Conference on Advances in Communication and Information Technology-CIT-2013.

7- WINSYS-2013 : Low Latency of Re-Authentication During Handover: Re-Authentication Using a Signed Token in Heterogeneous Wireless Access Networks, Hassane Aissaoui-Mehrez, Pascal Urien & Guy Pujolle; WINSYS-2013, INSTICC International Conference on Wireless Information Networks and Systems, SIGMAP, pages: 248-254. SciTePress, 2013; Reykjavík, Islande, 29-31-07-2013.

Annexe A – Quelques attaques liées au vol d'identité

Les techniques liées au vol d'identité numérique sont nombreuses. Généralement, elles sont motivées par le gain d'argent en accédant au compte bancaire d'une victime. Nous allons, dans cette annexe détailler les principales techniques qui permettent de perpétrer une usurpation d'identité.

A.1 Les techniques d'attaques :

Le Phishing : hameçonnage ou filoutage, est une attaque reposant sur l'ingénierie sociale, utilisée par des hackeurs pour obtenir des informations privées et usurper l'identité d'une victime. Elle consiste à se faire passer pour un tiers de confiance (le plus souvent : grandes banques, sites marchands en ligne, organismes financiers, etc.).

Le principe de cette technique repose sur l'envoi d'un courriel frauduleux contenant un lien, vers un site Web, lequel correspond à une adresse URL d'un site contrefait. Quand l'utilisateur se rend sur ce site, il est invité à renseigner un formulaire avec des informations personnelles. Ensuite, ses identifiants (mot de passe, comptes bancaires, numéros de cartes de crédit, etc.) sont récupérés et détournés pour extirper des fonds.

Le Pharming : technique consistant à détourner une résolution de nom (DNS) d'un site Internet, ainsi un utilisateur est redirigé vers un autre site, en croyant qu'il se connecte sur un site légitime. Cet exploit peut s'opérer soit localement sur le poste de l'utilisateur en installant un Trojan redirecteur soit sur un serveur DNS. Cette dernière attaque se concrétise par un empoisonnement du cache DNS d'un fournisseur.

Le Trojan : Cheval de Troie ou encore Troyen, est une technique qui consiste généralement à installer un logiciel lors de la consultation d'une page Web malicieuse. Ce Trojan peut aussi télécharger d'autres composants (virus, keylogger, logiciel espion, etc.), lesquels seront exécutés sous forme de modules dans un navigateur Web ou sous forme d'un Trojan résident dans le système d'exploitation avec des fonctionnalités "Root-Kit".

C'est un programme qui ne se reproduit pas mais sert de base pour permettre des intrusions sur une machine. L'objectif est généralement d'ouvrir un port TCP dérobé ("BackDoor") sur le système hôte, permettant par la suite à un hackeur de se connecter par le

réseau pour collecter des informations personnelles, voire même le contrôler à distance pour perpétrer des attaques sur d'autres machines (i.e. **Déni de Service).**

Denial-of-service (DoS) : le déni de service, est une attaque qui vise à submerger les services réseau ou système d'une ressource avec du trafic inutile. Par contre l'attaque distribuée (DDoS) implique plusieurs machines zombies contrôlées par Root-Kit pour submerger une cible donnée.

Le DoS cible généralement un défaut de conception ou d'implémentation d'un protocole de communication. Il peut constituer un enjeu énorme pour une entreprise, et lui coûter très cher puisqu'il rend difficile ses transactions.

Brute Force Attack : l'attaque force brute est un procédé qui vise à tester de façon exhaustive des combinaisons possibles d'un ensemble de clés ou de mots de passe pour trouver au moins une combinaison valide.

Généralement, cette attaque est considérée comme simple à mettre en œuvre et permet de cracker un secret, indépendamment du protocole d'authentification utilisé. Le temps nécessaire pour casser un mot de passe dépendant de longueur de celui-ci (i.e. c'est une fonction exponentielle), mais avec l'optimisation des algorithmes en utilisant des méthodes de cryptanalyse et la puissance de calcul croissant des nouveaux CPU ou GPU, ce temps pourrait être très réduit.

Man in the Middle (MITM) : l'homme du milieu, est une attaque mettant en place une médiation protocolaire pour atteindre son objectif. Les communications de la victime sont interceptées par l'intrus, lequel redirige les flux vers les destinataires légitimes.

En effet, l'intrus se positionne entre deux systèmes communicants, se fait passer auprès de chacune pour l'entité légitime. Il observe et intercepte la totalité des messages supposés être protégés. Ensuite, il les retransmet vers une des parties. La réussite de cette attaque repose sur la faiblesse du protocole d'authentification implémenté. Par conséquent, tous les protocoles sans authentification mutuelle sont une cible de cette attaque MITM, en particulier les protocoles avec un Login/mot de passe.

Replay Attack : le rejeu est une forme d'attaque considérée comme une technique de type MITM consistant pour une tierce partie à intercepter et à enregistrer des messages d'authentification pour les rejouer sans les modifier auprès d'un serveur destinataire pour obtenir des autorisations d'accès frauduleux à des services protégés.

Les mesures anti-rejeu consistent en général à échanger des nombres aléatoires uniques (Nonces) ou à mettre en œuvre un horodatage (Timestamp) pour limiter la durée d'utilisation d'une authentification.

A.2 Quelques vulnérabilités théoriques du protocole TLS :

Depuis quelque temps, le protocole TLS a fait l'objet de nouvelles attaques théoriques et pratiques. Quelques vulnérabilités intéressantes ont été dévoilées et médiatisées ces dernières années. Parmi les plus connus, nous pouvons citer :

✓ Vulnérabilité Lucky13 : c'est une vulnérabilité annoncée par deux chercheurs britanniques (6 février 2013) [30] contre le protocole de communication sécurisé SSL/TLS et DTLS (Datagram TLS). DTLS est une variante de TLS. L'attaque est de type MITM, qui permet à un intrus de récupérer du texte chiffré à partir d'une connexion DTLS, lorsque le chiffrement en mode CBC (Cipher Block Chaining) est utilisé [31].

L'objectif de l'attaque est de déchiffrer ce texte chiffré recueilli par écoute d'une session DTLS. Pour ce faire, ils utilisent une attaque de type Padding Oracle Attack qui effectue plusieurs milliers de requêtes sur le serveur pour récupérer un maximum de Padding (rembourrage). Le contenu est chiffré mais le serveur aura quand même répondu. Ensuite l'attaquant cherche à reconstituer et tirer des conclusions sur les messages chiffrés obtenus du serveur [31].

Ce type d'attaque est difficile à mettre en œuvre, car elle nécessite des conditions particulières ; elle n'est faisable que si l'attaquant est proche du serveur et du client attaqués.

✓ Single-byte bias attack contre RC4 : les faiblesses de RC4 sont bien connues depuis que le protocole WEP a été craqué. Malheureusement, il reste utilisé pour assurer la sécurité dans les GSM et le Wi-Fi avec TLS/SSL. Environ 50 % du trafic TLS est actuellement protégé en utilisant l'algorithme RC4. Une nouvelle attaque a été démontrée contre RC4, par des chercheurs britanniques (12 Mars 2013) [32], mais elle nécessite la récupération au préalable d'une très grande quantité de données, et ces derniers affirment qu'elle n'est pas très pratique à mettre en œuvre. En effet, un attaquant ne peut récupérer qu'une quantité limitée de texte en clair à partir d'une connexion TLS lorsque le cryptage RC4 est utilisé. Les attaques proviennent des défauts statistiques dans le flux de clés générées par l'algorithme RC4. Celui-ci révèle des parties des textes chiffrés lorsque l'attaquant cherche à déchiffrer une donnée qui

est répétée plusieurs fois et prévisible (mot de passe, cookie,, etc.). Cette attaque est réalisable à condition de collecter suffisamment de trafic.

✓ Rollback (Rétrogradation des algorithmes de chiffrement) : l'attaquant "MITM" tente de rétrograder le choix des algorithmes d'échanges de clés de façon à ce que les deux entités négocient un niveau de chiffrement le plus faible et n'utilisent pas les mêmes (RSA et DH par exemple). Pendant le Handshake, les messages ClientHello et ServerHello sont envoyés en clair. Un attaquant pourrait forger un message ClientHello avec une CipherSuite moins robuste, et pourrait ainsi déchiffrer le message. Cette attaque peut être réalisée au moment d'une reprise de session. Dans ce cas la version SSLv3 passe vers la version SSLv2 [92] [33] [91].

Annexe B – Étude des Infrastructures d'Authentification et d'Autorisation

B.1) SAML: Security Assertion Markup Language

SAML [34] est un standard définissant un protocole pour échanger en toute sécurité des informations d'identité (authentification, autorisation et attributs) entre applications, indépendamment de la technologie utilisée par des applications (PKI, SSO, LDAP, KERBEROS, etc.). La version 2.0 de SAML a été normalisée en mai 2005 par l'OASIS (Organization for the Advancement of Structured Information Standards) dans le Service de Sécurité du Comité Technique (SSTC). Dans ses premières versions (1.0 et 1.1), SAML a été conçu uniquement pour faciliter l'échange d'informations d'identité et des autorisations utilisateurs pour permettre une authentification unique (SSO) sur le Web. En revanche, la version actuelle (2.0) a été étendue pour couvrir des concepts et des mécanismes issus des projets Liberty Alliance Identity Federation Framework (ID-FF) V1.2 et Shibboleth V1.2. Ces projets ont des objectifs plus larges, tels que la création de fédérations, le partage d'informations de sécurité et la gestion des identités.

Il est important de noter que SAML v2 est le résultat de la convergence de l'ID-FF V1.2, de SAML v1.X d'OASIS. Il a incorporé quelques fonctionnalités de l'architecture de Shibboleth 1.2, et à ce titre, il n'y a pas de compatibilité ascendante entre SAML 1.X et SAML 2.0 même s'ils traitent les mêmes cas d'usage.

B.1.1) Échange des informations d'identité dans SAML

Pour le format des assertions, SAML utilise le standard XML pour représenter les informations de sécurité. Les spécifications de SAML définissent cinq composants [13] ; elles sont illustrées Figure 68 :

- Le composant "*Roles*" : c'est le rôle que chaque entité peut jouer dans l'infrastructure SAML (IdP, SP, etc.). Ce composant définit également les métadonnées qui décrivent ces entités.
- Le composant "*Profiles*" : il décrit les protocoles et les assertions des transferts de données spécifiques pour fournir la gestion des identités et l'authentification unique.

- Le composant "*Assertions*" : il spécifie le format pour représenter des informations de sécurité sur un sujet qui peut être une personne, une organisation, un ordinateur, etc. Il est essentiel pour la gestion des identités et du contexte de sécurité.

- Le composant "*Protocols*" : il est utilisé pour demander et transférer les assertions entre les entités. Ce composant peut être transféré avec les différents mécanismes du composant "*Transport*".

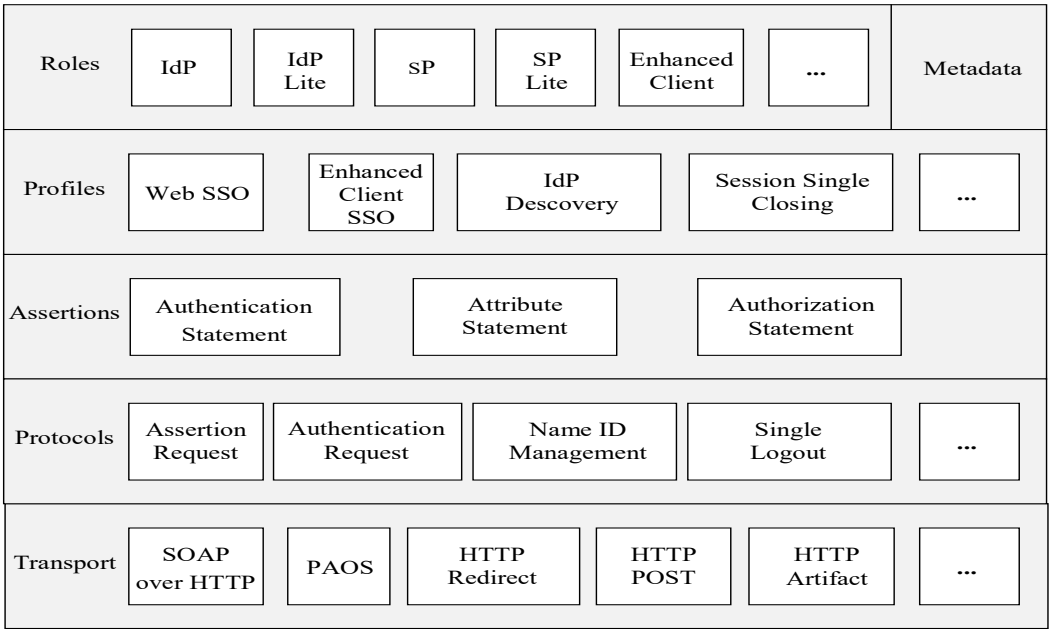

Figure 68 : Architecture SAML (Source [13])

B.1.2) Vie privée et fédération d'identité dans SAML

SAML offre des fonctionnalités pour mettre en œuvre différentes approches de gestion des identités. En général, SAML permet à toutes les infrastructures externes à une organisation ou à tous les mécanismes d'associer et de contrôler les différentes identités d'un objet. Dans les premières versions, peu de mécanismes étaient disponibles pour préserver la vie privée et la confidentialité quand il s'agissait de partager les informations des utilisateurs.

Par exemple, deux applications pouvaient utiliser une base de données synchronisée pour suivre la trace des utilisateurs avec le même nom d'utilisateur, ou la même adresse e-mail. SAML 2.0 prend en charge l'utilisation de pseudonymes, qui sont des identifiants dynamiques et non liés à des attributs d'identité de l'objet.

Ces pseudonymes sont gérés par des protocoles : "*Name Identifier Management Protocol*", *"Name Identifier Mapping Protocol ", "Transport Protocols" et "Associated profiles"*.

Les Pseudonymes servent d'identificateurs communs entre SP et IdP et peuvent être utilisés de deux manières (*Pseudonymes Persistant ou Pseudonymes Transitoire*). Ils sont créés par l'IdP.

- ***Le Pseudonyme Persistant*** est associé de manière permanente à une identité d'un objet. Dès réception d'une assertion d'authentification contenant un pseudonyme, le SP crée un cache en local en utilisant ce pseudonyme ainsi que d'autres informations d'identité correspondant à cet objet. Lors d'accès ultérieurs du même objet, le SP reçoit son pseudonyme qui est déjà enregistré dans le cache et peut prendre des décisions d'autorisation en se basant sur les données locales fournies par l'IdP. Le SP peut même demander à un IdP ou une autorité d'attribut, des attributs supplémentaires associés à ce pseudonyme. Ce système combiné avec les politiques de protection de la vie privée et de confidentialité peut garantir la protection de l'identité. Toutefois, l'accès aux différents SPs peut encore être tracé si ces derniers agissent en connivence.
- ***Le Pseudonyme Transitoire*** est associé aussi à l'identité d'un objet pendant la durée d'une session sécurisée. Ainsi, le SP peut encore décider d'autoriser un objet en fonction des attributs émis par l'IdP pendant cette session, mais ne peut pas demander plus d'information sur cet objet persistant dans son cache, au-delà de la durée de sa session. Par ailleurs, les SPs ne peuvent pas corréler différentes sessions d'un même objet et ainsi un certain niveau d'anonymat est assuré.

B1.3) Avantages et inconvénients de SAML

Les avantages du langage SAML tiennent à la maturité de son développement en relation étroite avec Shibboleth et Liberty Alliance, en particulier le développement de la fédération d'identité qui a donné naissance à : SAML, version 2.0 en 2005. Paradoxalement, cette maturité qui constitue sa force est devenue un gros handicap, car elle le rend assez lourd et complexe pour une implémentation massive et simple dans des entités plus petites, lesquelles préféreront le déploiement d'une technologie plus facile et moins complexe.

Par ailleurs, SAML est compatible avec les spécifications des Web Services, lesquelles profitent de nombreuses fonctionnalités de SAML visant à assurer la fédération d'identité, en

particulier du point de vue de la protection de la vie privée. La spécification WS-Federation [71] (§3.4.2.2), par exemple, s'appuie sur les mécanismes de gestion des pseudonymes de SAML qui permettent de rendre impossible pour un IdP de connaître et de suivre les SP consultés par un utilisateur.

B.2.) Web Services et ces extensions (WS-*)

Les spécifications du Web Services (WS) [53] se composent de plusieurs extensions. La spécification WS-Security (Figure 69) définit des mécanismes pour assurer l'intégrité et la confidentialité des messages SOAP, (*Simple Object Access Protocol*). Elle permet d'assurer la sécurité des API SOAP, car elle intègre les deux extensions de XML : XML Signature et XML Encryption.

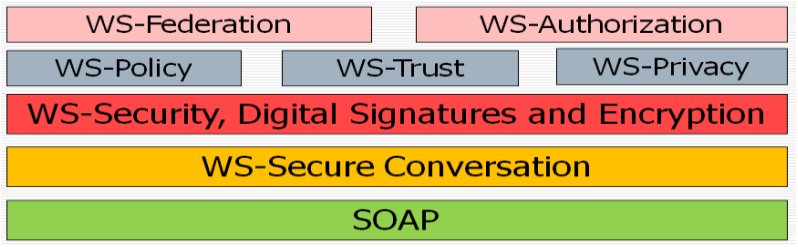

Figure 69 : Les composants de l'architecture du WS

B.2.1) WS-Trust

La spécification WS-Trust étend WS-Security avec la définition d'un protocole d'échange et de dissémination de l'information d'identification de sécurité entre les différents domaines. WS-Trust fournit également des moyens permettant de vérifier la véracité d'une relation de confiance ou des informations d'identification. Le modèle de confiance défini dans la spécification WS-Trust est basé sur un service de jeton de sécurité "*Security Token Service*" (STS). Le STS est un service Web qui implémente une interface de la norme WSDL [54], où les opérations d'émission, de renouvellement, de validation et de révocation des informations d'identification sont définies. Typiquement, si un client souhaite accéder à une application sécurisée, celle-ci le redirigera vers un STS de confiance qui va l'authentifier et lui générer un jeton de sécurité. Ensuite, le client sera redirigé de nouveau vers l'application sécurisée. Il présentera le jeton à l'application. Celle-ci vérifiera son origine (i.e. STS de confiance) et donnera ou non l'autorisation au client. Ainsi, le STS servira de médiateur de confiance entre l'application et le client.

En général, le modèle de confiance de WS-Trust spécifie qu'un fournisseur de services peut exiger que le jeton contienne la preuve d'une relation de confiance, avant qu'il soit traité. Cette exigence de sécurité imposée par le fournisseur de services se traduit par le support d'un ensemble de jetons de sécurité et d'un ensemble de STS de confiance. Si un message contient suffisamment de jetons de sécurité émis par les STSs de confiance, le SP continue le traitement. Cette exigence doit être décrite sous forme de politique spécifiée dans le standard *WS-Policy* [55]. Ces politiques devraient également être attachées aux descriptions WSDL d'un service. La Figure 70, illustre une médiation de confiance basée sur un tiers de confiance (TTP : Trusted Third Party) qui implémente un STS :

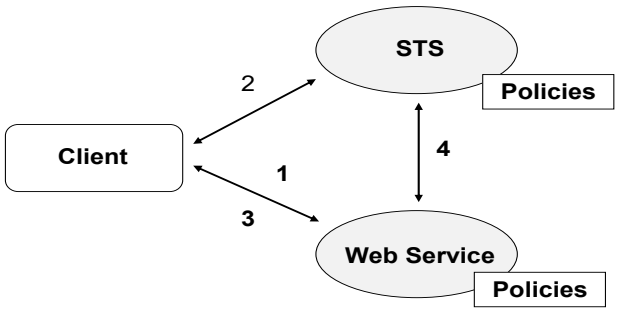

Figure 70 : Le modèle de confiance du WS-Trust

Lorsque le client veut accéder au WS, il obtient tout d'abord sa politique (étape 1). Grâce à cette politique, le client découvre l'ensemble des exigences du WS et son STS de confiance. Si le client ne dispose pas de suffisamment de jetons pour satisfaire les exigences du WS, il peut demander des jetons à l'un des STS énumérés par ce WS (étape 2).

Toutefois, le STS peut également exiger une politique spécifiant un ensemble d'exigences différentes pour qu'il puisse émettre des jetons. Si le client dispose de suffisamment de droits pour accéder au STS approuvé par le WS, il peut lui demander de délivrer des jetons requis pour ce WS. Le STS délivre les jetons. Le client les présente en même temps que la requête envoyée au WS (étape 3). Ensuite, le WS vérifie la validité des jetons, il peut éventuellement la revérifier (étape 4) avec le STS.

Les messages échangés par le STS sont assez génériques et permettent des extensions et des compositions futures. Chaque opération (délivrance, validation, renouvellement, révocation, etc.) spécifie quels sont les éléments XML qui doivent être inclus dans le message échangé. Le message général de la requête comporte l'élément *RequestSecurityToken* (RST), et la réponse est constituée de l'élément *RequestSecurityTokenResponse* (RSTR). Le RST

contient les paramètres généraux (*type de requête, exigences de service, type d'informations d'identification, etc.)*. Selon le type de requête, il peut contenir des éléments plus spécifiques, tels que la preuve de la possession d'une clé privée, la date d'expiration des jetons, etc. par contre, le RSTR contient essentiellement le jeton demandé ainsi que les paramètres d'informations d'identification (*type, date d'expiration, le contexte d'utilisation, etc.*).

B.2.2) WS-Federation

La spécification WS-Federation définit des mécanismes de construction de fédérations basés sur les normes WS-*, notamment *WS-Security, WS-Trust et WS-Policy*. Ces normes définissent déjà la base pour la gestion des identités fédérées, mais WS-Federation propose des extensions qui définissent la façon de combiner ces modèles afin de fournir des fonctionnalités plus riches aux domaines de sécurité (ou aux domaines administratifs) au sein et à travers des fédérations. L'objectif principal de WS-Federation est d'assurer la gestion fédérée d'identités pour les services Web. Par conséquent, la spécification supporte les clients SOAP et la communication directe entre les services Web. WS-Federation inclut également un profil de demandeurs passifs où les navigateurs Web HTTP participent en tant que clients dans le workflow message.

WS-Federation permet la gestion fédérée des identités et fournit un ensemble de fonctionnalités similaires à SAML2.0 : SSO, Single Logout (SLO), des échanges d'attributs basés sur des politiques de protection de la vie privée et de confidentialité, des pseudonymes permanents et transitoires et des métadonnées. WS-Federation implémente les services de gestion des identités basés sur le STS spécifié dans WS-Trust. Ainsi, le STS joue le rôle de fournisseur d'identités et peut émettre des jetons contenant des informations d'identité pour un objet dans son domaine de sécurité. Ces jetons d'identité peuvent être utilisés dans d'autres domaines administratifs qui ont des relations de confiance avec le domaine d'origine.

B.2.3) Gestion des services Attributs et Pseudonymes

En plus de la fonction de fournisseur d'identité, WS-Federation définit également le service d'attribut (*Attribute Service*) et le service pseudonymes (*Pseudonyms Service*).

- **Le Service d'attribut (Attribute Service) :**

Il est responsable de la délivrance des attributs d'identité des clients. La spécification ne nécessite aucune interface spécifique. Toutefois, elle recommande l'utilisation de STS afin

d'exploiter le même modèle de confiance WS-Trust et le protocole de communication. En plus, l'utilisation de STS facilite l'implémentation de deux services à la fois dans la même entité : l'IdP et le Service d'attributs.

Le Service d'attribut doit assurer la protection de la vie privée et les informations des clients. Par conséquent, WS-Federation spécifie que les attributs doivent être associés à des politiques d'accès différentes, en fonction des préférences d'un client.

- **Le Service pseudonyme (Pseudonym Service) :**

Il a pour fonction d'associer un pseudonyme à une identité utilisateur. Un pseudonyme est un type spécial d'attribut utilisé pour identifier un utilisateur sans révéler ses informations d'identité. D'une manière similaire à SAML, un pseudonyme peut, dans WS-Federation, avoir différents niveaux de personnalisation pour préserver la vie privée. Par exemple, un client peut avoir des pseudonymes qui ne durent que pendant une seule session d'authentification. Par ailleurs, pour augmenter la protection de sa vie privée, il empêche les services de lui associer toute information persistante.

Le *Service pseudonyme* utilise une interface, spécifiée dans *WS-Transfer* [56], qui est différente à celle utilisée par l'IdP et le *Service d'attribut*. Il ne repose pas sur les STS. Cette spécification définit des méthodes pour créer, supprimer, mettre à jour et accéder à des pseudonymes existants. Les services (attribut, Pseudonym et IdP) peuvent fonctionner ensemble pour fiabiliser des fonctionnalités complexes qui assurent la protection de la vie privée et la confidentialité des individus. Par exemple, le *Service d'attribut* peut répondre aux demandes d'attributs associés à un pseudonyme. Pour cela, il doit invoquer le service pseudonyme et obtenir l'identité du client associée à ce pseudonyme. En outre, ces services peuvent être mis en œuvre dans le même système.

WS-Federation définit également un format XML pour spécifier des métadonnées de fédération. Celles-ci contiennent la description des ressources pour faciliter la découverte de services et la communication entre les membres d'une fédération. Cette description spécifie les mécanismes de découverte de services fédérés et des métadonnées (WSDL, politiques, etc.) associées aux membres de la fédération. Le format XML proposé est basé sur les métadonnées de la spécification SAML.

B.3) OpenID

OpenID est un Framework de gestion des identités léger, évolutif et extensible maintenu par OpenID Foundation, une organisation à but non lucratif qui promeut son développement technologique. Il a été développé en 2005 par Brad Fitzpatrick, architecte en chef à SixApart, pour le site Live Journal, afin de faciliter l'authentification des commentateurs de blogs et d'éviter le spam dans les commentaires [14]. Il est basé sur des standards Web tels que HTTP et URI et offre une gestion des identités centrée sur l'utilisateur (User-Centric) et l'authentification unique (SSO). L'idée de base est de permettre à un utilisateur d'accéder à un site Web s'il est en mesure de démontrer qu'il contrôle un identifiant OpenID. Cet identifiant est généralement une URL (Uniform Resource Locator) ou, dans certains cas, une XRI (Extensible Resource Identifier).

De nos jours, de nombreuses organisations et fournisseurs utilisent l'authentification OpenID pour permettre l'accès à leurs sites Web. Environ un million de sites Web étaient compatibles avec OpenID en 2012 et plusieurs grandes entreprises l'ont adopté comme système de gestion des identités : Google, LiveJournal, Facebook, Yahoo, Microsoft, AOL, Universal Music Group, MySpace, France Télécoms (Orange), Novell, Sun racheté par Oracle, Telecom Italia, IBM, PayPal, VeriSign, Ustream…

Cette infrastructure OpenID est présentée, dans ce qui suit, dans certaines de ses versions existantes, avec ses composants, ses opérations, ses formats et ses échanges de messages. Nous montrons aussi comment un utilisateur final peut facilement utiliser son identifiant OpenID pour effectuer l'authentification OpenID décentralisée.

B.3.1) Principaux composants d'OpenID

La dernière version 2.0 d'OpenID est rétro-compatible et remplace les versions précédentes. Elle implémente la gestion de l'identité User-Centric, ainsi que le concept de la fédération d'identité [14]. OpenID est composé de plusieurs modules et leurs dernières spécifications sont les suivantes :

✓ *OpenID Authentication 2.0 :* définit la structure d'authentification de base, incluant le format et les flux des messages entre les composants de l'infrastructure ;

✓ *OpenID Attribute Exchange 1.0* : définit comment les messages d'authentification peuvent être étendus pour inclure des informations supplémentaires (attributs) sur les utilisateurs ;

✓ *OpenID Provider Authentication Policy Extension 1.0* : permet aux fournisseurs d'identité de décrire les propriétés de sécurité des mécanismes utilisés afin de vérifier l'identité de l'utilisateur ;

✓ *OpenID Simple Registration Extension 1.0* : définit un ensemble réduit d'attributs communs utilisés par les fournisseurs de services pour l'enregistrement des utilisateurs (par exemple, nom, prénom, adresse e-mail, date de naissance et la langue de préférence) ;

OpenID s'appuie sur le protocole *Yadis Discovery Protocol* pour découvrir le fournisseur OpenID qui fournit l'identificateur URL de l'utilisateur. Le module d'authentification (*OpenID Authentication 2.0*) définit également l'ensemble des concepts de base qui sont intégrés dans le Framework :

❖ *End User :* utilisateur final, représente l'utilisateur (ou la personne) qui fait usage de l'infrastructure OpenID pour établir une connexion qui va lui permettre d'avoir un accès à différents sites Web durant sa session informatique ;

❖ *Consumer or Relying Party (RP) :* c'est le site Web auquel l'utilisateur effectue des accès en utilisant son identifiant OpenID. Il est souvent appelé un consommateur « consumer » qui « consomme » l'information fournie par le fournisseur d'identité OpenID ;

❖ *Identifier :* se réfère à une URL (ou une URI, dans un cas plus général), qui représente l'identité numérique de l'utilisateur final ;

❖ *Claimed Identifier* : un identificateur qui n'a pas été vérifié par le consommateur et noté comme identifiant revendiqué ;

❖ *Verified Identifier* : un identifiant est vérifié lorsque l'utilisateur utilise des méthodes d'authentification et prouve au « consumer », qu'il est le propriétaire de l'identifiant.

❖ *Identity Provider* : un fournisseur OpenID (OP) est le serveur où les informations d'identification des utilisateurs sont stockées. Un URI OpenID d'un utilisateur se réfère à l'IdP (le serveur) où les informations d'identification sont stockées. Au cours de la procédure d'autorisation, en utilisant l'identifiant OpenID de l'utilisateur, le consommateur peut localiser l'OP et valider cet identifiant (l'URI ou URL OpenID) par échange de messages avec ce fournisseur d'identité ;

❖ *User Agent :* représente le navigateur Web de l'utilisateur final.

La section suivante illustre les protocoles de création d'un identifiant OpenID et l'utilisation de cet identifiant dans l'authentification de l'utilisateur avec les fournisseurs d'identité OpenID.

B.3.2) Principe de fonctionnement

Le fonctionnement d'OpenID est basé sur un protocole d'authentification dans lequel les OPs émettent des assertions qui prouvent que l'utilisateur possède un identifiant donné. Ces assertions sont utilisées par les fournisseurs de services (RP) afin de donner à l'utilisateur l'accès aux services. Le Protocole d'authentification OpenID spécifie les échanges de messages utilisés pour accomplir le processus d'authentification. La Figure 71, donne un aperçu du processus des échanges.

L'utilisateur démarre le processus d'authentification en présentant un identifiant au Consumer (RP) (étape 1), i.e. le site Web auquel il souhaite accéder et qui prend en charge un identifiant OpenID. Sur la base de l'identifiant fourni par l'utilisateur, le RP effectue la découverte de l'URL du fournisseur OpenID qui va authentifier l'utilisateur (étape 2). Une association est créée entre le RP et l'OP (cette étape 3 est facultative). La Clé secrète de l'association est établie en utilisant le protocole Diffie-Hellman pour échanger les clés. Cette association permet la vérification des messages échangés entre le RP et l'OP.

Le RP redirige le navigateur de l'utilisateur vers l'OP (étape 4) avec une demande d'authentification OpenID. L'OP vérifie si l'utilisateur final est enregistré et associée à cet identifiant (étape 5). La spécification ne décrit pas la façon dont les OPs doivent authentifier les utilisateurs finaux, i.e. chaque OP est libre de choisir la manière la plus appropriée pour effectuer l'authentification d'un utilisateur et s'assurer qu'il est le propriétaire de l'identifiant. Par exemple, Google utilise le mot de passe de compte Google. Après l'authentification, l'OP redirige le navigateur Web de l'utilisateur final vers le RP et fournit une assertion indiquant que l'utilisateur a été bien identifié (une affirmation positive) ou dans le cas contraire il retourne une (assertion négative) indiquant que l'authentification a échoué (étape 6). Le RP vérifie les informations envoyées par l'OP, ainsi que l'URL de retour, les informations découvertes (OP), la valeur à usage unique (Nonce) et la signature (étape 7).

La vérification peut se faire par la signature incluse dans le message, au cas où il existe une association entre OP et le RP. Dans le cas contraire, le RP peut envoyer une requête directement à l'OP pour une vérification (cette partie n'est pas représentée dans la Figure 71). Ce processus est effectué chaque fois que l'utilisateur final tente d'accéder à un RP compatible OpenID. Si l'utilisateur présente le même identifiant dans un accès ultérieur à un autre RP, les étapes 1-4 du protocole sont exécutées exactement de la même façon que dans la

première authentification. En revanche, l'étape 5 peut se produire différemment : si l'OP initie une session avec l'utilisateur final après la première authentification, cette session est réutilisée afin d'éviter à l'utilisateur de s'authentifier à nouveau. Dans ce cas, l'étape 5 est transparente pour l'utilisateur final et le protocole se poursuit comme dans la première authentification. Ce mécanisme correspond au SSO d'OpenID (i.e. l'étape 5 n'est pas spécifiée, alors que le SSO n'est pas une exigence d'OpenID, mais il est facilement mis en œuvre et généralement adopté).

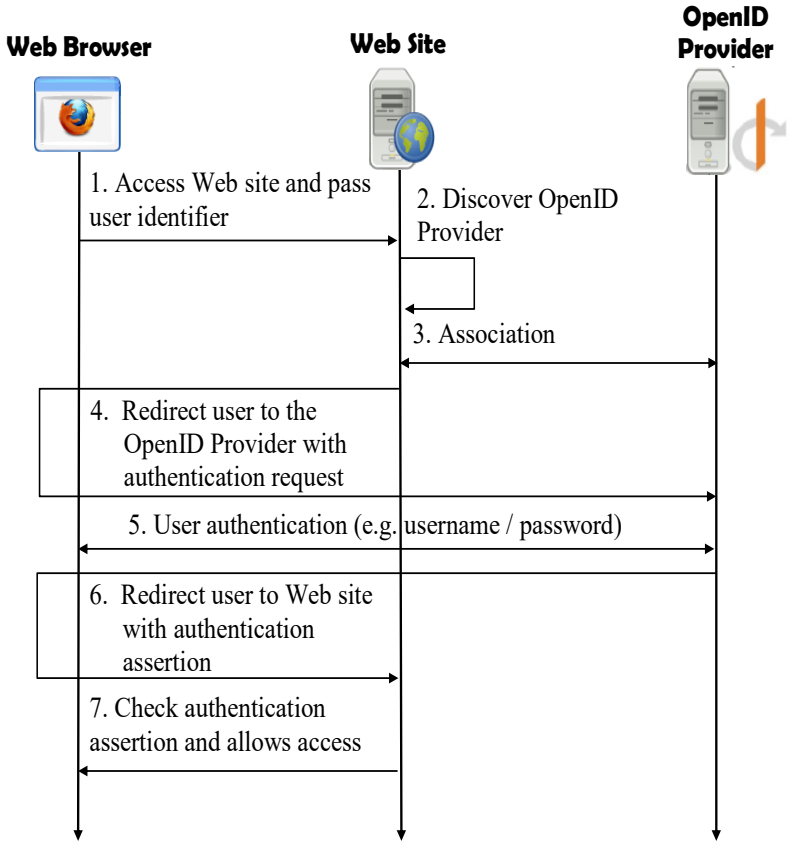

Figure 71 : L'authentification dans OpenID (*source J.Fraga de l'équipe brésilienne*)

B.3.3) Les échanges d'Attribut

Les échanges d'attributs dans OpenID sont spécifiés par le module d'extension "*OpenID attribute Exchange*". Cette spécification définit les messages qui peuvent être utilisés par les RPs pour demander l'accès aux attributs de l'utilisateur final stockés dans le serveur OP. Les demandes d'attributs stockés sont intégrées dans les requêtes standards d'authentification OpenID de manière à tirer parti des mécanismes de sécurité fournis par ces échanges. La requête d'attributs contient une liste d'attributs qui doivent être lus à partir de l'OP. En outre, le RP peut indiquer quels sont les attributs nécessaires pour que le service

puisse être fourni à l'utilisateur final. Les exigences des RPs sont justes une indication et l'OP est libre de décider quels attributs faut-il décliner (probablement pour se conformer à la politique souhaitée par l'utilisateur final). La réponse à une demande d'attribut contient une liste de valeurs d'attributs conformes à la demande du RP et à la politique de libération de l'attribut de l'OP.

Le RP peut utiliser des extensions OpenID pour obtenir des valeurs et des attributs utilisateur stockés dans un OP et puis les communiquer aux clients. L'extension "*Attribute Exchange"* fournit également un moyen pour le RP de communiquer les attributs utilisateur final aux clients. Une demande de stockage d'attributs dépend de la politique de l'OP. En plus des demandes d'attributs, l'extension "*OpenID Simple Registration"* définit un protocole simplifié d'échange d'attribut dans lequel le RP demande un ensemble réduit d'attributs prédéfinis habituellement utilisés dans l'enregistrement des utilisateurs. Ces attributs sont échangés de la même manière que les autres attributs afin qu'ils soient soumis aux mêmes règles de sécurité.

B.4) Cardspace de Microsoft

Le système Cardspace appelé à l'origine (Information Card : InfoCard) [57], est un composant de la plateforme de Microsoft. Net conçu pour offrir aux utilisateurs un support qui consiste à manipuler de multiples identités numériques. Le protocole est documenté par Microsoft et implémenté par Cardspace dans les spécifications InfoCard. La technologie Cardspace est disponible dans Windows Vista et Windows 7. Elle est également prise en charge dans le navigateur Internet Explorer (depuis la version 7.0). Elle est basée sur des fichiers d'échange XML et un protocole sécurisé compatible Web Service (*WS-Security, WS-Trust, WS-MetadataExchange et WS-SecurityPolicy*).

Le système Cardspace met l'accent sur la collecte de données de l'utilisateur, appelée "*information cards*". Il se présente sous forme d'une interface logicielle pour sélectionner une identité (*Identity Selector : IS*), similaire au portefeuille d'un utilisateur avec plusieurs cartes d'identités. Chaque InfoCard représente une identité différente (Figure 72). Lorsqu'un fournisseur de services demande les informations d'identification d'un utilisateur, l'agent de l'utilisateur sélectionne une de ces identités à partir d'un sélecteur [13].

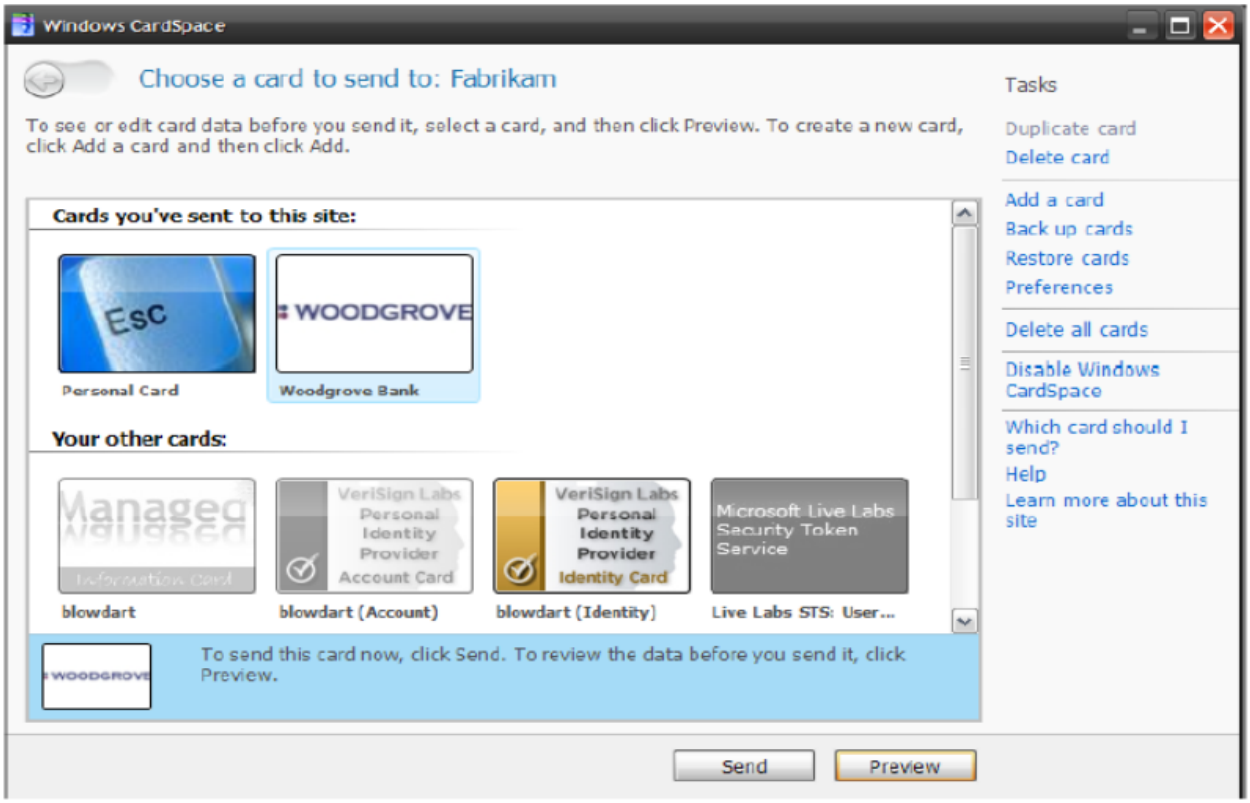

Figure 72 : Sélecteur d'Identité (Identity Selector) dans InfoCard

B.4.1) Scénario d'échanges de Cardspace

Le scénario (Figure 73) illustre les échanges d'un utilisateur (utilisant un navigateur Web) qui tente d'accéder au service B (RP-B) qui supporte la technologie Cardspace. Grâce à son IS, cet utilisateur est en mesure de choisir une identité parmi un groupe de fournisseurs d'identités, celle qui doit être présentée au RP-B. Quel que soit le choix de l'utilisateur, la cinématique des messages échangés entre les participants est la suivante :

- Étape 1) l'IS récupère les exigences de sécurité du RP auquel l'utilisateur souhaite accéder. Le RP peut demander plusieurs informations selon sa politique (le type de jeton de sécurité et les attributs qu'il doit contenir ou l'assertion que le RP peut accepter).

Une fois cette information transmise à la plateforme Cardspace, l'IS affiche à l'écran l'identité sélectionnée, sous forme de carte d'information (étape 2) qui correspond à la politique du RP.

Les cartes dont les attributs (e.g. jetons de sécurité) ne sont pas compatibles avec les exigences du RP apparaissent grisées sur l'écran et l'utilisateur ne peut pas les choisir. Après

sélection de la carte, Cardspace envoie une demande pour un jeton de sécurité à l'IdP associé à la carte, lequel renvoie alors un jeton de sécurité (étape 3). Lorsque l'IdP renvoie le jeton de sécurité signé, l'ensemble des attributs du jeton correspond à la notion d'assertion SAML. Une fois le jeton reçu, Cardspace le fournit à chaque demande au "*User Agent*" pour le présenter ensuite au RP (étape 4). Le RP peut donc utiliser ce jeton soit pour authentifier l'utilisateur soit à d'autres fins (e.g. consommer un service).

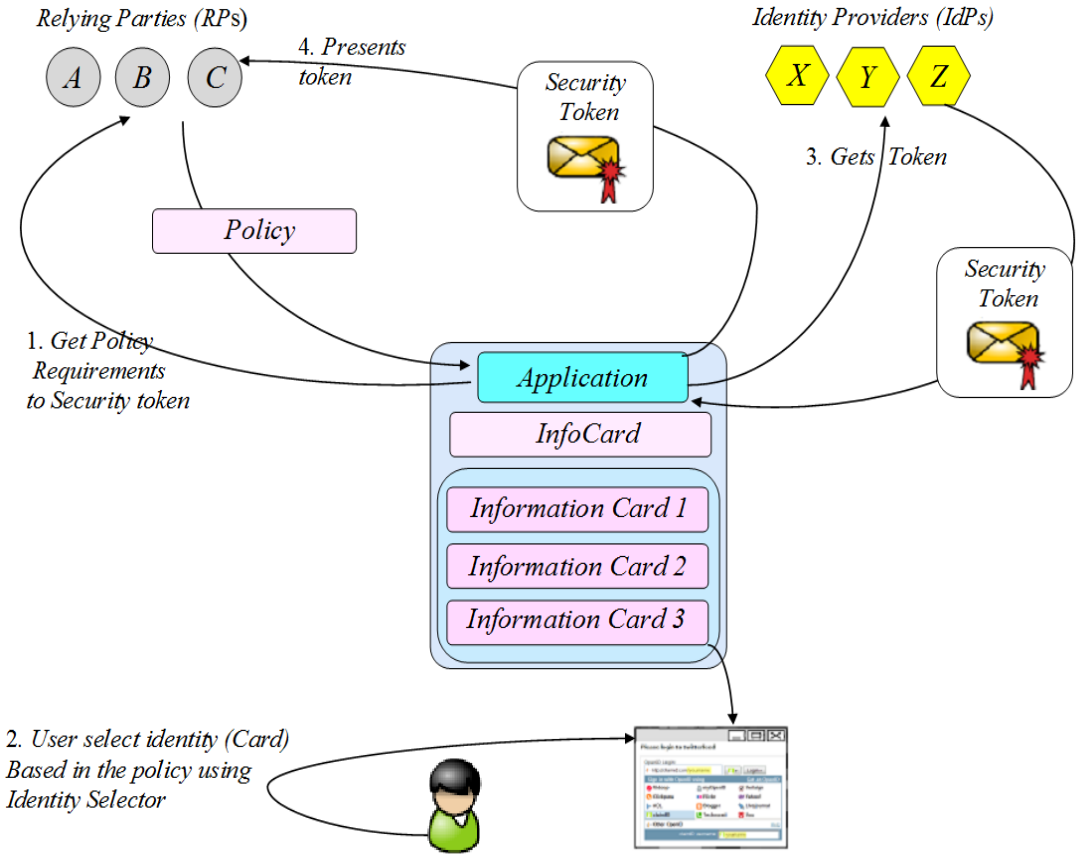

Figure 73 : Les échanges de messages dans Cardspace

B.4.2) Les différents type de carte dans Cardspace

Cardspace prend en charge deux types de cartes (*Self-Asserted ou Managed cards). Les Self-Asserted cards* représentent un ensemble d'attributs dont les valeurs sont uniquement déterminées par l'utilisateur (similaire aux identités OpenID). Dans l'implémentation de Microsoft, ces attributs sont stockés directement sur le dispositif de l'utilisateur. Les *Managed cards* correspondent à un ensemble d'attributs étendus des utilisateurs gérés par un IdP. Typiquement, à chaque fois que l'utilisateur sélectionne une carte spécifique, appropriée aux exigences du RP, le sélecteur d'identité libère les attributs de l'utilisateur correspondant à

l'IdP. Les cartes sont similaires aux identités fédérées de SAML dans lequel les IdPs contrôlent leurs attributs et leur validité.

Les approches "User-Centric" imposent des coûts de mise en œuvre. Cependant, elles permettent également d'implémenter des solutions plus élégantes et peuvent ainsi résoudre des problèmes tels que la découverte IdP. Le modèle InfoCard résout ce problème en éliminant les besoins de recourir à un fournisseur de service (Relying Party) pour se connecter aux IdPs. Dans les *Self-Asserted cards*, le dispositif du client peut être considéré comme son propre IdP, par contre, pour le *Managed Card*, les IdPs stockent leurs adresses sur le sélecteur d'identité du dispositif client pour que l'utilisateur les réutilise si nécessaire. Les *Managed Cards* reflètent l'étroite relation entre l'utilisateur et son IdP. Le sélecteur d'identité peut utiliser ces adresses pour augmenter la prévention contre des attaques de *phishing* pendant l'authentification des utilisateurs. Pour des communications d'intermédiation entre les IdPs et les RP, un sélecteur d'identité permet également aux utilisateurs d'éviter un IdP pour identifier des sites Web et des applications Web visitées (Relying Party).

La technologie Cardspace est totalement indépendante du format de jeton de sécurité requis par un fournisseur d'identité [44]. Cela signifie que le RP ne se préoccupe pas du format des jetons. En fait, Cardspace n'a généralement pas connaissance du format du jeton. Par conséquent, Cardspace peut fonctionner avec n'importe quel système d'identité, en utilisant n'importe quel type de jeton de sécurité, y compris les noms d'utilisateur sous forme de jetons (certificats X509, tickets KERBEROS, jetons SAML, etc.).

Tous les échanges illustrés à la Figure 73, et implémentés par Cardspace sont basés sur des protocoles et des standards ouverts. La politique du RP est décrite en utilisant la spécification WS-SecurityPolicy ; cette politique est obtenue en utilisant le protocole WS-MetadataExchange [58]. La sécurité du jeton est acquise en utilisant WS-Trust et à l'aide de WS-Security le jeton est transféré vers le RP.

B.5) Le projet Higgins

Le projet Higgins (Eclipse Foundation) [59] est un Framework qui accepte tous les protocoles connus pour la gestion des identités, y compris WS-Trust, OpenID, SAML, XDI, LDAP, etc. L'idée de base de ce projet est de créer une seule identité à partir des identités des autres domaines. Ce Framework définit un ensemble d'interfaces de programmation que les développeurs peuvent utiliser pour lier leur logiciel de gestion des identités au projet Higgins.

La motivation initiale du projet est de mettre en œuvre un système de gestion des identités basé sur le modèle centré sur l'utilisateur (User-Centric Identity). En d'autres termes, l'objectif de ce Framework est de permettre aux utilisateurs d'avoir plus de contrôle sur leurs identifiants et leurs profils. L'utilisateur doit être en mesure de décider quelles sont les informations qu'il veut partager et avec quels sites Web. Le projet Higgins a obtenu plusieurs contributions technologiques d'IBM, Novell, Oracle, CA, Serena, Google, Corisecio, etc. il couvre également les quatre aspects suivants :

- Il fournit un support cohérent aux cartes d'information appelées i-Cards, pour la gestion et la diffusion des données d'identité des utilisateurs ;

- Il permet aux utilisateurs un contrôle significatif sur la divulgation de leurs informations personnelles aux sites Web avec lesquels ils interagissent ;

- Il fournit une API et un modèle de données pour supporter les identités fédérées, en prenant en considération une large variété de technologies de sécurité. Grâce à cette API, le projet Higgins encourage ses développeurs à créer des plugins pour fonctionner avec des protocoles et des jetons de sécurité des systèmes existants ;

- Il fournit des plugins pour des sources de données existantes, y compris les annuaires, les systèmes de communication et les bases de données, qui peuvent aussi être intégrés au Framework ;

Les identités du modèle Higgins suivent l'approche basée sur le client actif (Identity Selector), c'est-à-dire sur une application permettant d'aider l'utilisateur à contrôler ses multiples identités et ses préférences. Il offre aux utilisateurs trois applications qui agissent comme des sélecteurs d'identité pour la création, la sélection, la gestion et le partage des i-cards qui représentent les identités des utilisateurs dans différents contextes. L'authentification centrée sur les cartes possède les mêmes avantages que Cardspace. Dans ce modèle, il est également possible de croiser des contextes et de gérer tout type d'information utilisateur stocké sur la carte (chansons préférées, numéro du permis de conduire, numéro de l'assurance sociale et régime de l'assurance-maladie). Dans le modèle Higgins, les sélecteurs sont interopérables avec Cardspace. Ces sélecteurs d'identités sont disponibles pour certains systèmes d'exploitation (Mac OSX, Linux et Windows) ainsi que les navigateurs Firefox et Internet Explorer.

B.5.1) Le processus d'authentification dans Higgins

Higgins propose deux fournisseurs d'identité (IdPs qui sont des services Web). Le premier est un STS (Security Token Service), introduit par WS-Trust (§3.4.2.1), et le second prend en charge la norme SAML 2.0.

La Figure 74, illustre l'ensemble des entités qui interviennent pendant le processus d'authentification. Higgins fournit également des bibliothèques pour les RPs nécessaires pour autoriser des sites Web et des requêtes systèmes à accepter des cartes d'information (i-cards). Ainsi, les développeurs peuvent se servir de ces bibliothèques pour les incorporer dans les RPs (code orienté i-card) de leurs applications et dans des sites Web. Un SP (Relying Partie) peut fournir l'authentification OpenID et l'authentification basée sur des cartes d'information (i-card).

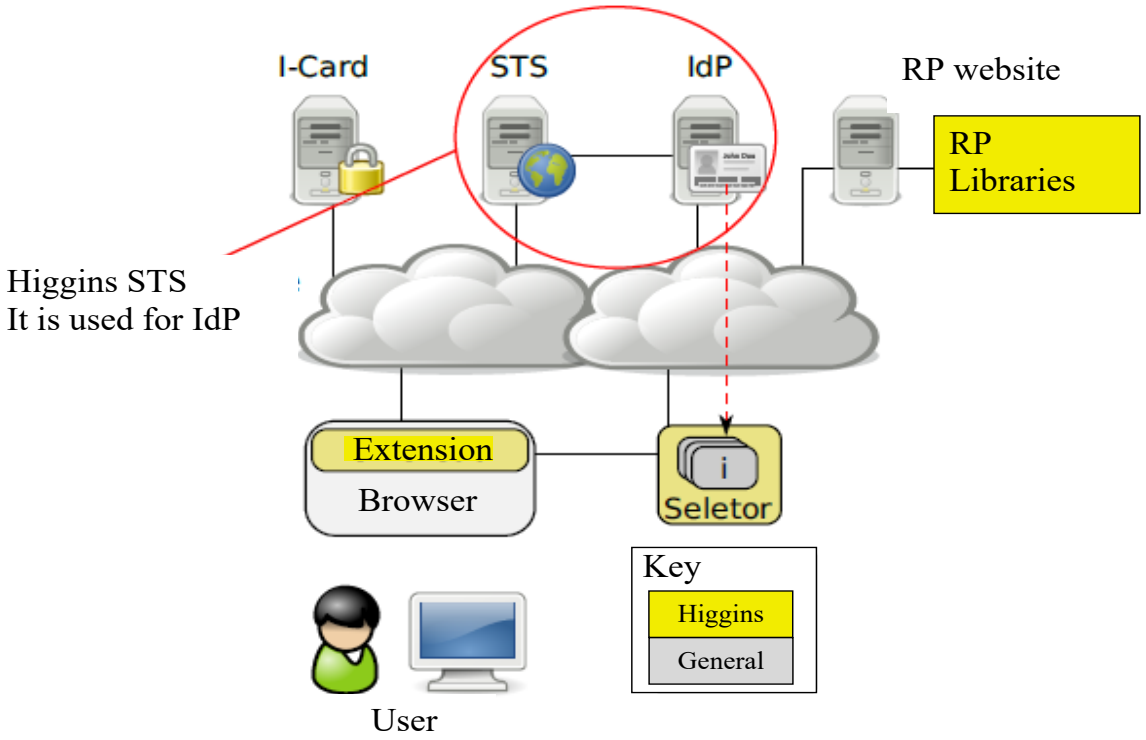

Figure 74 : Processus d'Authentification dans le projet Higgins
(*source J.Fraga de l'équipe brésilienne*)

B.5.2) Le service d'attribut d'identité (IdAS)

Le sélecteur (applications et services Web) est pris en charge par une couche d'abstraction de gestion des identités. Cette couche est constituée d'un Framework qui peut être étendu grâce à des plugins. Le niveau inférieur de ce Framework est le service d'attribut

d'identité "*Identity Attribute Service*" (IdAS) qui assure l'interopérabilité et la portabilité à travers les fédérations de données d'identité (Figure 75).

Le IdAS fournit l'accès en lecture et en écriture à une large variété de sources de données, y compris LDAP et les fichiers XML, et peut être étendu en utilisant des plugins appelés fournisseurs de contexte (Context Providers). En d'autres termes, ce service rend possible la combinaison des données provenant des réseaux sociaux et des données d'identité par le biais de sources de données très hétérogènes, y compris des annuaires et des bases de données relationnelles.

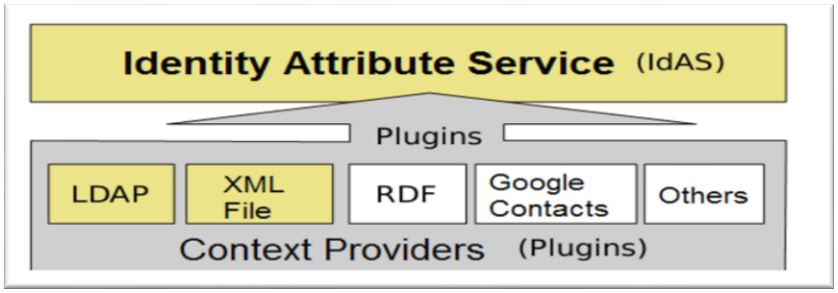

Figure 75 : Le Service Identity Attribute

Annexe C – Notion sur la virtualisation

La virtualisation a donné un nouveau souffle aux systèmes d'information et à la manière dont ces derniers doivent être organisés. Elle les pousse vers de nouvelles frontières ; scalabilité, élasticité, flexibilité, etc. La virtualisation représente une réelle opportunité pour éliminer la complexité architecturale et opérer une rupture par rapport au modèle architectural traditionnel. Cette technologie a été essentiellement appliquée à la gestion informatique et aux infrastructures de stockage.

Les architectures basées sur la virtualisation apportent des solutions au problème de mobilité de l'utilisateur. Elles permettent l'implémentation d'un haut niveau d'abstraction de bout en bout. Les différents types de ressources physiques deviennent mutualisables et flexibles, ils permettent une fluidité dans les configurations et une gestion dynamique du provisioning.

- **Les différentes technologies de virtualisation**

Il existe plusieurs technologies de virtualisation, chacune a un objectif spécifique. Les technologies les plus répandues sont :

a) la virtualisation matérielle : est une technologie qui permet de faire fonctionner n'importe quel OS sans lui apporter des modifications et sans perte de performance. Elle utilise plusieurs instructions spécifiques implémentées dans des processeurs de type AMD-V [77] ou Intel-VT [78] et d'un ensemble de primitives de bas niveau facilitant la virtualisation de plusieurs systèmes d'exploitation invités sans les modifier. Cette technologie présente les meilleures performances d'exécution.

b) la virtualisation complète ou totale : est une émulation de l'intégralité du système invité sans lui apporter des modifications. Le système hôte gère toutes les ressources nécessaires au système invité pour lui faire croire qu'il s'exécute sur un équipement physique associé. L'avantage de cette technologie est de pouvoir installer un système d'exploitation invité sans aucune modification de ses applications ou ses pilotes. On peut donc émuler un OS de manière indépendante de l'architecture matérielle du système hôte. Ce dernier doit implémenter une couche logicielle pour simuler complètement le matériel des invités. Cette couche logicielle de virtualisation transforme les requêtes d'un système invité en instructions pour le système hôte. Par conséquent, les performances de la VM seront donc très dégradées.

c) la para-virtualisation : est un intermédiaire entre le matériel et les systèmes d'exploitation invités. Contrairement à la virtualisation complète, le système d'exploitation invité modifié au préalable s'exécute sous contrôle d'un système hôte. Le système invité communique avec le système hôte via un émulateur matériel ou une couche d'abstraction appelé hyperviseur ou VMM (Virtual Machine Monitor). Ce type de virtualisation requiert la modification des OS invités pour collaborer avec la couche de virtualisation plus efficacement. Ce type de virtualisation permet des performances bien plus importantes que la virtualisation totale.

L'hyperviseur ou VMM (Virtual Machine Monitor) fournit une couche d'abstraction pour exécuter différents systèmes d'exploitation sur le même équipement physique comme s'ils fonctionnaient sur des équipements distincts. La virtualisation repose sur les composants illustrés Figure 76 :

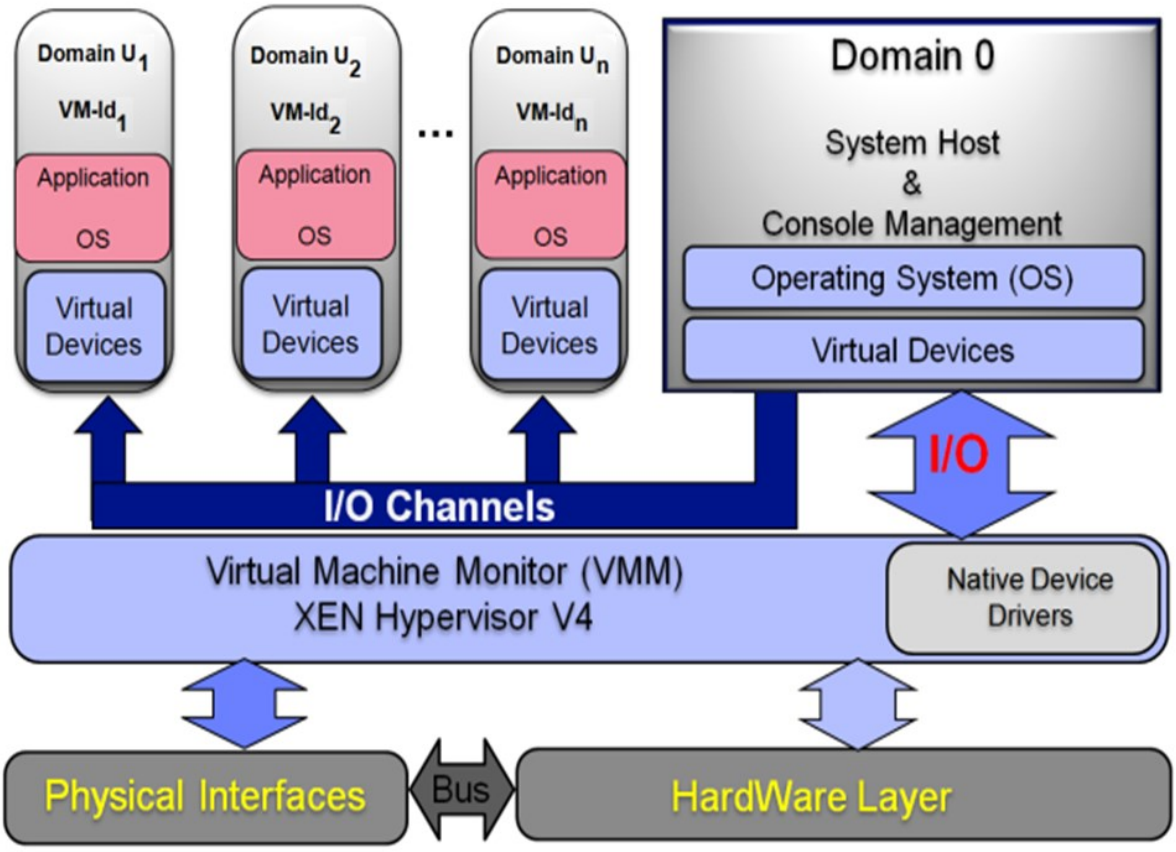

Figure 76 : Architecture de l'Hyperviseur XEN

- Un système d'exploitation principal appelé "système hôte" installé sur un serveur physique. Ce système sert d'accueil pour d'autres systèmes d'exploitation.

- Un hyperviseur installé sur le système d'exploitation principal. Il permet la création d'un environnement confiné et indépendant pour installer et instancier des systèmes d'exploitation invités appelés machines virtuelles (VM). Cette couche d'abstraction permet de déconnecter le système d'exploitation du matériel sous-jacent.
- Une VM s'exécutant dans un espace clos fonctionnant indépendamment des autres machines virtuelles. Chaque machine virtuelle dispose d'un accès aux ressources physiques du serveur (CPU, mémoire, réseau, espace disque, etc.).
- Une console de gestion permettant de gérer l'hyperviseur et les machines virtuelles.

Le logiciel XEN est un exemple d'hyperviseur libre (Open source) [19]. Il se compose d'une console de gestion, d'un système d'exploitation hôte, nommé (Domain 0), c'est le domaine privilégié ou (Privileged Domain) et de plusieurs systèmes d'exploitation invités, appelés (Domain U ou Guest), c'est le domaine non privilégié ou (Unprivileged Domain).

L'Hyperviseur XEN ordonnance le temps d'utilisation de la machine hôte pour chaque domaine. Au démarrage (boot) d'une VM, il détecte et démarre les pilotes (drivers) non initialisés par le système hôte, redirige les interruptions vers le BIOS, et énumère les bus PCI de la carte mère. Dans les processeurs compatibles aux architectures x86, XEN utilise des modes de protection de 4 niveaux d'exécution nommés anneaux ou rings, du Ring 0 (le plus privilégié) à Ring 3 (le moins privilégié) :

- Le Ring 0 est dédié à l'exécution de l'OS principal.
- Les Rings 1 & 2 sont prévus à l'origine pour la virtualisation. XEN utilise ces niveaux intermédiaires pour le domaine privilégié et pour héberger les systèmes d'exploitation invités.
- Le Ring 3 est dédié aux applications de l'espace utilisateur.

Dans une architecture x86 (64 bits), le système XEN utilise : le Ring 0 pour l'hyperviseur, le Ring 3 pour les OS invités et les applications, mais les Rings 1 et 2, ont été supprimés. Pour le partage des périphériques, XEN délègue généralement au Domain 0 la gestion des périphériques physiques et le contrôle du pilote associé. Ainsi, ce dernier peut offrir plusieurs ressources virtualisées aux différentes applications (i.e. carte graphique, stockage de données, interface réseau, etc.).

Pour assurer l'ordonnancement des entrées/sorties aux périphériques, XEN réutilise le principe piggy-backing pour partager les pilotes des périphériques avec les autres domaines.

L'hyperviseur met en place un mécanisme de découverte des périphériques grâce à la mémoire partagée. Il alloue une zone tampon sous forme d'un anneau pour échanger les informations entre les systèmes invités (Figure 77).

Dans XEN, le pilote d'un périphérique est partagé en deux parties, le Back-End et le Front-End, dans une architecture connue sous le nom de Xenbus :

✓ Le Back-End est associé au domaine responsable du matériel physique (en particulier Domain 0), lequel utilise le pilote natif du périphérique pour gérer ses entrées/sorties.

✓ Le Front-End est l'interface virtuelle du périphérique adaptée pour l'utilisation dans le Domain U.

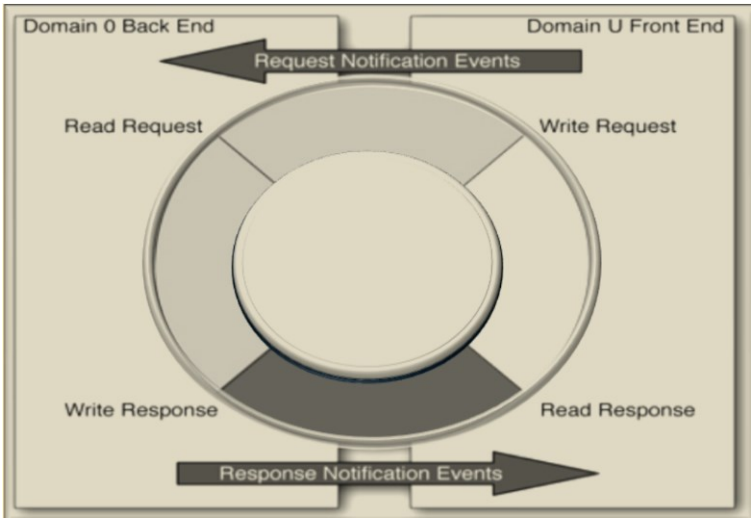

Figure 77 : le mécanisme de partage des périphériques (source [38])

Ces deux parties communiquent à travers des canaux partagés d'événement (*Request/Response Notification Events*) et le tampon en anneau. L'anneau basé sur la mémoire partagée n'est pas très efficace puisqu'il est affecté par le polling qui consiste à interroger périodiquement le statut des registres du périphérique et à identifier la présence de trafic.

Ce problème est résolu sous XEN grâce à un mécanisme plus souple d'événements qui permet l'acheminement des notifications asynchrones, lesquelles sont utilisées pour informer les deux parties du périphérique de la présence d'une requête ou d'une réponse en attente.

Dans le cas par exemple du partage d'un périphérique réseau (Figure 78), XEN crée sept paires d'interfaces réseaux virtuelles utilisées par le Domain 0.

Chaque paire **vethx/vif0.x** correspond à un domaine et ses deux composants sont reliés par un câble logique croisé ; l'interface physique eth0 est reliée à vif0.0.

Si on dispose des interfaces réseau multiples dans le Domain 0, les interfaces virtuelles sont représentées de la façon suivante :

- ✓ **EthX et reliée aux vifX.0.**
- ✓ **vethX.1 est relié à vifX.1, etc. vethX.7 est relié à vifX.7**

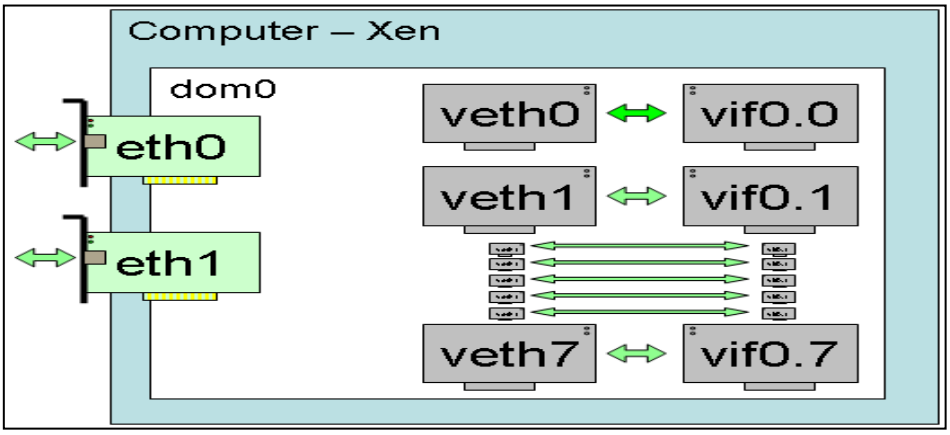

Figure 78 : Le partage du périphérique réseau (source [76])

- **Avantages et inconvénients**

La virtualisation apporte beaucoup d'avantages : une économie financière (réduction de la consommation électrique, diminution du nombre de serveurs physiques, etc.) ainsi qu'une plus grande réactivité (gestion dynamique du provisioning, gestion des configurations, scalabilité, etc.), une flexibilité. La virtualisation est un composant clé dans le Cloud Computing. Elle permet de consolider, de mutualiser et de rationaliser l'infrastructure d'une entreprise. Mais elle a aussi des inconvénients : les machines virtuelles sont tributaires du serveur physique et de l'hyperviseur ; si l'un présente des dysfonctionnements, celles-ci seront impactées (panne physique, vulnérabilité, bug, etc.).

Comme dans toute technologie, l'ajout d'une nouvelle couche d'abstraction, entraîne des problèmes de sécurité. Les performances du système sont également impactées suivant la technologie de virtualisation (matérielle, totale, para-virtualisation, etc.). En outre, certains périphériques ne sont pas adaptés à la virtualisation, ce qui nécessite l'implémentation d'une couche logicielle pour pouvoir les partager entre les systèmes invités. Les cartes graphiques et certains périphériques réseaux n'implémentent pas les techniques de partage des ressources au

niveau matériel en particulier les cartes réseaux qui comportent plusieurs interfaces (multiports). À ces inconvénients s'ajoutent des problèmes de vulnérabilité. En effet, un environnement virtuel peut accroître les risques (fuite d'information et atteinte à la vie privée, atteinte à la disponibilité des services, atteinte à l'intégrité et la confidentialité des données, etc.). Ces menaces et ces risques sont comparables à ceux des serveurs et réseaux physiques.

XEN est sujet à plusieurs vulnérabilités de type Buffer Overflow et Integer Overflow. L'attaque « access-driven side-channel » [20] permet à un utilisateur malveillant d'utiliser une VM pour écouter et surveiller le comportement d'un composant partagé dans une architecture pour ensuite obtenir, par effet de bord, des informations traitées par une autre VM cible instanciées sur le même équipement physique. Ainsi, l'attaquant peut extraire des informations ayant un intérêt pour lui (e.g. clé cryptographique) se trouvant dans la mémoire cache du système hôte. Cette attaque a été réalisée sur un système de multitraitement symétrique (SMP) lequel permet d'allouer dynamiquement de la mémoire cache aux processus, en s'assurant que ces derniers ne vont pas lire ou écrire à la même adresse. À l'aide d'un VMM moderne (XEN) installé sur un système SMP, cette attaque a été démontrée en exploitant les attaques side-channel en utilisant le square-and-multiply algorithme utilisé pour le calcul RSA. Cette menace a été classée à la première place par le Cloud Security Alliance (CSA) en 2013 [21].

- **Conclusion**

La virtualisation se révèle donc être une très bonne solution pour rationaliser l'utilisation des ressources informatiques et réduire ainsi la consommation de l'énergie électrique. Elle permet d'exécuter des systèmes invités indépendamment de l'architecture physique pour les rendre mutualisables, flexibles, etc.

Selon la technologie de virtualisation utilisée, les performances peuvent se rapprocher de celles d'un système natif ou peuvent être totalement dégradées. Certains constructeurs (Intel et AMD) ont apporté quelques modifications à leur matériel pour supporter la virtualisation matérielle pour permettre à des systèmes invités d'être virtualisés sans lui apporter de modification.

La virtualisation présente de nombreux avantages mais apporte aussi des problèmes de sécurités liés aux équipements et aux logiciels de virtualisation.

Annexe D - Le modèle de gestion des identités et le contrôle des attributs centrés sur l'utilisateur (contribution de l'équipe brésilienne).

D.1) Introduction

Comme nous l'avons évoqué précédemment, un grand nombre d'applications Internet existantes emploient l'approche traditionnelle de la gestion des identités centrée sur le SP. Ce dernier implémente alors lui-même tous les services et les contrôles nécessaires pour gérer le cycle de vie d'une identité (inscription, authentification, autorisation, etc.) [8] [9] [10]. Ce modèle présente plusieurs lacunes, en particulier le manque d'interopérabilité des SPs et l'augmentation de leur nombre, ce qui oblige les utilisateurs d'une part à gérer un grand nombre de comptes et à maintenir plusieurs informations d'identification (certificats, login/mot de passe, etc.) et d'autre part à garder ces informations d'identification à jour dans les différents SPs [22]. Ces SPs doivent maintenir des bases de données ou des référentiels des identités pour garantir la confidentialité de ces informations sensibles. Plusieurs modèles et infrastructures ont été proposés pour faire face à ces problématiques en utilisant des approches différentes. La Figure 79 illustre l'architecture la plus commune du modèle de gestion des identités où les SPs délèguent à l'IdP la fonction d'authentification.

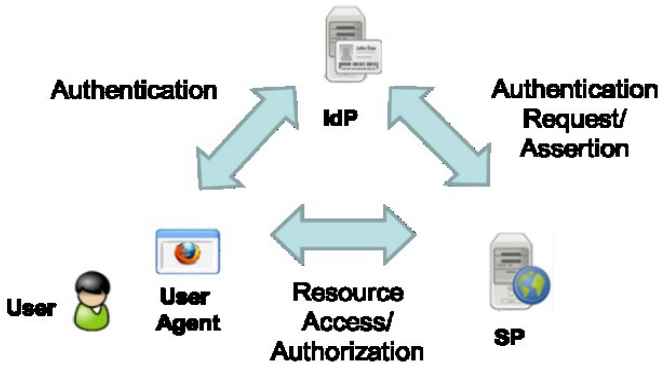

Figure 79 : Les composants du système de gestion des identités et leurs relations

Ce dernier sert de médiateur pour le processus d'authentification des utilisateurs et une fois ces derniers sont authentifiés, ils seront autorisés à accéder aux différents SPs, lesquels s'appuient également sur cet IdP pour obtenir une confirmation de leur authentification. Dans ce modèle, l'IdP est approuvé à la fois pour fournir des informations d'authentification et d'autorisation fiables aux SPs et pour préserver l'identité de l'utilisateur ainsi que ses

attributs. De plus, le mécanisme d'authentification, le plus populaire implémenté dans la grande majorité des IdPs actuels, est le fameux (login/mot de passe), lequel comporte plusieurs risques de sécurité, car il dépend en général de la capacité humaine limitée à créer et à se rappeler des phrases secrètes [22]. Dans le standard OpenID, ce couple (login/mot de passe) est largement déployé, ce qui le rend léger et relativement simple à mettre en œuvre. Mais en plus des problèmes évoqués ci-dessous ce mode est vulnérable aux attaques de type phishing [25] [45]. Pour éliminer ces risques, l'équipe brésilienne a proposé dans ce projet un nouveau modèle de gestion des identités et une approche de contrôle des attributs par l'utilisateur qui s'appuie fortement sur l'utilisation des éléments de sécurité.

L'objectif de cette annexe est de détailler ce nouveau modèle d'authentification et d'autorisation pour SecFuNet (§C.4) basé sur deux approches : une approche de gestion de l'authentification centrée sur l'utilisateur (*User-Centric Identity*) et une approche de contrôle d'attributs elle aussi centrée sur l'utilisateur (*User-Centric Attribues Control*). Cette infrastructure est basée sur des briques logicielles architecturales et des cartes à puce pour établir des liens logiques et des relations de confiance entre les utilisateurs, l'IdP SecFuNet et les SPs. Les attributs de l'utilisateur ne sont plus stockés dans les bases de données des IdPs mais dans la carte à puce de l'utilisateur. Celui-ci peut décider, à chaque instant, quelle identité ou quel profil (ensembles d'attributs) sera publié au cours d'une session. Nous détaillerons dans la suite l'implémentation de ces briques architecturales et l'orchestration de chaque composant dans l'infrastructure SecFuNet (§C.5).

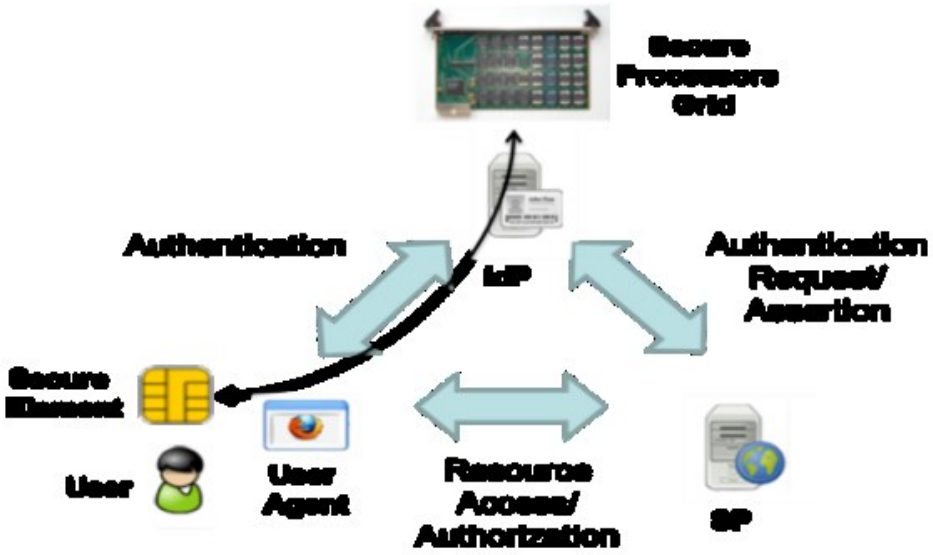

Figure 80 : Le système global d'identité SecFuNet Identity

Nous décrivons brièvement une authentification mutuelle entre le composant Identity Selector et l'IdP SecFuNet et l'établissement d'un canal sécurisé pour protéger les échanges des clés sensibles entre deux ressources. Cette authentification est effectuée directement entre les cartes à puce et une grille de cartes à puce reliée à un IdP SecFuNet, comme l'illustre la Figure 80.

Ensuite, nous présentons quelques rappels sur le modèle général Cryptoki (§C.2) qui permet de gérer les cartes à puce et sur la gestion des attributs (§C.3) dans le standard OpenID.

D.2) Rappels sur le modèle général Cryptoki

PKCS#11 est une API multiplateforme appartenant à la famille de standard cryptographique à clé publique (PKCS : Public-Key Cryptography Standard), aussi appelé Cryptoki. C'est une norme publiée par les laboratoires RSA [73]. PKCS#11 est une API qui permet d'interagir avec un ou plusieurs éléments sécurisés et d'accéder aux fonctions de chiffrement fournies par ces éléments appelés Jetons cryptographiques. Ces jetons cryptographiques peuvent être de plusieurs formes : cartes à puce, jetons USB, TPM, HSM (Hardware Security Module), etc. contenant un microprocesseur et une mémoire. Ces dispositifs ont été conçus pour conserver et gérer des informations sensibles : stockage en toute sécurité de clés privées et de certificats, identifiants, etc. Ils sont capables d'exécuter des opérations cryptographiques en interne (signature numérique, chiffrement, déchiffrement, etc.).

La spécification définit une API sous forme de bibliothèque destinée à être utilisée par des applications qui nécessitent un accès à un ou plusieurs jetons pour accélérer les opérations cryptographiques et conserver des clés en toute sécurité. La Figure 81 montre le modèle général Cryptoki [73]. Les bibliothèques PKCS#11 fournissent une interface à un ou plusieurs dispositifs cryptographiques branchés dans des emplacements (slots, voir § 4.2.1). Chaque slot peut contenir un jeton qui correspond à un lecteur physique. Une application peut se connecter à un jeton et établir une session pour accéder aux objets et aux fonctions cryptographiques qu'il offre à condition que l'utilisateur fournisse le code PIN secret de sa carte à puce.

Les objets peuvent être : des clés cryptographiques (publiques, privées ou secrètes), des certificats numériques ou des applications spécifiques. Ces objets peuvent être stockés de

façon permanente dans le jeton appelé Token Objects ou provisoire pendant la durée d'une session. Dans ce cas, on parle alors de Session Objects. Ils sont définis comme un ensemble d'attributs spéciaux (SENSITIVE et EXTRACTABLE) déterminant ainsi la façon dont un secret matériel peut être extrait à partir du jeton. Si un objet est marqué NOT EXTRACTABLE, il ne pourra pas être lu, même s'il sera utilisé comme clé matérielle pour les fonctions cryptographiques exécutées au sein du jeton. Si l'objet est marqué SENSITIVE, il peut être uniquement lu sous une forme cryptée, en utilisant une clef publique ou secrète. Ces objets sont utilisés en entrée et en sortie dans des fonctions cryptographiques implémentant des algorithmes standards. Certaines fonctions relatives à la création et à la dérivation de clés secrètes présentent un intérêt particulier pour le projet SecFuNet, car elles constituent la base pour la mise en place de connexions TLS.

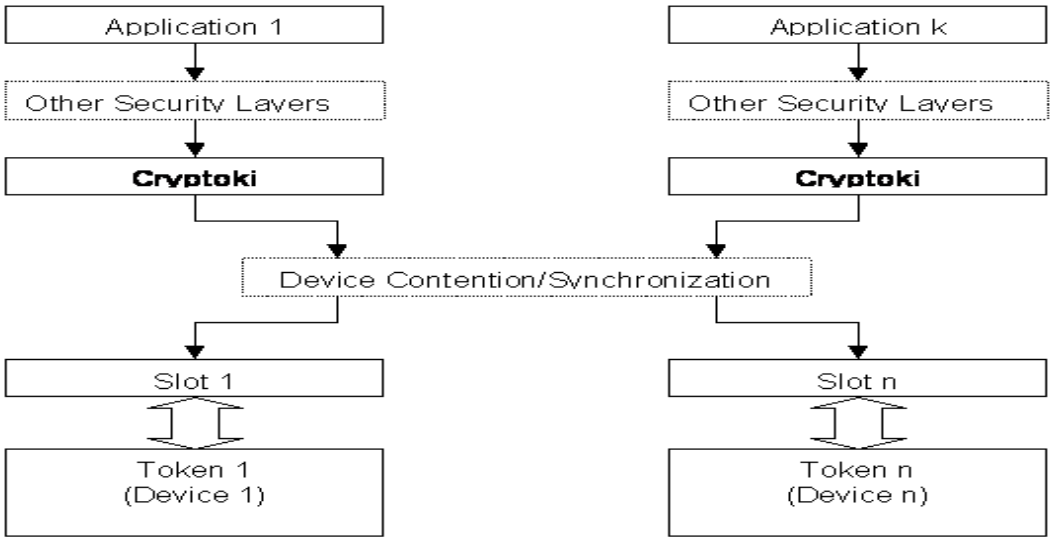

Figure 81 : Le modèle général Cryptoki (source [73])

Les fonctions Cryptoki sont sous forme d'algorithme de chiffrement, de génération de nombres aléatoires, ou de fonctions de gestion [73] et peuvent être classées dans les principales catégories suivantes :

- ✓ Lister le nombre des emplacements disponibles, déterminer la présence ou non de jeton et interroger les jetons sur les fonctionnalités supportées (i.e. mécanismes cryptographiques pris en charge) ;
- ✓ Établir et finaliser les sessions avec un jeton, gérer la connexion (par exemple l'authentification basée sur un PIN) ;

- ✓ Se charger de la création, de la copie, de la suppression des objets, de l'interrogation et de la définition des paramètres de l'objet ;
- ✓ Générer des clés (ou des clés paires), prendre en charge l'encapsulation et/ou la désencapsulation, et la dérivation des clés ;
- ✓ Se charger du chiffrement et/ou du déchiffrement, du hachage des messages, (signature/MACing) et (signature/vérification des MACs) ;
- ✓ Se charger de la génération des nombres aléatoires (ou pseudo-aléatoire).

D.3) Rappel sur la gestion des attributs dans OpenID

Au moment d'une authentification, l'OP (OpenID Provider) agit comme un IdP et envoie des assertions d'authentification et des attributs de l'utilisateur pour les fournisseurs de services, appelés Relying Parties (RPs). Après ce processus d'authentification, l'utilisateur peut accéder à un RP.

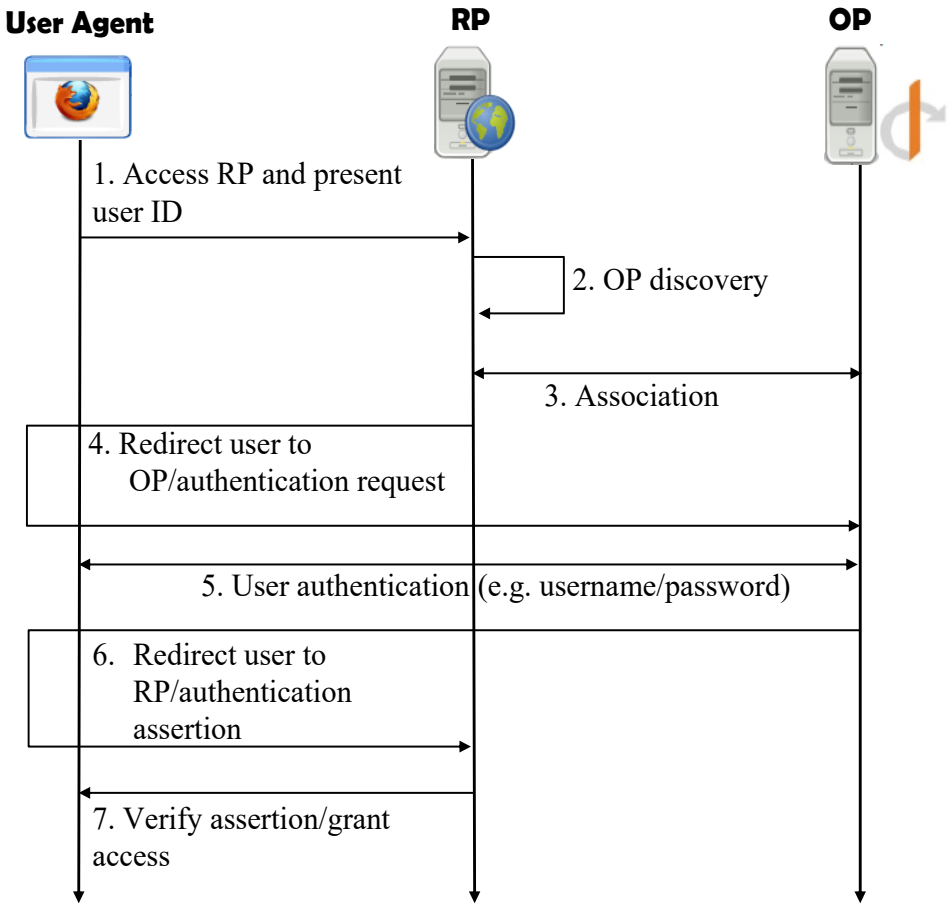

Figure 82 : Les échanges d'OpenID

Si par la suite, ce dernier souhaite accéder à un autre service, certaines étapes du processus d'authentification seront ignorées, car l'OP crée une session d'authentification avec le navigateur de l'utilisateur, de telle sorte que les accès suivants dans cette même session ne nécessitent pas une nouvelle authentification de l'utilisateur. C'est ainsi que OpenID supporte le SSO pour simplifier les accès à plusieurs RPs aux utilisateurs dans une même session d'authentification. La Figure 82, illustre les échanges entre l'utilisateur, le RP, l'OP : le RP peut demander un ensemble d'attributs utilisateur pendant une demande d'authentification (étape 4). Ces attributs peuvent être envoyés au RP avec l'assertion d'authentification (étape 6). Ainsi, le RP peut effectuer des contrôles d'accès supplémentaires basés sur des attributs de l'utilisateur avant d'accorder l'accès à une ressource. La spécification OpenID n'impose aucun mécanisme d'authentification (à l'étape 4), et aucun mécanisme pour stocker les attributs de l'utilisateur (étape 6).

Les spécifications du standard OpenID sont utilisées, dans (§C.4.3), comme protocole de découverte et d'échange d'informations. En outre, les risques potentiels qui pèsent sur le standard OpenID sont évités grâce à l'intégration des cartes à puce et au stockage des attributs sous le contrôle de l'utilisateur.

D.4) L'infrastructure d'authentification et d'autorisation

L'infrastructure d'authentification et d'autorisation de SecFuNet est basée sur OpenID et sur des éléments matériels sécurisés. L'authentification des utilisateurs est centralisée dans des IdPs compatibles avec les spécifications d'OpenID.

L'IdP SecFuNet est constitué de plusieurs applications (Figure 83) : un sélecteur d'identité (Identity Selector) du côté utilisateur et d'un serveur d'authentification côté IdP SecFuNet. Du côté utilisateur, il s'agit d'une seule carte à puce qui constitue son jeton (Token), par contre du côté de l'IdP SecFuNet, le serveur d'authentification est relié à une grille de processeurs sécurisés. La carte à puce de l'utilisateur SecFuNet contient une clé privée et le certificat correspondant qui constituent les informations d'authentification de cet utilisateur.

La clé privée n'est certainement pas extractible, étant donné que c'est l'attribut de sécurité majeur stocké dans la carte à puce. Les attributs de l'utilisateur peuvent également être marqués comme SENSITIVE, ce qui indique qu'ils peuvent être uniquement extraits de la

carte à puce au moyen d'une clé. L'Infrastructure globale se compose de deux parties que nous décrivons ci-dessous :

A) Partie : User Agent

Cette partie cliente se compose principalement de l'agent utilisateur (i.e. le navigateur) et de la carte à puce. Le sélecteur d'identité (IS) est un composant logiciel qui s'intègre et s'exécute dans le navigateur. Il fournit une interface utilisateur pendant le processus d'authentification et le contrôle d'attributs de l'utilisateur. Ces attributs sont stockés et sécurisée dans la carte à puce et ne peuvent être manipulés ou contrôlés que par l'utilisateur qui possède le code PIN de celle-ci. Le sélecteur d'identité indique à l'utilisateur l'attribut demandé et après une confirmation de ce dernier, le sélecteur ne lit que cet attribut (§C.4.1).

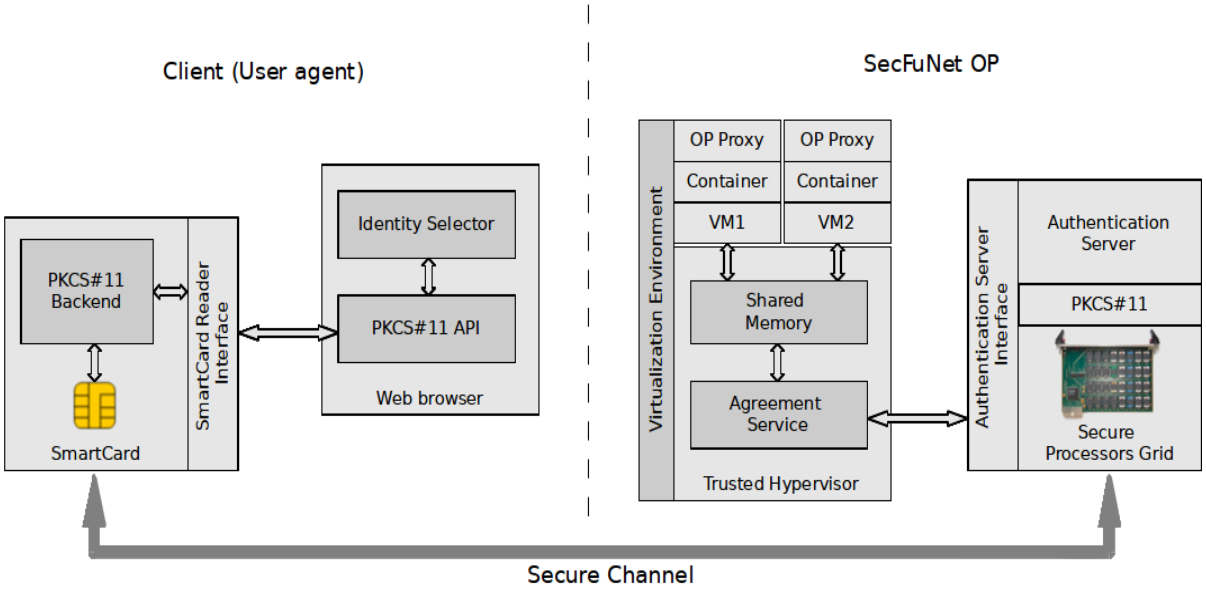

Figure 83 : Le modèle d'authentification de SecFuNet

B) Partie : SecFuNet OP

La partie IdP de SecFuNet est constituée d'un environnement de virtualisation et d'un serveur d'authentification (AS) (Figure 83). Ce serveur est relié à une grille de cartes à puce. Chacune conserve les données des SP et contrôle l'authentification des utilisateurs. L'AS n'est pas visible du réseau et ne peut communiquer qu'à travers un environnement de virtualisation (VE), responsable de l'acheminement du protocole OpenID. Il agit comme un serveur frontal (Front-End) pour l'AS.

Cet environnement est constitué d'une ou plusieurs machines virtuelles. Chacune exécute une instance nommée OP Proxy. Ce Proxy ne stocke pas de données utilisateurs, mais est simplement utilisé pour assurer les échanges du protocole OpenID. Toutes les fonctionnalités liées à l'authentification sont déléguées au serveur d'authentification au moyen d'une communication de confiance implémentée dans l'hyperviseur du VE (§C.4.2).

Du côté client, ces opérations de gestion cryptographique sont fournies directement par l'utilisation des API PKCS#11 (§C.5.1) (§C.6). Du côté OP, les mêmes fonctionnalités sont implémentées grâce à des appels de fonction spécifique émis par le conteneur OP Proxy vers l'AS. Ce dernier convertit ces appels sous forme d'appels PKCS#11 et les envoie à la grille de microcontrôleurs sécurisés (§C.5.1).

L'objectif de ce mécanisme d'authentification est d'établir un canal TLS sécurisé avec authentification mutuelle, reliant logiquement les éléments sécurisés situés aux deux extrémités. Le tunnel TLS est établi entre le sélecteur d'identité et l'OP Proxy, dans lequel les algorithmes cryptographiques, les clés privées et les secrets partagés sont stockés en toute sécurité (au sein des éléments sécurisés).

Cette session TLS fournit donc un canal pour échanger en toute sécurité des informations entre le client et l'IdP. Elle est utilisée pour notifier au sélecteur d'identité l'attribut qui a été sollicité par le RP et également pour envoyer les attributs de l'OP lus à partir de la carte à puce. Ce protocole d'authentification sera détaillé à la section (§C.4.3).

D.4.1) Composants de l'architecture du client SecFuNet

L'utilisateur accède au fournisseur OpenID en utilisant le navigateur Web et télécharge le sélecteur d'identité. Ce composant a des caractéristiques similaires à celles des projets : CardSpace [44] et Higgins [59]. Il intègre une interface utilisateur pour gérer le protocole d'authentification et le transfert des attributs de l'utilisateur, il assure les accès à la carte à puce. C'est une application de confiance signée par l'IdP SecFuNet ou par une autre entité de confiance approuvée par l'utilisateur.

Pour éviter le recours à une application séparée, le sélecteur d'identité a été développé en Java qui s'intègre de façon transparente dans les navigateurs Web et qui supporte facilement les signatures numériques.

La communication entre la carte à puce et l'IdP SecFuNet est arbitrée par l'IS qui s'exécute dans le navigateur et utilise les bibliothèques PKCS#11 pour se connecter à la carte à puce de l'utilisateur. La Figure 84 illustre l'architecture du client SecFuNet.

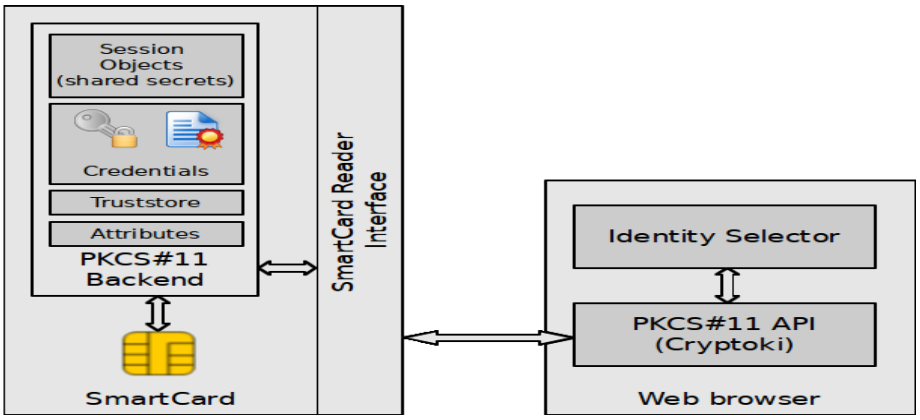

Figure 84 : L'architecture du client SecFuNet

La carte exécute l'application embarquée PKCS#11 Back-End qui contient les informations d'identification de l'utilisateur (clé privée et certificat), ses attributs et ses profils d'identité, ainsi que le certificat de l'OP de confiance.

Les objets Session Objets tels que définis dans PKCS#11 sont des objets qui sont supprimés lorsque la connexion avec la carte à puce est terminée (i.e. quand l'utilisateur retire la carte du lecteur). Dans ce scénario, ces secrets sont principalement partagés au cours d'une session TLS.

D.4.2) Composants de l'architecture de l'OP SecFuNet

La Figure 85 illustre l'imbrication d'un ensemble de couches impliqué dans l'intégration d'un serveur d'authentification et du standard OpenID dans le modèle SecFuNet.

Le serveur d'authentification est relié à une grille de cartes à puce, centralise les authentifications des utilisateurs dans un domaine où on peut appliquer une politique de sécurité (entreprise, administration, etc.). Le contrôle des décisions d'authentification des utilisateurs est géré par chaque carte à puce associée à une ressource qui conserve ses données dans ce domaine.

Les OP Proxy implémentent le protocole OpenID et sont exécutés sur des VMs. Celles-ci sont considérées par l'utilisateur comme des fournisseurs d'identité OpenID

puisqu'elles respectent les spécifications du protocole OpenID. Ces proxys ne contrôlent pas nullement les données utilisateurs, mais transmettent leurs informations d'authentification vers le serveur d'authentification via le serveur OpenID appelé Proxys OP. Ce dernier est maintenu isolé de l'environnement de réseau.

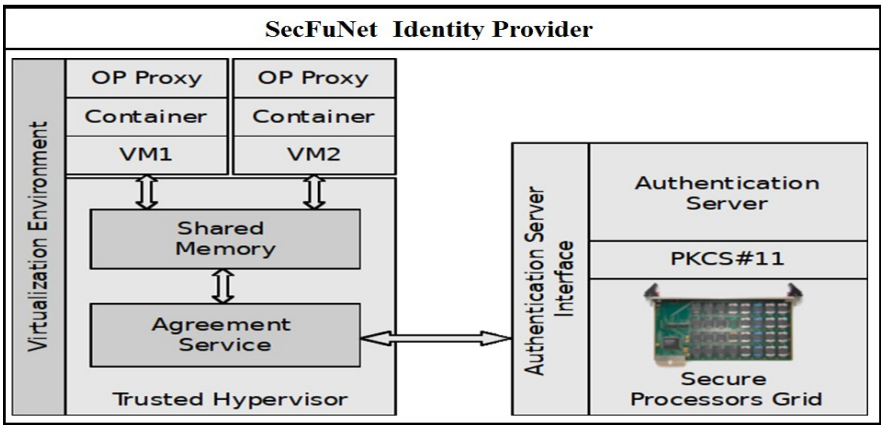

Figure 85 : Le fournisseur d'identité SecFuNet

L'hyperviseur XEN offre un bloc de mémoire partagée Shared Memory pour les VMs afin de faciliter les échanges avec un composant de confiance nommé Agreement Service (AS). Ce dernier est responsable de l'interaction avec le service d'authentification relié à la grille d'éléments sécurisés.

Ces deux couches d'abstraction (le bloc mémoire et l'AS) sont implémentées dans le niveau de privilège Domain0 de l'hyperviseur XEN. Le confinement de celles-ci est une obligation pour assurer une confiance dans les IdPs SecFuNet et pour limiter la compromission de ces composants par des attaques réseau. Ainsi, les attaques et les intrusions sont limitées aux seules machines virtuelles visibles de l'extérieur exécutant les OP proxys.

Lorsqu'une VM est compromise, l'administrateur de l'hyperviseur peut facilement la remplacer par une nouvelle instance pour stopper une éventuelle fuite de données sensibles des utilisateurs ou du serveur sans affecter le fonctionnement global du système.

L'AS est responsable de la gestion des VMs et des OP Proxy et fait office d'intermédiaire entre les OP Proxy et le serveur d'authentification par le biais d'un lien dédié et des protocoles appropriés.

L'AS remplit plusieurs fonctions : vérifie les requêtes reçues par les OP Proxy depuis des clients, s'interface avec le serveur d'authentification, crée des assertions OpenID qui doivent à la fois refléter le résultat de la procédure de connexion et contenir des attributs

transmis par l'utilisateur. C'est l'AS qui maintient ces assertions au cours de la session de l'utilisateur et doit les transmettre aux fournisseurs de services (RPs) suite à leurs demandes selon les spécifications OpenID.

L'interface Authentication Server interface (Figure 85) permet d'isoler le serveur d'authentification du reste du réseau externe. Cette interface offre les fonctionnalités suivantes :

- *VerifyCertificate ()* : qui vérifie les certificats de l'utilisateur envoyés en tant que paramètres pour le serveur d'authentification et renvoie une réponse positive ou négative en fonction du succès ou de l'échec de la vérification de l'AS.
- *CreateSign ()* : qui renvoie une signature réalisée avec la clé privée du serveur d'authentification sur les données passées en paramètre.
- D'autres fonctionnalités sont implémentées en utilisant des mécanismes similaires au standard PKCS#11 pour réaliser des opérations cryptographiques nécessaires pour l'établissement du canal TLS. Celles-ci sont semblables aux fonctions *C_UnwrapKey et C_DeriveKey* du standard PKCS#11. Elles sont utilisées par l'AS pour stocker les secrets dans les éléments sécurisés et gérer la création du canal TLS.
- Les fonctionnalités liées à la gestion du cycle de vie des cartes à puce, qui ne sont visibles que par les administrateurs du serveur d'authentification : création et suppression des comptes utilisateurs, des certificats, introduction et suppression des attributs, etc.

D.4.3) Modèles User-Centric dans SecFuNet

Les spécifications OpenID n'imposent aucun mécanisme d'authentification ou de stockage des attributs des utilisateurs ; deux modèles basés sur des cartes à puce et centrés sur l'utilisateur on était proposés dans le projet SecFuNet : la gestion des identités centrée sur l'utilisateur (User-Centric Identity) et le contrôle de l'attribut centré sur l'utilisateur (User-Centric Attributes Control). Le protocole d'authentification mutuelle proposé est basé sur des certificats numériques.

Les deux parties stockent la clé privée dans un élément sécurisé inviolable. La carte à puce de l'utilisateur sera également capable de fournir un stockage sécurisé de ses attributs. Tout le calcul cryptographique est effectué au sein de la carte à puce, sans que sa clé privée

soit divulguée. En outre, toutes les interactions avec la carte à puce nécessitent un code PIN avant d'autoriser l'accès aux données sensibles.

Un autre aspect important du protocole proposé est de permettre à l'utilisateur de contrôler ses propres attributs d'identité, stockés dans sa carte à puce avec ses informations d'identification. L'IdP SecFuNet ne stocke pas ces attributs, mais informe l'utilisateur quand le RP en demande certains. C'est l'utilisateur qui décide de divulguer ou d'autoriser la lecture de ceux-ci et de les transférer au RP.

Dès que l'OP Proxy envoie les attributs exigés par la politique du RP, dans une assertion signée par le serveur d'authentification, l'IdP SecFuNet peut exiger que des attributs soient signés par une autorité de confiance. Ces attributs peuvent être stockés uniquement par une entité d'enregistrement de confiance sur la carte à puce. L'utilisateur ne pourra pas les modifier bien qu'il a le contrôle sur ses attributs.

Un domaine de politiques de l'IdP SecFuNet peut établir des profils spécifiques ou créer des identités numériques pour chaque utilisateur selon ses attributs signés. Chaque profil d'identité peut être stocké dans la carte à puce selon différents niveaux d'assurance (LoA).

Ce modèle de confiance suppose que le RP a une liste des IdPs SecFuNet de confiance. La confiance dans un IdP SecFuNet implique que le RP doit accepter toutes les assertions d'authentification et les attributs de cet IdP. Cette confiance accordée aux IdPs est gérée conformément aux politiques du RP, contrairement au modèle fédéré dans lequel les fournisseurs de services et d'identité constituent un domaine commun.

Dans les spécifications OpenID la confiance est unidirectionnelle : l'OP ne s'engage pas dans la sélection des RPs de confiance mais laisse le choix à l'utilisateur de décider de dévoiler ou non ses informations d'identité.

La confiance dans un IdP SecFuNet ne suffit pas à accorder aux utilisateurs authentifiés l'autorisation d'accéder à des ressources fournies par le RP. Le RP peut posséder ses propres politiques de contrôle des accès complexes, basées sur des attributs fournis par l'utilisateur et relatives à un IdP. Il est également possible que des niveaux d'accès différents peuvent être exigés en fonction de la quantité d'informations dont un utilisateur dispose.

D.4.4) Chorégraphie des échanges dans l'IAA SecFuNet

La Figure 86 illustre les étapes du protocole d'authentification dans l'infrastructure SecFuNet. Aucune modification des RPs n'est nécessaire pour s'adapter à l'infrastructure proposée car les étapes (1 à 8 et 20 à 21) impliquant le RP se déroulent d'une manière similaire à la norme OpenID.

Lorsque l'utilisateur tente d'accéder à une ressource protégée dans un RP (étapes 1 à 5), le RP découvre le fournisseur d'identité OpenID (étape 6), puis, une association est établie entre l'IdP de l'utilisateur et le SP (étape 7). Le SP redirige ensuite le navigateur Web de l'utilisateur vers l'OP Proxy (étape 8). Lors de cette redirection le SP indique les attributs d'identité qui sont nécessaires pour le contrôle des accès (ou indique sa politique d'accès). L'OP Proxy présente sa page de connexion avec le sélecteur d'identité (étapes 9 et 10).

Le sélecteur d'identité demande l'accès à la carte à puce de l'utilisateur ainsi que le code PIN correspondant, pour effectuer une authentification basée sur les certificats (étape 11 et 12). L'utilisateur saisit le code PIN et déverrouille la carte à puce (étape 13).

L'IS utilise les informations d'identification de l'utilisateur stockées dans la carte à puce pour réaliser une authentification mutuelle entre la carte et le serveur d'authentification, pour ensuite établir un canal sécurisé (étape 14). Ce canal est utilisé ultérieurement dans les transferts d'attributs de l'utilisateur à partir de la carte à puce vers l'IdP SecFuNet.

Si le certificat de l'utilisateur n'est pas reconnu, par l'IdP, alors l'authentification échoue. Dans le cas contraire, les échanges continuent, et l'IS indique à l'utilisateur la liste des attributs requis par le RP pour effectuer le contrôle des accès par attributs (étapes 15 et 16).

Si l'utilisateur décide de dévoiler ses attributs (étape 17), l'IS les lit à partir de la carte et les envoie à travers le canal sécurisé (étape 18). L'IdP génère une assertion qui sera signée par le serveur d'authentification, contenant le message d'authentification et les attributs de l'utilisateur (étape 19). Cette étape est contrôlée par l'AS.

L'OP Proxy reçoit les attributs de l'utilisateur et les transmet à l'AS en utilisant les primitives d'accès à la mémoire partagée disponibles pour les VMs dans l'environnement virtualisé de l'IdP SecFuNet.

L'AS formate cette assertion d'authentification et ces attributs conformément aux spécifications d'OpenID et les transmet au serveur d'authentification afin qu'il les signe.

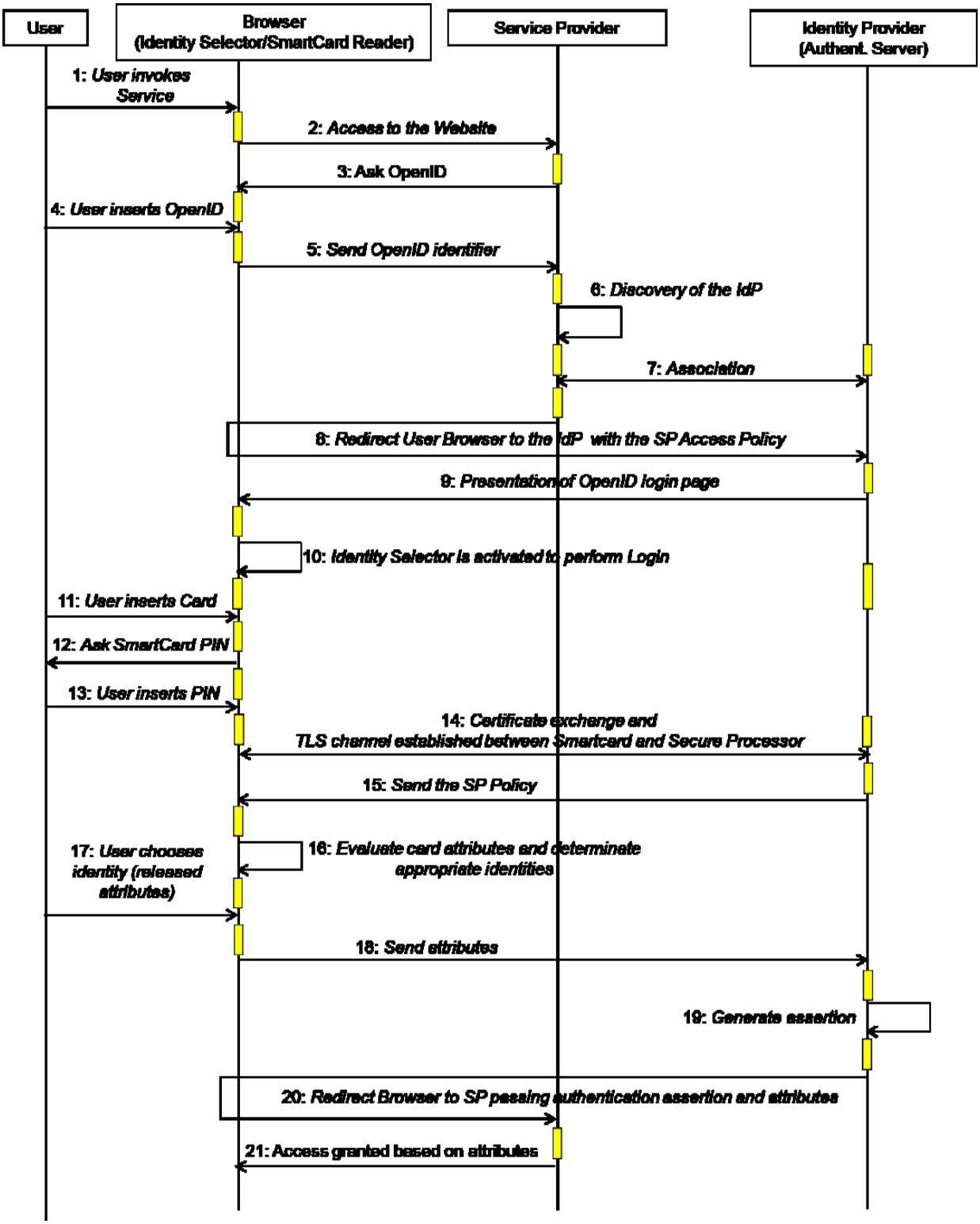

Figure 86 : Les échanges du protocole d'Authentification

Cette assertion est signée avec la clé secrète partagée par le serveur d'authentification et le RP et négociée dans l'étape (7), de sorte que le RP puisse vérifier son authenticité. L'OP Proxy (i.e. l'IdP SecFuNet) redirige l'utilisateur en relayant cette assertion vers le RP (étape

20). Enfin, le RP applique les politiques de contrôle des accès afin d'accorder ou de refuser à l'utilisateur l'accès à une ressource (étape 21).

Les procédures d'authentification et d'accès aux différents RPs pendant une session sont indépendantes. Si l'utilisateur accède à un RP différent, après avoir été authentifié, la même procédure se reproduit, mais la seule partie visible par l'utilisateur est la renégociation des attributs demandés par le nouveau RP et la décision à prendre par rapport à leur dévoilement. Cependant, le sélecteur d'identité permet de mémoriser des décisions antérieures afin que les futurs accès aux mêmes RPs soient effectués automatiquement.

D.5) Implémentation des composants IAA dans SecFuNet

La Figure 87 illustre les quatre composants principaux de l'IAA dans SecFuNet : la carte à puce, l'IdP SecFuNet avec son environnement virtualisé, le serveur d'authentification et le sélecteur d'identité.

D.5.1) L'IdP SecFuNet et l'OP Proxy

L'IdP SecFuNet se compose d'un serveur OpenID. Ce dernier a été modifié pour pouvoir déléguer à l'AS toutes les fonctionnalités relatives à la gestion des utilisateurs. Ainsi un utilisateur peut facilement transférer ses attributs et réaliser des authentifications basées sur des certificats moyennant son IS. L'implémentation de l'IdP SecFuNet est basée sur le code Open source JOIDS (Java OpenID Server) [79].

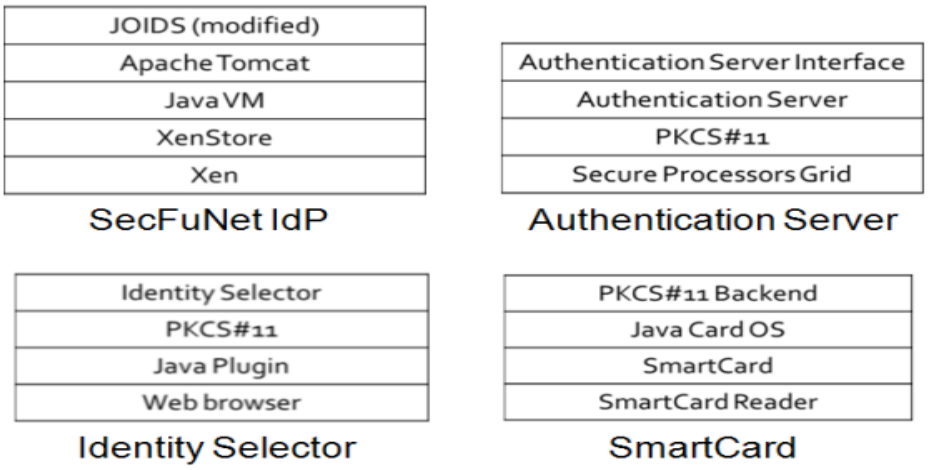

Figure 87 : Les principaux composants de l'IAA dans SecFuNet

Les modules associés à la gestion des comptes utilisateurs ont été retirés car dans le modèle IdP SecFuNet, les comptes utilisateurs sont gérés par l'AS et les attributs sont stockés dans la carte à puce de l'utilisateur. L'AS est maintenu complètement isolé du réseau.

Les OP Proxy s'exécutent dans le serveur d'applications Apache Tomcat 7.0. Les bibliothèques de sécurité de ce dernier sont modifiées pour utiliser notre IdP personnalisé de sorte à ce que les fonctions d'établissement de canal TLS soient déléguées à l'AS intégré dans l'hyperviseur XEN.

Cette intégration de l'AS au même niveau d'exécution que l'hyperviseur XEN lui assure une bonne isolation. Cependant, une zone de mémoire partagée a été allouée pour protéger les échanges entre ce service et les VMs exécutant l'OP Proxy. Cette zone mémoire est composée d'un ensemble de registres. Ces registres sont utilisés par le fournisseur de sécurité personnalisé de Tomcat pour envoyer des requêtes à l'AS, qui à son tour les envoie au serveur d'authentification afin d'établir le canal sécurisé avec la carte à puce de l'utilisateur. La mémoire partagée a été implémentée à l'aide des outils de XenStore [38].

Le XenStore est un espace de noms qui fournit plusieurs registres mémoires pour la communication entre les domaines exécutés dans l'hyperviseur XEN. Cet espace de noms est géré par le domaine 0 qui est responsable du contrôle des accès à ces registres mémoire.

D.5.2) Le Sélecteur d'Identité

La Figure 88 est une capture d'écran de l'interface utilisateur lors d'une demande d'authentification avec un IdP SecFuNet. On y voit l'IS qui est exécuté dans le navigateur de l'utilisateur et qui fournit deux fonctions de base :

- La gestion et l'accès à la carte à puce pour authentifier l'utilisateur au moyen du certificat et pour obtenir les attributs stockés.
- La gestion et le contrôle des attributs grâce à une interface graphique. C'est l'utilisateur qui décide d'autoriser ou pas, le dévoilement de ses attributs au fournisseur de services. Les accès à la carte à puce se font à la fois pour établir un canal TLS et pour récupérer les attributs, via l'implémentation de l'API Java Card 2.1 et SunPKCS#11.

Dans la procédure d'authentification, l'OP Proxy envoie l'applet du sélecteur d'identité à l'utilisateur ainsi que cette page de connexion. L'IS demande à l'utilisateur d'insérer la carte à puce et de saisir le code PIN.

L'IS utilise l'API Java Card 2.1 et SunPKCS#11 pour réaliser une authentification mutuelle avec l'IdP SecFuNet et ensuite établir un canal TLS. L'IS ne peut accéder à la carte pour récupérer les attributs et les envoyer à l'IdP SecFuNet qu'après accord de l'utilisateur. Si ce dernier décide de ne pas les libérer le processus d'authentification sera interrompu.

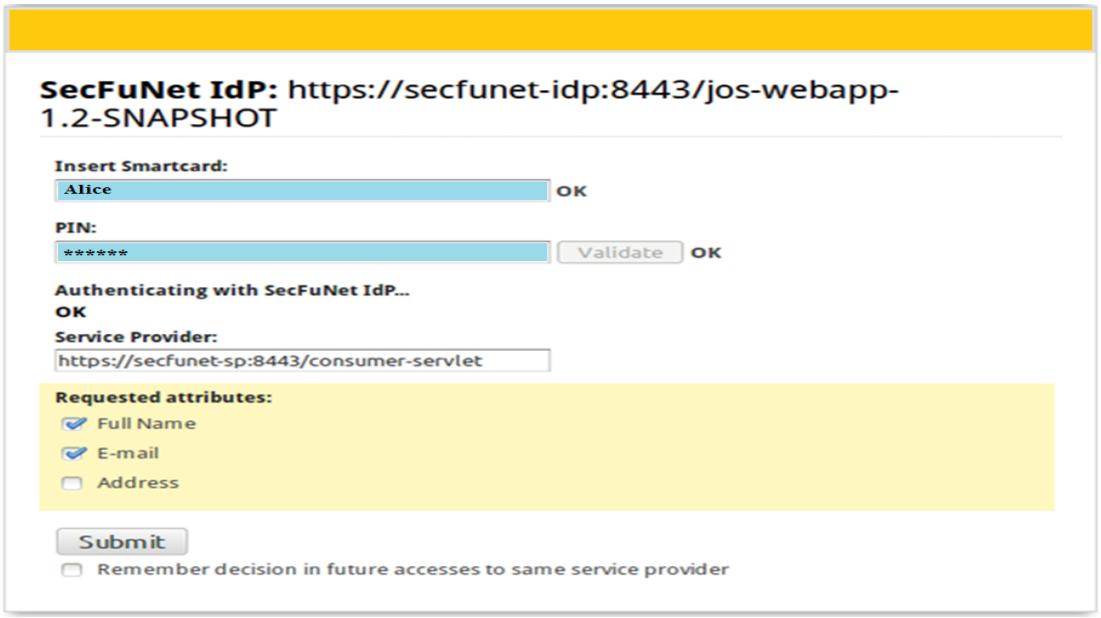

Figure 88 : Une capture d'écran de l'IS

Les attributs sont accessibles via les fonctions de gestion d'objets PKCS#11 (i.e. *C_GetAttributeValue* et *C_SetAttributeValue*) et sont marqués comme SENSITIVE [73], de sorte qu'ils doivent être cryptés par une clé publique approuvée avant d'être extraits. De cette façon, un IS compromis ne sera pas capable de lire les valeurs de la carte à puce. Cela exige de l'IdP SecFuNet de déchiffrer les attributs du serveur d'authentification avant de les relayer vers le RP.

D.5.3) Le jeton cryptographique

Un jeton (i.e. Smart card) est un lecteur de carte, avec une carte à puce qui embarque des applets Java Card qui implémentent le protocole PKCS#11 sous forme de procédures. Celles-ci fonctionnent avec les capacités cryptographiques des cartes à puce et les APIs PKCS#11 utilisées par l'IS.

Ces APIs permettent de stocker dans la carte les attributs, les certificats et donnent accès à des fonctions cryptographiques de la carte à puce. Dans cet actuel prototype, nous utilisons Java Card 2.1 et PKCS#11 mis en œuvre par SafeSign.

D.5.4) Le Fournisseur de Services (RP)

Le fournisseur de services (RP) n'avait pas besoin d'une extension spéciale. Il a été développé en utilisant AJAX (Asynchronous Javascript) et XML. L'utilisation d'AJAX permet plus de contrôle de l'application et de ses échanges de messages avec l'IdP SecFuNet. Les OP Proxy peuvent être remplacés par une application AJAX qui s'exécute sur le navigateur du client et qui peut se substituer à ces derniers.

D.6) Établissement d'un canal TLS en utilisant PKCS#11

Dans cette section, nous décrivons brièvement la mise en œuvre du protocole TLS avec authentification mutuelle entre l'IS et l'IdP SecFuNet en utilisant le module PKCS#11, tout en gardant toutes les clés matérielles sensibles au sein des cartes à puce de l'utilisateur et le serveur d'authentification. La Figure 89 illustre les étapes initiales de la Phase I pour l'établissement d'un canal TLS sécurisé avec authentification mutuelle en utilisant un jeton PKCS#11 des deux côtés.

Cette illustration est basée sur l'implémentation trouvée dans les SunPKCS#11 de la bibliothèque Java. La connexion côté client est initiée par l'IS, qui s'interface avec la carte à puce de l'utilisateur et du côté serveur, la connexion est assurée par l'OP Proxy, qui délègue à l'AS, les échanges avec le serveur d'authentification :

Aux étapes 1 à 6, les informations échangées sont envoyées en clair entre l'IS et l'OP Proxy. Ils échangent des nombres aléatoires, les certificats, et choisissent ensuite des mécanismes et des algorithmes cryptographiques.

À l'étape 7, l'AS invoque le serveur d'authentification pour vérifier le certificat de l'utilisateur (via l'appel *VerifyCertificate*).

À l'étape 8, le sélecteur d'identité génère la clé *PreMasterSecret* (PMS) laquelle est très sensible car elle permet la génération de la clé partagée *Master Secret* (MS). La clé PMS est générée dans la carte à puce par le biais d'un appel à la fonction (*C_GenerateKey*) et cette clé est marquée (SENSITIVE).

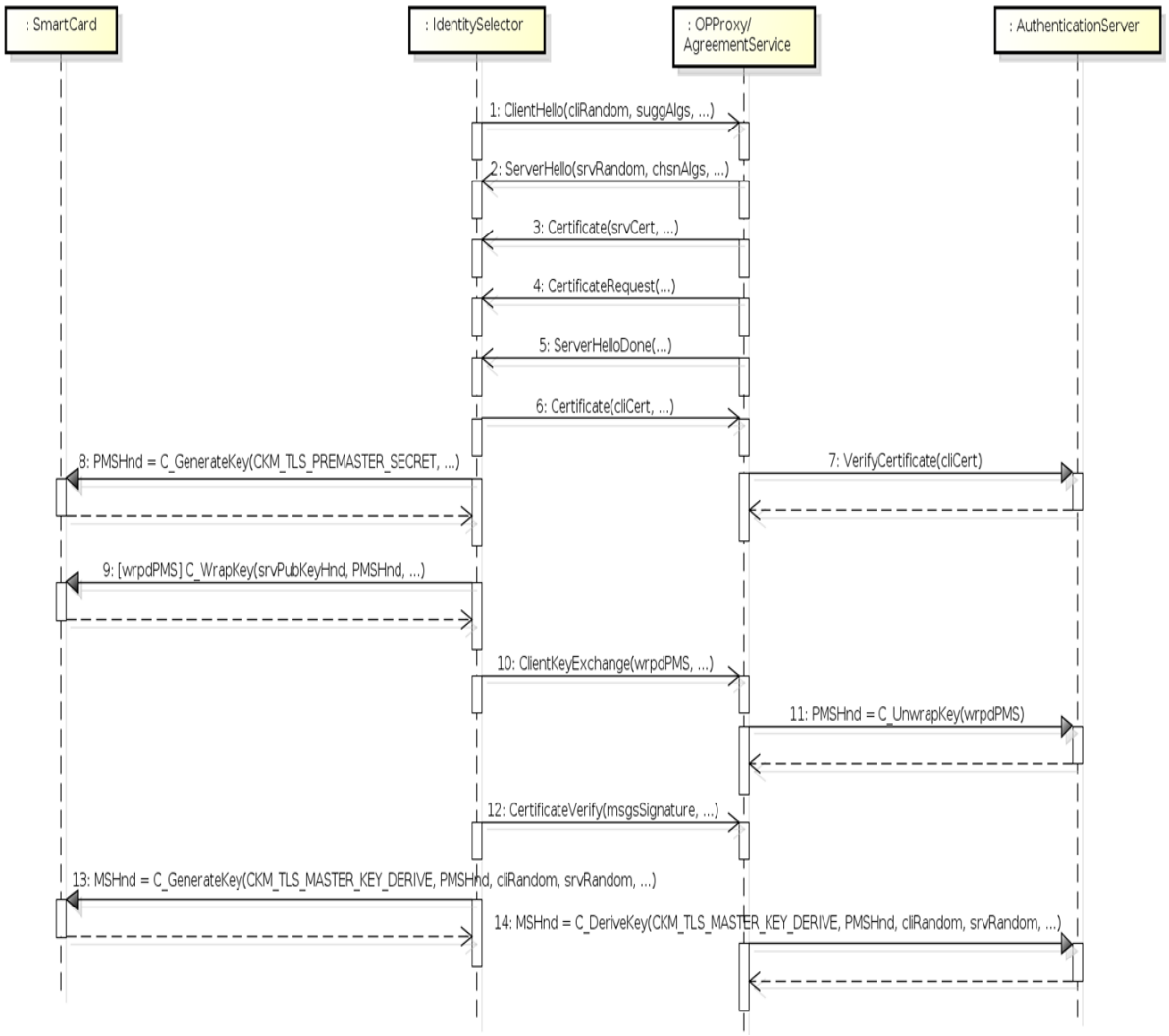

Figure 89 : Phase 1 d'une authentification mutuelle en utilisant PKCS#11

À l'étape 9, la clé PMS est chiffrée par la clé publique de l'IdP SecFuNet. Pour extraire cette clé chiffrée de la carte, la fonction (*C_WrapKey*) est utilisée.

À l'étape 10, cette clé chiffrée, est envoyée à l'OP Proxy et ensuite à l'AS.

Dans l'étape 11, l'AS envoie une requête au serveur d'authentification pour la déchiffrer en utilisant la clé privée de l'IdP SecFuNet. Ce dernier retourne un pointeur vers la clé PMS décryptée.

À l'étape 12, l'IS envoie une preuve de la possession d'un certificat de l'utilisateur en signant un message contenant tous les précédemment échangés. Ce message signé est ensuite envoyé au serveur.

Enfin, aux étapes 13 et 14, l'IS et l'AS calculent la clé *MS* à partir de la clé PMS et des nombres aléatoires échangés (*C_DeriveKey*).

La clé MS est générée à l'intérieur des éléments sécurisés comme une clé secrète non extractible (NOT EXTRACTABLE), elle ne sera pas extraite de l'élément sécurisé. La fonction (*C_DeriveKey*) renvoie un pointeur de sorte que l'IS ou l'AS puissent se référer au MS pendant les prochaines phases du protocole.

De cette façon, il est clair qu'il n'y aura aucune fuite de secret matériel au niveau de l'IS ou de l'OP Proxy/l'AS. La possession de manière continue des éléments sécurisés est donc nécessaire afin de préserver le canal TLS. Cela assure la liaison entre le client et l'OP SecFuNet. Ce canal sécurisé est utilisé plus tard pour transférer les attributs d'utilisateur à partir de la carte à puce à l'IdP SecFuNet.

D.7) Conclusion

Cette annexe présente des solutions de gestion des identités et de contrôle d'attributs centrées sur l'utilisateur, lesquelles reposent sur le standard OpenID et l'intégration de jetons cryptographiques (cartes à puce).

Cette intégration fournit un niveau de sécurité très élevé par rapport à la méthode traditionnelle d'authentification par login/mot de passe, tout en garantissant la confidentialité et le contrôle des attributs par l'utilisateur. Une attention particulière a été portée à l'élaboration des protocoles d'authentification en particulier le respect des spécifications du standard OpenID.

Pour maintenir les propriétés du RP OpenID et pour assurer une interopérabilité avec les IdP SecFuNet, peu de modifications lui ont été apportées. En outre, le stockage des identifiants et des attributs dans la carte à puce permet d'éviter d'une part la mise en place d'un annuaire ou d'une base de données au niveau de l'IdP SecFuNet et d'autre part l'administration des identités des utilisateurs dans un système décentralisé qui pourrait entraîner des coûts d'administration supplémentaires. Les attributs seront signés et stockés uniquement par une entité d'enregistrement de confiance sur la carte à puce et validés par le RP OpenID. L'utilisateur ne pourra pas les modifier bien qu'il a le contrôle sur ses attributs.

Le modèle architectural de l'IdP SecFuNet a deux avantages principaux : l'isolation du serveur d'authentification et l'implémentation de la tolérance aux pannes et/ou aux intrusions au niveau de l'OP Proxy. En cas de compromission d'une VM, elle sera remplacée.

Outre la substitution des OP Proxy défectueux, il est également possible d'utiliser des techniques de réplication [28]. En effet, pour éviter de dégrader les performances de ce

prototype et d'engendrer potentiellement des coûts élevés d'accès continus à la grille d'éléments sécurisés au cours d'une session TLS, une parade a été mise en place pour stocker les clés secrètes provisoires partagées dans les OP Proxy (Bien que ce prototype n'ait pas été conçu initialement pour atteindre de meilleures performances et optimisations).

Enfin, comme perspective futur, ce travail peut être étendu pour mettre en place des protocoles de base pour la gestion des identités fédérée, en établissant des relations de confiance entre les IdP SecFuNet et en réalisant la transposition des assertions émises par un domaine administratif pour accéder aux services d'un autre, ces points constituent des aspects importants d'une infrastructure globale de gestion des identités fédérée.

Acronymes:

AAI	Authentication and Authorization Infrastructure
AES	Advanced Encryption Standard
AID	Application IDentifier
AJAX	Asynchronous Javascript and XML
ANSI	American National Standards Institute
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASCII	American Standard Code for Information Exchange
ASN.1	Abstract Syntax Notation One
CA	Certificate Authority
CAD	Card Acceptance Device
DoS	Denial of Dervice
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIPS	Federal Information Processing Standard
HMAC	Hash-based Message Authentication Code
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
IAM	Identity Access Management
IDP	Identity Provider
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
M2M	Machine to Machine
MAC	Message Authentication Code / Medium Access Control
MITM	Man in the Middle
NAS	Network Access Server
NIST	National Institute of Standards and Technology
PC/SC	Personal Computer / Smart Card
PIN	Personal Identity Number
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
PRF	Pseudo Random Function
PSK	Pre-Shared Key
RADIUS	Remote Authentication Dial In User Service
RAM	Random Access Memory
RC4	Rivest Cipher 4
ROM	Read-Only Memory
SAML	Security Assertion Markup Language
SIM	Subscriber Identity Module
SP	Service Provider
SSL	Secure Socket Layer
SSO	Single Sign On
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USIM	Universal Subscriber Identity Module
WS	Web Services
XML	Extensible Markup Language
XRI	Extensible Resource Identifier.

Table des illustrations

Table des tableaux