ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-035

EDITE ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

Télécom ParisTech

Spécialité Informatique et Réseaux

présentée et soutenue publiquement par

Christophe KIENNERT

le 10 juillet 2012

Elaboration d'un modèle d'identité numérique adapté à la convergence

Directeur de thèse : **Pascal URIEN**
Responsable scientifique : **Ahmed SERHROUCHNI**

Jury

M. David SIMPLOT-RYL, Professeur, Université de Lille 1	Président du Jury
M. Serge CHAUMETTE, Professeur, LaBRI, Université de Bordeaux 1	Rapporteur
M. Didier DONSEZ, Professeur, LIG, Université Joseph Fourier, Grenoble	Rapporteur
M. Bertrand DU CASTEL, Schlumberger Fellow, Austin, Texas	Examinateur
M. Guy PUJOLLE, Professeur, LIP6, Université de Paris 6	Examinateur
M. Bernard COUSIN, Professeur, ISTIC, Université de Rennes 1	Examinateur
M. Pierre PARADINAS, Professeur, CNAM, Paris	Examinateur

T H È S E

Télécom ParisTech
Ecole de l'Institut Télécom – membre de ParisTech
46, rue Barrault – 75634 Paris Cedex 13 – Tél. + 33 (0)1 45 81 77 77 – www.telecom-paristech.fr

Remerciements

Je tiens à remercier ici les personnes qui m'ont permis de mener mes travaux à leur terme, que ce soit par leur encadrement scientifique, leur soutien moral, ou encore leur éclairage extérieur critique et objectif.

Tout d'abord, je remercie vivement le professeur Pascal Urien, qui a encadré mes travaux en s'efforçant de trouver des axes de recherches compatibles avec l'activité de la start-up Ethertrust qui a subventionné cette thèse. Ses nombreuses idées m'ont permis d'acquérir une vision scientifique précise et critique des technologies liées aux réseaux et aux cartes à puce, ainsi que des applications industrielles qu'envisagent d'en tirer les entreprises. Je le remercie également de son soutien tout au long de ces trois années.

Je remercie vivement Ahmed Serhrouchni, Philippe Godlewski et Jean Leneutre, qui m'ont fourni de nombreux conseils visant à surmonter les différents problèmes rencontrés au cours de cette thèse, qu'il s'agisse de difficultés scientifiques, organisationnelles, ou morales.

Je remercie spécialement les professeurs Serge Chaumette et Didier Donsez qui ont accepté de rapporter ma thèse, ainsi que Bertrand Du Castel, dont la contribution à la Java Card est de notoriété mondiale et qui a spécialement accepté de faire le trajet depuis les Etats-Unis, et les professeurs Guy Pujolle, David Simplot-Ryl, Pierre Paradinas et Bernard Cousin, qui ont tous accepté de faire partie du jury lors de ma soutenance.

J'adresse également mes remerciements à Paul Jouguet et Sébastien Kunz-Jacques pour leur regard extérieur et pertinent porté sur mes travaux de thèse ainsi que pour leurs précieux conseils relatifs à la bonne gestion et appréciation des contraintes propres à l'entreprise accueillant ma thèse, conseils qui m'ont permis de travailler aussi efficacement d'un point de vue industriel qu'académique.

A Naoki Iso, j'adresse ces quelques phrases : 交換授業ありがとうございました。社会学に関する会話や翻訳は興味深かったです。そして、日本語や日本の文化については非常に勉強になりました。またいつか会いましょうよ。

Je remercie Véronique Guillo qui m'a permis de poursuivre le piano sur des œuvres de choix, quoiqu'aux dépens de nos voisins d'appartement.

Je salue et remercie mes amis Ben, Bertrand, Bastien, Boris, Rémi, Pascal, Shno, Gervaise et Seb, juste pour le plaisir de les faire apparaître dans une thèse pour laquelle ils n'ont aucun intérêt particulier. J'en ai sans doute oublié, mais c'est tant pis pour eux. Ou tant mieux, c'est selon.

Enfin, je ne remercierai jamais assez mon démon Sylvain, qui par son soutien à tous points de vue m'a permis de faire face aux pires difficultés rencontrées communément au cours de ces trois

années. Sa vision des enjeux, ses connaissances techniques, son recul et son indéfectible appui resteront à jamais ancrés en moi. Et si Dante a vu courir Scarbo dans les cercles des enfers alors même que des Ballades se déclinaient en Variations Sérieuses, c'est parce que nous savons que l'Isle Joyeuse dissimule un Gibet auprès duquel le nain toujours s'affaire. Le croyait-on évanoui ? Soudain il pirouettait, dansant une frénétique Valse de Mephisto lors que résonnaient les Funérailles des condamnés, son rire effroyable résonnait à travers les Cathédrales Engouties, son corps s'élançait dans un ultime Feu d'Artifice – et soudain il s'éteignait.

Résumé

L'évolution des réseaux informatiques, et notamment d'Internet, s'ancre dans l'émergence de paradigmes prépondérants tels que la mobilité et les réseaux sociaux. Cette évolution amène à considérer une réorganisation de la gestion des données circulant au cœur des réseaux. L'accès à des services offrant de la vidéo ou de la voix à la demande depuis des appareils aussi bien fixes que mobiles, tels que les Smartphones, ou encore la perméabilité des informations fournies à des réseaux sociaux conduisent à s'interroger sur la notion d'identité numérique et, de manière sous-jacente, à reconsidérer les concepts de sécurité et de confiance.

En effet, l'absence de norme universelle définissant les caractéristiques d'une identité numérique conduit à une multiplicité de modèles, tels que le couple « identifiant / mot de passe », mais aussi à l'utilisation de modalités biométriques, ou encore de certificats numériques, tel que cela est pratiqué par exemple dans le protocole TLS. L'évolution actuelle de l'Internet amène à poser des modèles d'identité disjoints et peu adaptés au principe de la convergence, qui vise à fusionner des technologies différentes pour obtenir un support technologique capable de gérer plusieurs données différentes, comme cela fut le cas pour la téléphonie sur IP qui a permis, dans les cas où elle a été déployée, de supplanter le réseau téléphonique commuté.

L'affirmation de la différence du traitement de l'identité numérique dans les réseaux mobiles tient au fait que des appareils tels que les Smartphones ne sont pas toujours conçus pour fonctionner avec des modèles fortement sécuritaires, pour des raisons évidentes d'ergonomie. Cela conduit dans la majorité des cas à ramener l'identité numérique au couple « identifiant / mot de passe » le plus simple qui soit. Parallèlement, les réseaux sociaux développent le même type de comportement en refusant toute complexité qui serait de nature à dissuader des utilisateurs de venir faire usage des services proposés. Dans ce contexte, la confiance à accorder à la vérification de l'identité, i.e. l'authentification, est toute relative.

Réduite ainsi à son plus simple appareil, l'identité numérique est pourtant un élément crucial dans l'utilisation des réseaux. A l'heure où les informations personnelles sont des données essentielles aux yeux d'entreprises commerciales ou bien d'individus malhonnêtes, l'accès frauduleux à ces informations, voire l'usurpation d'identité, peuvent conduire à des conséquences désastreuses pour la victime. Par ailleurs, ces modèles « simples » contrastent fortement avec ceux déployés dans des cadres plus formels, qui impliquent l'utilisation d'éléments plus complexes pour mener à bien une authentification. Ces modèles, en l'état, peuvent difficilement se rapprocher, pour des raisons liées à la difficulté de gestion par l'utilisateur, mais aussi à l'interopérabilité.

De plus, le principe de la fédération d'identités, déployé dans des cadres tels que les réseaux sociaux, repose sur la notion d'identifiants uniques. Par conséquent, le vol d'une seule identité numérique peut permettre l'accès à de nombreux services du Web. Dans ce contexte, la promotion

d'un modèle d'identité numérique le plus simple et le moins sécurisé possible doit être évitée à tout prix.

La contribution réalisée dans ce travail de thèse consiste, dans une première partie, à analyser les différents modèles d'identité numérique existants ainsi que les architectures de fédération d'identité, mais également les protocoles déployés pour l'authentification et les problèmes de confiance engendrés par l'absence d'élément sécurisé tel qu'une carte à puce. Dans une deuxième partie, nous proposons, en réponse aux éléments dégagés dans la partie précédente, un modèle d'identité fortement attaché au protocole d'authentification TLS embarqué dans un composant sécurisé, permettant ainsi de fournir les avantages sécuritaires exigibles au cœur des réseaux actuels tout en s'insérant naturellement dans les différents terminaux, qu'ils soient fixes ou mobiles. Ce modèle d'identité sera complété par la proposition d'un serveur permettant une gestion simple de ces identités – sans que l'aspect « gestion d'identité » ne constitue toutefois l'axe principal de la recherche. En revanche, nous montrons que ce serveur d'identités permet sans effort de rendre viable le modèle d'identité numérique présenté dans cette thèse. Nous montrons également comment ce modèle peut se situer à la fois dans la convergence des architectures de fédération d'identité, et également dans la convergence des réseaux fixes et mobiles. Enfin, dans une dernière partie, nous expliciterons plusieurs applications concrètes, testées et validées, de ce modèle d'identité, afin d'en souligner la pertinence dans des cadres d'utilisation pratique extrêmement variés.

Abstract

IT networks evolution, chiefly Internet, roots within the emergence of preeminent paradigms such as mobility and social networks. This development naturally triggers the impulse to reorganize the control of data spreading throughout the whole network. Taking into account access to services such as video or voice on demand coming from terminals which can be fixed or mobile such as smartphones, or also permeability of sensitive information provided to social networks, these factors compel a necessary interrogation about digital identity as a concept. It also intrinsically raises a full-fledged reconsideration of security and trust concepts.

As a matter of fact, the lack of worldwide standard defining specificities for digital identity consequently prompts a multiplicity of frameworks, from the classical “login/ password” couple, to the use of biometrical modalities, or also digital certificates which are massively used along with the TLS protocol. The actual Internet development sets in motion new identity frameworks which are heteroclite and poorly apt to match with convergence principles. Convergence, which aims to merge dissimilar technologies in order to build a technological support able to manage a manifold set of data, was notably successful in the case of VoIP which succeeded to supersede PSTN wherever VoIP has been deployed.

The fact that digital identity needs be handled in a different manner within mobile networks, originates in the way Smartphones have ben initially conceived regarding security. For obvious ergonomic purposes, they have not always been meant to run within high-security infrastructures, thus coercing digital identity to be limited to a simple “login/password” pairing. In addition, social networks adopt the same behavior by refuting any kind of intricacy which would constrain their users and the practice of their services. In this case, the amount of trust which can be placed within identity checking obviously does not weigh more than a penny.

This severely dismantled digital identity is however a critical key if one considers networks and their usage as a whole. At a time where personal information is a valuable and priceless data for business establishments, ill-intentioned characters or overwhelming states, fraudulent access to this data or identity usurpation sometimes lead to disastrous, even deadly, consequences for a victim of those forfeits. Besides, those too basic frameworks are in total discrepancy with those which are deployed within more formal situations (i.e. within companies or administrations) and which are based on more elaborated patterns in order to process a secure authentication. As they stand, those two types of frameworks are unlikely to be linked because the second one is too difficult to apprehend for an average user and an everyday use, and because it does not match with interoperability criterions.

In addition, the key principle of identity federation as it is deployed within social networks is strongly based on the use of a single login. Consequently, stealing one single digital identity lets the thief access numerous services of the Web and sensitive information, from bank account to medical

information. At this point, the promotion of a digital identity paradigm which aims to be as simple as it is unsecure should be avoided at all costs.

The contribution of this thesis project is in line, in a first part, with the analysis of the sundry existing digital identity frameworks as well as the study of current authentication protocols and trust issues raised by the lack of trusted environment such as smartcards. In a second part, as an answer to the concerns suggested in the first part, we will advocate an identity framework strongly bounded to the TLS authentication protocol which needs to be embedded in a secure component, thus providing the mandatory security assets for today's networks while naturally fitting with a varied scope of terminals, be it fixed or mobile. This digital identity framework will then be appended with the design of a server able to conveniently manage identities, while this identity management does not stand as the main focus of our work. That being said, we will demonstrate that this identity server grants the capacity to effortlessly produce a feasible – and achieved – platform materializing the digital identity framework introduced in this thesis project. We will also exhibit how this framework is in adequacy with both identity federation convergence and fixed and mobile networks convergence. In a last part, we will finally exhibit a few practical applications of this identity framework, which have been thoroughly tested and validated, in order to emphasize its relevance throughout multifarious uses cases.

Introduction

L'évolution de l'Internet et la croissance des services Web proposés aux utilisateurs conduisent naturellement à reconsidérer la question de l'identité numérique, ainsi que les problématiques annexes que sont les besoins de sécurité et de confiance dans la gestion de celle-ci. Qu'il s'agisse de la dématérialisation des supports physiques, conduisant naturellement les utilisateurs à réaliser des achats en ligne, de l'accès à des ressources variées telles que des flux audio ou vidéo, ou du développement considérable des réseaux sociaux dans lesquels l'identité même de l'utilisateur est devenue l'élément central du service, l'identité numérique prend une place toujours plus importante au cœur des réseaux. Ces derniers, par ailleurs, se sont largement étendus, par l'intermédiaire des réseaux mobiles et des *Smartphones*, permettant aux utilisateurs d'avoir, à tout moment, accès à leurs services favoris.

Dans ce contexte, la sécurité et la flexibilité de la gestion de l'identité numérique sont des aspects primordiaux du développement actuel du réseau Internet. Pourtant, ceux-ci ne reçoivent que peu les faveurs des géants du Web, notamment parce que les contraintes du développement de la sécurité sont notoirement coûteuses, et souvent peu propices à l'amélioration de l'ergonomie des interfaces utilisateurs. Cette absence de réels efforts dans la normalisation de modèles d'identités numériques répondant à de véritables exigences de sécurité a conduit à une utilisation désormais universelle du modèle « identifiant / mot de passe », dont les défauts de sécurité ne se dénombrent plus. Dans des contextes où l'identité se fédère dans des cercles plus ou moins restreints de sites dits « de confiance », il n'est pas envisageable de perpétuer ce modèle obsolète dont les risques, en terme de sécurité, peuvent facilement amener un attaquant à accéder à plusieurs services en s'attaquant à un unique compte utilisateur. Des solutions alternatives existent, telles que les PKI ou la biométrie, offrant de meilleures garanties de sécurité (dont les limites demeurent réelles), mais aucune ne bénéficie, à l'heure actuelle, d'un déploiement massif auprès des utilisateurs, qui n'ont de fait pas de vision d'un modèle d'identité numérique plus abouti que celui qu'ils utilisent quotidiennement.

Afin de repenser la définition d'un modèle d'identité numérique adapté à la convergence, il convient avant tout d'analyser correctement le contexte actuel. Dans cet esprit, les notions que sont l'identité numérique, la sécurité, la confiance, le principe des modèles symétriques et asymétriques, et la convergence, doivent être clarifiées et explicitées.

Qu'est-ce que l'identité numérique ?

Comme le souligne Louis Jean-Camp[1], l'identité numérique demeure encore aujourd'hui un concept flou, et s'oppose notamment à l'identité « papier » par l'absence de norme universelle instaurant un modèle unique, comme cela est par exemple le cas avec le passeport électronique, au niveau international, et la carte d'identité, au niveau national. Cela conduit naturellement à la conception de nombreux modèles d'identité numérique consistant, dans l'idée, en une

dématérialisation du modèle « papier ». Une des différences principales que cela induit entre l'identité numérique et non numérique, est que la seconde établit par nature une bijection entre l'entité et son identité. Une personne physique ne peut en effet ne posséder qu'une seule carte d'identité ou qu'un seul passeport. Or, dans le modèle numérique, une entité peut posséder plusieurs identités, qui seront nécessairement différenciées par l'identifiant. Chaque identité possède alors un ensemble d'attributs qui la caractérisent, par exemple ses droits d'accès au service considéré. Une entité peut ainsi posséder une identité en tant qu'utilisateur d'un service, et une autre en tant qu'administrateur, mais aussi plusieurs identités utilisateur avec les mêmes droits d'accès – l'intérêt résidant alors dans l'utilisation de pseudonymes.

Toutefois, l'identité numérique est apparue très rapidement dans le monde de l'Internet, avant que ne soient pris en considération les enjeux majeurs liés à la sécurité, tels que nous les connaissons aujourd'hui – et avant, également, d'envisager que ne soit généralisé le concept selon lequel une identité numérique peut être associée à une entité autre qu'une personne humaine ou une machine informatique, ce qui constitue pourtant le principe très actuel de l' « Internet des Objets », où toute machine, quelle que soit sa nature, est susceptible de posséder une adresse IP[2], mais également une identité qui lui permette de s'authentifier. Dès l'origine, l'identité numérique était fermement associée à un compte utilisateur, dont la création reposait sur un simple couple « identifiant / mot de passe ». Si cette idée, très simple, pouvait alors permettre une personnalisation de la navigation par l'utilisation de sessions HTTP[3], les données personnelles sensibles étaient loin d'être aussi présentes sur le Web qu'elles ne le sont actuellement. Pourtant, malgré des enjeux sans cesse croissants concernant l'accès à ces données personnelles, ces modèles extrêmement simples se sont pérennisés.

Sur cet aspect, la comparaison avec les premiers protocoles adoptés pour le déploiement de l'Internet, tels que TCP[4], HTTP, DNS[5], SMTP[6], etc., est assez pertinente. En effet, ces derniers ont été développés avec un niveau de sécurité extrêmement faible, voire inexistant. Pour autant, étant donné que les fonctionnalités qu'ils visaient à implémenter étaient malgré tout assurées, leur déploiement n'a pas été entravé. Ils sont ainsi, encore aujourd'hui, les protocoles phares de l'Internet. La sécurité n'a été pensée et ajoutée qu'après coup, avec des protocoles tels que IPSEC[7], SSL/TLS[8][9], DNSSEC[10], qui peuvent se greffer, de manière facultative, sous les premiers protocoles non sécurisés. A l'heure actuelle, il est pourtant aisé de constater qu'il est très fréquent, ou en tout cas possible, d'utiliser HTTP ou SMTP tels qu'ils ont été conçus au démarrage d'Internet, c'est-à-dire en texte brut, sans chiffrement, et sans authentification.

L'identité numérique semble avoir subi un sort similaire. Une conception hasardeuse à ses débuts, et une absence de normalisation, d'unification des modèles vers un paradigme entièrement repensé quant à l'évaluation des risques d'usurpation d'identité pour un utilisateur, ont conduit à la diffusion du modèle le plus primitif, tout en offrant plusieurs propositions de modèles plus intéressants mais peu déployés.

Sécurité et confiance

Les modèles d'identité actuels sont ainsi, dans une large partie, très peu compatibles avec des exigences de sécurité élevées, et ne se préoccupent généralement pas de la question de la confiance pourtant cruciale dans une étape d'authentification. Cette question doit être séparée de la notion de sécurité. En effet, si l'on peut ainsi faire usage d'un protocole d'authentification sécurisé par des algorithmes cryptographiques éprouvés et certifiés, à quel point pouvons-nous être certains que notre identité est soumise à un serveur qui est bien celui qu'il prétend être ? Les problèmes liés au défaut de confiance sont nombreux et s'incarnent notamment par le principe du *phishing*, qui est finalement une attaque portant sur le fait qu'un utilisateur pense être dans un environnement sécurisé et aura de fait une entière confiance à la fois quant à la sécurisation, notamment cryptographique, du serveur distant (sécurisation qui peut être tout à fait réelle et solide), mais aussi quant à l'identité de ce dernier. Il sera donc naturellement amené à donner des informations sensibles telles que ses coordonnées bancaires, avec les conséquences que l'on connaît.

Si le principe de l'authentification mutuelle se propose de résoudre le problème de la confiance, les solutions déployées actuellement laissent en général de côté certains éléments essentiels pouvant nuire à l'ensemble de la solution. Ainsi, le déploiement de TLS sur des serveurs amène-t-il à authentifier le serveur au client avant que celui-ci ne s'authentifie, souvent par un protocole différent. Ceci est l'exemple le plus frappant d'une solution défaillante, car les problèmes de certificats X509[11] sur les serveurs supportant TLS sont nombreux, et les utilisateurs sont le plus souvent amenés à poursuivre leur navigation quelle que soit la nature du problème de certificat soulevé. De plus, la séparation nette de l'authentification du serveur et de celle du client ne permet pas de considérer une telle solution comme une authentification « mutuelle ».

En sus de cette constatation, vient se greffer la question de l'environnement de confiance, qui concerne la qualité du milieu dans lequel les éléments sensibles sont stockés. Davantage même que cela, elle s'attaque aux fondamentaux des protocoles de sécurité tout entiers. En effet, les différentes opérations réalisées par un algorithme, de même que les différentes variables stockant des résultats intermédiaires ou définitifs, sont, dans leur ensemble, susceptibles d'être espionnées ou modifiées dans le cadre d'attaques internes à l'environnement où s'exécutent les algorithmes. Ainsi, un élément aussi simple que la saisie d'un mot de passe sur un site Web quelconque, puis sa sauvegarde automatique dans le navigateur, permet à un attaquant de récupérer cette donnée par l'intermédiaire d'un *malware* tel qu'un cheval de Troie ou un *Keylogger*, quelle que soit la force du mot de passe choisi par l'utilisateur.

Afin de remédier à ce type d'attaques, un élément sécurisé tel qu'une carte à puce peut servir d'environnement de confiance adéquat, notamment par rapport aux environnements usuels tels que les ordinateurs ou les Smartphones. Ces éléments sont usuellement utilisés pour stocker des éléments sensibles tels que des mots de passe ou des clés privées d'algorithmes de cryptographie asymétriques, mais aussi pour réaliser des calculs cryptographiques sensibles. Plus rarement pousse-

t-on la logique jusqu'à embarquer un protocole entier dans une carte à puce. Pourtant, une telle démarche exploite au plus haut degré les possibilités offertes par un tel « élément de confiance », puisque si les calculs cryptographiques sont certes sensibles par nature, d'autres parties des protocoles de sécurité usuels ne méritent pas moins d'attention. Un exemple de cette démarche, qui constitue un des fondements de ce travail de thèse, et détaillé plus avant dans la suite de ce rapport, est la carte à puce EAP-TLS[12][20], qui consiste à embarquer entièrement le protocole TLS dans un élément sécurisé de type carte à puce.

Dans le cadre d'un travail consistant à définir un nouveau modèle d'identité numérique, il nous a semblé indispensable de tenir compte de ces considérations et de conditionner, de fait, la conception dudit modèle à son association intrinsèque avec un élément de confiance. Autrement dit, un modèle d'identité numérique permettant d'offrir à l'utilisateur des garanties de sécurité mais aussi de confiance ne saurait se passer de cet élément. Quoique ce dernier puisse sembler être un ajout contraignant – les utilisateurs n'ayant que peu l'habitude d'utiliser une carte à puce avec leur ordinateur – les téléphones mobiles et les Smartphones ont en revanche souvent une carte à puce intégrée, dont il serait possible de tirer partie si les opérateurs étaient convaincus du bien fondé de l'idée développée dans ce travail de thèse.

Considérations pratiques et ergonomiques mises à part, l'une des difficultés à surmonter en tenant compte de ce choix dans la conception du modèle d'identité numérique sera la compatibilité avec les modèles de gestion d'identité existants et déjà déployés à grande échelle, notamment la prise en compte des architectures de fédération d'identité ou bien la compatibilité avec les modèles d'authentification à la fois symétriques ou asymétriques.

Modèles symétriques et asymétriques

Les modèles d'identité développés à l'heure actuelle peuvent être classés en deux catégories : les modèles symétriques et les modèles asymétriques. Les premiers consistent à partager préalablement un secret, tel qu'un mot de passe ou une clé symétrique, avec le serveur distant. Ces modèles ont l'avantage de la simplicité, puisque la gestion de tels éléments ne nécessite pas d'infrastructure particulière. En revanche, l'utilisateur doit pouvoir stocker, ou retenir, autant de secrets que de serveurs auprès desquels il s'est enregistré. Par ailleurs, ces modèles sont extrêmement sensibles au stockage d'une banque de secrets sur les serveurs : si une banque est compromise, tous les utilisateurs enregistrés deviennent des victimes potentielles d'une usurpation d'identité.

Les modèles asymétriques consistent à publier un élément non sensible tout en conservant un élément privé, les deux étant mathématiquement liés. C'est le cas par exemple des clés de l'algorithme RSA[13] ou bien de la cryptographie sur courbes elliptiques (ECC)[14], utilisées notamment dans le protocole TLS. Dans ce paradigme, le vol de clés publiques suite à la compromission d'un serveur n'apporte strictement aucune information à l'attaquant, ce qui rend l'usurpation d'identité moins évidente à réaliser. Néanmoins, l'inconvénient majeur de ce modèle est

sa complexité de gestion : des infrastructures lourdes telles que le Private Key Generator (PKG) des IBE[15], ou bien les PKI[11], doivent être déployées et gérées sans faille, ce qui est loin d'être aisé – et loin d'être le cas, comme l'ont montré les problèmes liés aux attaques portées sur les certificats X509[16].

Quels que soient, malgré tout, les défauts et qualités de ces deux modèles, il reste essentiel que la conception d'un modèle d'identité numérique puisse se greffer, dans la mesure du possible, avec chacun d'entre eux. Dans ce travail de thèse, comme il sera expliqué par la suite, les différentes preuves de concept nous ont amené à développer un prototype davantage adapté à un modèle asymétrique. Néanmoins, sur un plan théorique, il reste parfaitement adaptable à une architecture de type symétrique.

Qu'est-ce que la convergence ?

La notion de convergence dans les réseaux informatiques, et plus largement dans les technologies de l'information et de la communication, est associée à toute évolution de technologies visant à traiter de manière uniforme des données distinctes. Les exemples les plus courants concernent les données de type texte, voix, et vidéos, qui peuvent à présent être transportées indifféremment dans les réseaux informatiques, lorsqu'il y a un peu plus d'une dizaine d'années la voix était exclusivement transportée par la radio et le Réseau Téléphonique Commuté (RTC) et la vidéo via la télévision. La voix sur IP[17] ou la télévision sur IP sont ainsi deux exemples de technologies s'inscrivant dans la convergence. Par association, les plateformes les plus récentes (Notebooks, Smartphones, etc.) mettant en œuvre cette notion sont dites convergentes.

Dans une acception plus large de la définition, il paraît possible d'y inclure l'Internet des objets, dont les enjeux consisteront entre autres à intégrer à nos réseaux le traitement de toute donnée émanant de tout type d'objet – donnée qui ne sera peut-être ni texte, ni voix, ni vidéo. Par ailleurs, nous emploierons également ce mot, par abus, dans le cadre de la mise en évidence des similitudes observables à la racine d'outils disparates, comme dans le cas de la gestion des sessions utilisateurs, conduisant à la conception d'un socle commun, et notamment d'un socle de sécurisation, reposant sur la prise en compte de ces similitudes. Bien que l'acception courante de la notion de convergence ne concerne, à l'échelle macroscopique, que l'unification du traitement de types de données différentes, le même phénomène peut en effet tout aussi bien se retrouver à plus petite échelle.

C'est dans l'acception la plus large du terme que nous avons travaillé à la conception d'un modèle d'identité numérique « adapté à la convergence ». En effet, dans son acception usuelle, la convergence regroupe un ensemble de contraintes auxquelles la problématique de l'identité numérique doit répondre sans s'y limiter. L'accès, depuis toute plateforme fixe ou mobile, à des services de voix sur IP, de vidéo à la demande, et autres technologies naturellement associées à la convergence, ne constitue qu'un sous-ensemble de la question de l'accès à des services personnalisés devant être munis de procédures d'authentification pourvues de garanties de confiance

et de sécurité, sans toutefois que ces dernières nuisent aux performances des serveurs par des coûts trop élevés en termes de temps de calcul.

Toutefois, en extrapolant le principe de la convergence, nous avons orienté notre travail de conception afin de nous inscrire dans la mouvance de l'unification de technologies existantes. Ainsi, le modèle d'identité numérique qui a été élaboré est applicable aussi bien à des individus humains qu'à des machines ou des objets, répondant ainsi parfaitement à la question de l'authentification dans le cadre du Machine to Machine (M2M) ou de l'Internet des Objets. De même, l'interopérabilité du modèle avec des architectures de fédération d'identité nous ont conduit à prendre en compte les différents outils de gestion de session inter-domaines – notamment, le jeton OpenID[18] et le jeton SAML[19]. Que ces derniers comportent une preuve sécurisée de la réussite de l'authentification de l'utilisateur est en effet une chose, mais qu'ils puissent n'être générés que lorsque l'authentification est elle-même suffisamment de confiance en est une autre. En définissant un modèle d'identité propre à satisfaire cette condition, la sécurisation d'une architecture de fédération d'identité, quelle qu'elle soit, devient possible, et aisément implémentable.

Ainsi, les enjeux liés à l'identité numérique et l'absence de normalisation de ce concept ont naturellement orienté notre travail de recherche vers la définition d'un modèle en phase avec les normes de l'Internet, tout en possédant suffisamment de flexibilité pour satisfaire les questions liées à la sécurité, la confiance, l'interopérabilité et la convergence.

Contributions de la thèse

La conception d'un modèle d'identité prenant en considération ces différentes constatations doit répondre aux problématiques suivantes :

- Assurer à la fois une garantie de sécurité, l'interopérabilité avec les modèles existants, et l'adaptation à la convergence avec l'Internet des objets, aussi bien dans le contexte de réseaux fixes que de réseaux mobiles.
- Assurer la simplicité et la flexibilité du modèle ainsi conçu afin de le rendre viable au cœur d'applications industrielles diverses, mais également compatible avec les différentes architectures de fédération d'identité.
- Résoudre la question de la confiance dans l'authentification mutuelle d'une manière entièrement satisfaisante.
- Envisager des cas concrets d'application de ce modèle.

Par conséquent, le travail réalisé dans le cadre de cette thèse s'est orienté sur le développement de deux axes constituant les principales contributions présentées dans ce rapport.

Le premier axe a mis en perspective l'analyse des insuffisances perçues dans l'état de l'art, aussi bien sur le plan pratique que théorique, à la lumière des caractéristiques que doit posséder un

nouveau modèle d'identité adapté à la convergence, dans le sens exposé précédemment. Ce travail d'analyse a donné lieu à la naissance d'un modèle d'identité adapté à la carte à puce EAP-TLS, dont la gestion et le cycle de vie ont été intégralement spécifiés. A cet effet, un serveur d'identité a été implémenté afin de fournir les principales fonctionnalités permettant de valider la viabilité du modèle ainsi élaboré. Par ailleurs, l'utilisation de différents composants sécurisés (types de cartes à puce, microcontrôleur sécurisé 8-bits, notamment) a permis de réaliser une comparaison des performances relatives au mode d'authentification déployé, à savoir, dans le cas présent, le protocole EAP-TLS[20] embarqué.

Le deuxième axe a consisté à imaginer et mettre en œuvre ce modèle d'identité dans des cas d'utilisation concrets. Ces applications sont extrêmement diverses et vont du M2M au Cloud Computing, en passant par le prépaiement et les serrures sans contact de type Mifare. Chacune de ces applications a été développée au moins en tant que prototype, et certaines ont fait l'objet de véritables déploiements industriels, pour lesquels des contraintes de performances nous ont été imposées. Ce travail de thèse a en effet été réalisé sous convention CIFRE, et a donc en partie été soumis à un contexte industriel fort.

Organisation du rapport

Le présent rapport du travail de thèse est organisé en trois parties. Les trois premiers chapitres ont pour objectif de présenter l'identité numérique au cœur des différents environnements caractéristiques du réseau Internet actuel. Les deux chapitres suivants visent à présenter notre proposition de solution pour un modèle d'identité adapté à la convergence. Enfin, les trois derniers chapitres s'inscrivent dans la description de l'adaptation de la solution à différentes applications industrielles, concrètes, qui ont été testées et validées.

De manière plus concrète, le chapitre 1 analyse les différentes solutions mises en œuvre, à l'heure actuelle, pour la gestion de l'identité numérique. Cette analyse s'intéresse notamment aux formats d'identité numérique les plus déployés, ainsi qu'aux protocoles réalisant l'authentification des utilisateurs. Au-delà des cas d'usages les plus répandus, qui correspondent à une authentification de type « identifiant / mot de passe » de la part de l'utilisateur, ce chapitre se propose de soulever, de manière plus générale, les problèmes liés à l'authentification dans le cadre de solutions plus fortes d'un point de vue sécuritaire. Ainsi, la biométrie, les OTP, et les technologies du type « cartes PKI » permettent d'offrir un certain nombre de garanties quant à la force de l'authentification. Pour autant, elles comportent également des failles génériques qui, une fois mises en lumière, nous ont permis de dégager des axes fondamentaux dans la conception d'un modèle d'identité numérique plus robuste.

Le chapitre 2 exprime notre souci, au cours de ce travail de thèse, de rendre ce modèle compatible avec les architectures de fédération d'identité, dont le principe est très souvent repris, d'une manière ou d'une autre, par les grands acteurs du Web, notamment dans l'idée de coupler les réseaux sociaux à d'autres services, permettant ainsi d'analyser plus finement et plus exhaustivement

les préférences individuelles des utilisateurs. Dans ce chapitre, les architectures les plus répandues, s'appuyant sur des standards ouverts, seront examinées afin d'en dégager les caractéristiques communes. Ces éléments nous permettront d'orienter la conception de notre modèle d'identité afin de pouvoir l'inscrire sans difficulté dans la mouvance de la fédération d'identité, en assurant l'interopérabilité avec les différentes technologies sur lesquelles elle s'appuie, mais aussi en proposant l'ajout d'une couche de sécurisation commune à chacune de ces architectures.

Le chapitre 3 propose, en guise de synthèse des deux précédents chapitres, une technologie de base qui servira de support au modèle d'identité conçu au cours de ce travail de thèse. Cette technologie, que nous avons baptisée « carte EAP-TLS », apporte notamment une réponse aux failles génériques dégagées au cours du chapitre 1. Elle propose une authentification mutuelle utilisant le protocole TLS embarqué dans un élément de confiance tel qu'une carte à puce. Dès lors, l'identité numérique prend la forme d'un conteneur constitué d'éléments personnels qui sera stocké dans ce même élément de confiance. Ce modèle d'identité numérique sera ainsi conçu comme étant entièrement lié à un élément physique porté en permanence par l'utilisateur, de la même manière qu'une carte bancaire. Le chapitre s'achève sur la présentation d'éléments de performances s'attachant notamment à réaliser une comparaison entre plusieurs composants de nature variée, tels qu'une Javacard, une carte USIM et un microcontrôleur 8-bits.

Le chapitre 4 vise alors à présenter le modèle d'identité numérique proprement dit, ainsi que le serveur d'identité permettant d'en réaliser la gestion. En effet, de même que les modèles d'identité les plus simples n'ont de sens que lorsqu'a été explicitée la manière dont l'utilisateur pouvait en faire usage – via une interface de création de compte, d'espace personnel, et éventuellement de suppression de compte – notre modèle va de pair avec un serveur dédié se proposant de réaliser les mêmes fonctionnalités. Le format de l'identité, la sécurisation mise en place lors de la génération de cette dernière, ainsi que ces fonctionnalités sont décrites en détail dans ce chapitre, et les choix effectués pour l'implémentation du prototype y sont également présentés et justifiés.

Le chapitre 5 présente les aboutissements du prototype réalisé, c'est-à-dire les démonstrations effectives intégrant une architecture SSO de type OpenID, ainsi que les extensions réalisées visant à prouver la flexibilité du modèle. Ainsi, les ajouts visant à pouvoir rendre la gestion de ce modèle d'identité accessible via les Smartphones, aussi bien que des propositions d'architectures alternatives visant des opérateurs de réseaux mobiles y sont présentés.

A partir du chapitre 6, des applications concrètes envisagées pour notre modèle d'identité numérique sont présentées. Dès lors, l'introduction de chacun des chapitres énoncera une problématique que nous proposons de résoudre au fil du chapitre.

Le chapitre 6 s'intéresse au domaine du Cloud Computing. Un prototype réalisé à partir d'une grille de 32 cartes EAP-TLS en mode serveur a permis d'illustrer la possibilité de déléguer à une tierce partie l'authentification associée à notre modèle d'identité. L'accès aux différentes cartes serveurs étant parallèle, une montée en charge des connexions, conjointement avec le nombre de cartes serveurs, nous apparaît théoriquement tout à fait viable. Nous détaillons comment cette architecture

peut s'inscrire dans la fédération d'identité par un exemple de sécurisation de génération et de transport d'un type de jeton de session inter-domaines. Ce chapitre fait ressortir une architecture forte dans laquelle l'authentification est réalisée de carte à carte. Cette idée nous a paru suffisamment intéressante pour en étudier une autre déclinaison, orientée vers le domaine du M2M.

Le chapitre 7 aura ainsi pour objectif d'expliciter une architecture de tunnel sécurisé pour le prépaiement intégrant notre modèle d'identité numérique. Cette architecture s'inscrit dans la mouvance du M2M tout en étant adaptée à l'utilisation d'identités pré-stockées dans les éléments de confiance. Dans le cadre de cette architecture, les identités ont surtout pour objet d'être « client » ou « serveur » dans une connexion EAP-TLS, ce qui conduit à considérer le cas d'une carte possédant les deux identités, dont elle fait usage en fonction du rôle qu'elle assure dans l'architecture à un moment précis. Ce cas d'utilisation représente ainsi une conception alternative de la mise en pratique de notre modèle d'identité, dans laquelle les identités représentent davantage des rôles associés à des machines que de véritables identités associées à des personnes physiques.

Enfin, le chapitre 8 conclut l'exploration des horizons applicatifs réalisée au cours de ce travail de thèse, en proposant une application du concept d'identité EAP-TLS à la gestion sécurisée de clés pour des serrures RFID, par exemple pour le cas de chambres d'hôtel. L'architecture proposée repose sur l'utilisation d'un téléphone Androïd capable d'agir en tant que lecteur de carte sans contact de type Mifare. Nous utilisons ainsi une carte sans contact embarquant le protocole EAP-TLS et pourvue d'une interface Mifare afin de réaliser une connexion sécurisée avec un serveur de clés qui renvoie, sous forme de conteneur sécurisé, une clé spécifique à l'identité de l'utilisateur qui sera chargée dans sa carte.

La thèse s'achève sur une conclusion générale mettant en valeur les contributions de ce travail ainsi que les perspectives de futurs travaux qui peuvent être bâtis sur les différentes pistes explorées et présentées dans ce rapport.

Sommaire

Liste des figures

Liste des tableaux

Partie I : L’identité numérique au cœur de l’Internet

Chapitre 1 : L'identité numérique et les protocoles d'authentification

1.1 Introduction

La problématique de la conception d'un modèle d'identité numérique répondant à des exigences de sécurité permettant de protéger les utilisateurs contre l'usurpation d'identité est très étroitement liée avec celle de l'assurance d'une authentification forte. Cela implique que la définition d'un modèle d'identité numérique n'a de sens que lorsque l'on envisage la compatibilité de ce dernier avec des protocoles d'authentification. De fait, un modèle d'identité tend à s'exprimer en termes de format, tel qu'un certificat X509[11] ou un couple « identifiant / mot de passe », mais en réalité, ce format comporte, en soi, une orientation qui lui permet d'être compatible avec tel protocole, et non avec tel autre. Ainsi, un certificat X509 ne pourra jamais être utilisé dans une authentification n'impliquant pas le protocole TLS[9]. La conséquence directe de cette observation triviale s'exprime par le fait que les failles de sécurité observables dans la gestion de l'identité numérique sont parfois davantage liées au format de l'identité numérique qu'au protocole d'authentification déployé.

En effet, à l'heure actuelle, la faiblesse la plus importante, et la plus évidente également, de l'identité numérique, est celle de la faiblesse des contraintes imposées aux modèles en vogue. Ainsi, le modèle le plus simple, celui de type « identifiant / mot de passe », où chacun des deux éléments est saisi au clavier par l'utilisateur, est soumis par nature à de nombreuses attaques indépendantes des protocoles mis en place pour l'authentification proprement dite.

Ce modèle est riche d'enseignement quant à ce qu'il faut ne pas faire. Nous allons ainsi détailler dans ce chapitre les nombreuses attaques dont il peut être victime, et également nous intéresser aux différentes solutions mises en place pour y remédier. Celles-ci nous permettront de mettre en évidence un aspect récurrent dans la question de l'authentification de l'identité numérique : la volonté de renforcer l'authentification, et ainsi d'éviter au maximum l'usurpation d'identité, amène nécessairement à l'utilisation de protocoles ou de moyens plus élaborés qui viennent se greffer sur l'existant sans le remplacer. Ainsi, les One-Time Passwords (OTP) sont souvent le recours utilisé pour s'assurer que l'utilisateur (déjà authentifié) est bien celui qu'il prétend être, lorsque des opérations sensibles, comme des virements bancaires, sont effectuées. Les organismes font leur possible pour limiter la faiblesse intrinsèque du « mot de passe » utilisateur, tout en se refusant à changer de paradigme.

Pourtant, dans des contextes professionnels, ces paradigmes sont nettement plus variés, car la mise en place d'exigences de sécurité n'est alors plus attachée à l'arbitraire de l'utilisateur, ni à sa peur potentielle de la complexité de l'informatique. Des protocoles intégrant par exemple la biométrie, les cartes à puce, ou encore l'authentification mutuelle TLS sont alors beaucoup plus répandus, et

permettent d'éliminer le principe du mot de passe – entendu au sens où l'utilisateur a choisi lui-même le secret partagé, avec les inconvénients que cela comporte en terme d'entropie. Nous nous intéresserons ainsi aux possibilités offertes par ces différents protocoles d'authentification qui s'attachent à renforcer l'authentification proprement dite sans chercher à combler les failles du modèle d'identité numérique à traiter. Nous tâcherons ainsi de dégager les qualités, mais aussi les faiblesses de ces protocoles afin de conclure ce chapitre, à la lumière des précédentes constatations, par l'élaboration des fondations de ce qui constituera notre modèle d'identité numérique – terme entendu au sens large, c'est-à-dire prenant en compte le format de l'identité, mais aussi les différents contextes d'authentification dans lesquels il pourra être utilisé.

1.2 Les faiblesses du modèle « identifiant / mot de passe »

1.2.1 Contexte

Comme évoqué précédemment, le modèle « identifiant / mot de passe » reste le format d'identité numérique le plus répandu, à l'heure actuelle, parmi les utilisateurs. Dans le contexte de la convergence des flux de données à travers des services, potentiellement payant, de voix ou de vidéo à la demande, mais aussi au cœur de la mouvance de la dématérialisation des biens, l'identité numérique de l'utilisateur reste réduite à son plus simple appareil lorsque l'usurpation de celle-ci peut conduire à de nombreux déboires – fuite d'informations privées sensibles, achats frauduleux, etc. Les grands acteurs de l'Internet, tels que Facebook, Google (et ses déclinaisons), Amazon, mais aussi les banques, ont en commun de privilégier ce format d'identité numérique, et de ne pas vouloir en bouger.

Le protocole d'authentification dans ce cas est extrêmement simple : il s'agit d'une simple requête HTTP, en général non chiffrée, à partir de laquelle le serveur distant vérifie simplement que les données fournies coïncident avec celles enregistrées dans sa base de données. A l'issue de cette vérification, si l'authentification est réussie, un cookie de session est transmis à l'utilisateur afin de « certifier » le statut authentifié de ce dernier. Dans le cas, assez répandu, où le protocole HTTP[3] n'est pas protégé par une préalable authentification TLS à sens unique de la part du serveur, les possibilités d'attaque sont nombreuses. Espionner l'échange suffit ainsi à récupérer les données, ce qui est extrêmement facile à réaliser dans les espaces publics de réseaux sans fil non protégés (ou protégés avec le protocole WEP[21], dont les failles sont désormais bien connues[22]).

1.2.2 Les contre-mesures simples limitant les attaques classiques

1.2.2.1 Mise en place du protocole TLS côté serveur

Une première contre-mesure, évidente, est la mise en place obligatoire du protocole TLS sur toute session HTTP réalisée sur le serveur (la description et l'analyse du protocole TLS figurent dans la section 1.5). Cela n'a aucune incidence sur l'interface utilisateur, et permet à la fois de certifier

l'identité du serveur auprès de l'utilisateur, et de chiffrer les données transmises postérieurement à l'établissement de la session – au prix, toutefois, d'un surcoût pour le serveur en termes de volumes de données échangées et de temps de calcul. Cette contre-mesure, qui ne procure que des avantages d'un point de vue sécuritaire et sans inconvénient vis-à-vis de l'utilisateur, a le mérite d'augmenter la confiance de l'environnement d'authentification, mais reste toutefois largement insuffisante. En effet, elle ne donne aucune garantie sur la force des mots de passe choisis par les utilisateurs, qui possèdent généralement une entropie assez faible (cf. paragraphe suivant). De plus, les attaques par phishing ne sont pas neutralisées pour autant, même si leur efficacité est moindre.

1.2.2.2 Instauration d'une plus basse entropie admissible

Une deuxième contre-mesure, qui peut être ajoutée à la précédente, consiste alors à forcer l'utilisateur à sélectionner un mot de passe jugé « fort », i.e. possédant une entropie suffisamment élevée. L'entropie, telle que définie par Claude Shannon dans le domaine de la théorie de l'information[23], exprime la quantité d'information contenue dans une source d'information (ici une chaîne de caractères), soit, en d'autres termes, le niveau d'incertitude de cette dernière. D'un point de vue mathématique, le calcul de l'entropie associée à une variable aléatoire discrète X est défini par la formule suivante :

$$H(X) = \sum_{x} P(X = x) \log_2 \left(\frac{1}{P(X = x)} \right)$$

Dans le cas d'une répartition équiprobable des événements, en notant *p* la probabilité de réalisation d'un événement, la formule se simplifie ainsi :

$$H(X) = \log_2 \left(\frac{1}{p} \right)$$

Plaçons-nous désormais, comme décrit dans l'appendice A de [24], dans le cas où un événement représente le tirage de caractères parmi un alphabet proposé qui en contient *b*. L'entropie d'une chaîne aléatoire de *l* caractères est alors égale à : $H(X) = \log_2 \left(b^l \right)$

Il est alors possible, pour un serveur, d'obtenir une borne supérieure de l'entropie du mot de passe de l'utilisateur, et avec des considérations de théorie des langages et de linguistique plus avancées, d'obtenir une estimation plus fine de son entropie réelle. En effet, par défaut le serveur connaît *l*, et par une analyse de la chaîne saisie, il est assez simple d'avoir une estimation de *b* (utilisation de minuscules seules, de chiffres, de majuscules et minuscules, de caractères spéciaux, etc.). L'équation précédente correspond, avec ces paramètres, à un mot de passe parfaitement aléatoire, ce qui n'est en général pas le cas. L'entropie réelle est donc nécessairement plus faible.

L'expérience montre alors, à la lumière de ces éléments théoriques, que les jugements de « force » des mots de passe sont généralement fallacieux, car il est souvent considéré comme impossible, et contre-productif, de demander à un utilisateur de composer une chaîne pseudo-aléatoire. Ainsi, dans le cadre de la création d'un compte sur le serveur de Gmail, un mot de passe a été jugé « fort » alors qu'il ne contenait que 9 lettres minuscules, où consonnes et voyelles étaient suffisamment alternées pour qu'il fasse éventuellement partie de la langue anglaise ou française. Cet exemple donne les paramètres *l = 9* et *b = 26* (contre une valeur maximale de 95 pour *b*). Le calcul donne alors une borne supérieure de l'entropie de 42 bits. Cependant, l'utilisation du tableau figurant dans la même annexe A de [24], et la prise en compte d'une distribution fort peu aléatoire des caractères de l'alphabet courant, donne une estimation de l'entropie réelle plus proche de 24 bits, ce qui peut être considéré comme acceptable, pour protéger l'accès à des données peu sensibles, mais non « fort ». Par ailleurs, le serveur de Microsoft permettant de vérifier dynamiquement la force d'un mot de passe[25] l'a jugé « faible ». De fait, une estimation du temps nécessaire pour casser un tel mot de passe, sur un serveur qui ne disposerait pas de contre-mesures supplémentaires, peut être obtenue en imaginant une attaque par force brute. Le temps maximal nécessaire pour retrouver le mot de passe est fonction du nombre de requêtes par secondes envoyées au serveur. Ainsi, à un rythme raisonnable de 1000 requêtes/s, le calcul donne : $t = \frac{2^{24}}{1000} s \approx 17000 s$, soit moins de 5 heures.

De la même manière, le serveur de Gmail considère comme fort un mot de passe constitué d'un prénom suivi d'un chiffre, pour peu que l'ensemble comprenne au moins huit caractères (entropie estimée à moins de 18 bits d'après le tableau précédent), alors que le serveur de Microsoft maintient le jugement « faible ». De fait, les tests de robustesse menés sur les mots de passe voient leurs résultats adaptés en fonction de plusieurs facteurs, notamment le public visé, les conséquences d'une usurpation d'identité, la présence de plusieurs contre-mesures différentes, et également la volonté de ne pas rebuter l'utilisateur. Au bout du compte, il est souvent assez simple de retrouver les mots de passe utilisateurs par une attaque de type dictionnaire, qui consiste à essayer tous les mots de passe à partir d'une liste de mots du dictionnaire et de mots de passe les plus courants. Les mots de passe constitués d'un nom commun ou propre suivis de chiffres, autrement dit la très grande majorité des mots de passe courants, sont particulièrement sensibles à ce type d'attaque. Des logiciels tels que l'Open Source *John the Ripper*[26], permettent d'automatiser cette attaque (contre des systèmes Unix, dans le cas de ce logiciel) et de retrouver assez rapidement la plupart des mots de passe. En moyenne, une durée de quelques minutes est suffisante pour trouver le mot de passe d'un utilisateur peu sensibilisé à ces problématiques.

1.2.2.3 Limitation du nombre de tentatives d'authentification

Une troisième contre-mesure à présent couramment déployée sur les serveurs consiste alors à limiter le nombre de mauvaises tentatives d'authentification. Cette initiative prend trois formes : la première consiste à bloquer l'utilisateur ayant échoué son authentification du côté client, sur la base

de son IP, et non à bloquer le compte utilisateur en lui-même. Cette forme est de loin la plus répandue, mais l'utilisation de proxys ou de réseaux d'anonymisation du type Tor[27] la rend largement inefficace face à un attaquant un peu motivé. La seconde forme consiste à ajouter une requête incitant l'utilisateur à saisir une chaîne de caractères représentée sur une image, afin de vérifier la présence d'un humain dans la tentative d'authentification. Toutefois, les techniques de reconnaissance des formes permettent à des bots de remplir cette tâche, pourvu que les caractères restent reconnaissables. Dans le cas contraire, la contre-mesure pourra être efficace contre l'attaque par force brute ou par dictionnaire, mais aura également toutes les chances de pénaliser l'utilisateur, car la chaîne à saisir est alors généralement passablement illisible. La troisième forme, plus rare, consiste à désactiver temporairement le compte utilisateur jusqu'à ce que la victime ait réussi, d'une autre manière, à prouver son identité. Cette contre-mesure est notamment mise en place sur les sites bancaires, et si elle s'avère assez efficace face à l'attaquant, elle l'est notoirement moins du point de vue de la victime, qui perd l'accès à ses services. Le but de l'attaquant, dans ce contexte, consistera alors plutôt à réaliser des attaques de type « Déni de Service », non pas du point de vue du serveur, mais de celui des utilisateurs dont le compte aura été bloqué. Il ne s'agit donc pas d'une solution efficace.

1.2.3 Le problème de la confiance

De manière plus générale, en supposant que l'entropie du mot de passe des utilisateurs soit suffisamment élevée pour résister aux attaques par dictionnaire et que le serveur a mis en place un système de chiffrement des données échangées, trois éléments persistent quant à la faiblesse de l'authentification :

- L'environnement sur lequel l'utilisateur saisit son mot de passe est généralement un ordinateur ou un Smartphone, sensibles à des attaques de type espionnage (notamment via des chevaux de Troie ou des Keyloggers).

- L'interface Web présentée à l'utilisateur ne certifie pas l'identité du serveur distant, sauf en cas de session TLS préalable. Toutefois, même dans le cas où cette dernière a lieu, l'utilisation d'un faux certificat de la part du serveur conduit à une alerte sur le navigateur client, que bien souvent l'utilisateur s'empresse de négliger – les problèmes de gestion de certificat sur des sites de confiance étant monnaie courante. Des attaques de type Phising sont alors possibles afin de récupérer les données soumises par l'utilisateur.

- A ces deux failles s'en ajoute une troisième, qui est celle de la sensibilité à une attaque de type Man In The Middle (MITM), qui sera davantage détaillée dans la section 1.3.5. Dans ce contexte toutefois, il ne sera souvent pas utile pour un attaquant de déployer ce type d'attaque quand les autres failles présentes demandent des efforts moins considérables pour aboutir au même succès.

Ces trois éléments amènent naturellement à poser la question de la **confiance**. En effet, dans un tel contexte, l'usurpation d'identité n'a rien de bien surprenant. Malgré les tentatives de pallier aux problèmes évoqués ci-dessus, et malgré la sécurisation du lien entre client et serveur, la validité de l'authentification de l'utilisateur demeure, à raison, considérée comme douteuse.

C'est pour cette raison que les contextes les plus sensibles sont à présent dotés de moyen de s'assurer de l'identité de l'utilisateur authentifié – autrement dit, de ré-authentifier un utilisateur déjà authentifié. Les systèmes bancaires, par exemple, fondent la sécurisation des paiements en ligne ou des virements bancaires par l'utilisation d'un OTP (voire de deux), qui vise à augmenter la confiance que l'on peut accorder à l'identité de l'utilisateur ayant su s'authentifier avec succès, mais ne s'attache pas à résoudre la question de la confiance d'un point de vue théorique.

Cela s'inscrit dans la logique de l'**authentification à deux facteurs**, dont le but est de demander à l'utilisateur deux moyens différents de prouver son identité. Parmi ces moyens, on distingue en général trois catégories : ce que l'utilisateur sait, ce que l'utilisateur possède, et ce que l'utilisateur est. Le mot de passe classique entre dans la première catégorie, tandis que l'OTP tend à être plutôt considéré comme faisant partie de la seconde, notamment dans son déploiement par jetons hardware. La troisième catégorie est quant à elle représentée par la biométrie, dont l'intérêt est détaillé à la section 1.4. L'analyse de l'OTP est quant à elle proposée dans la section suivante, et en montre les limites génériques, i.e. indépendantes des choix d'implémentation et de mise en œuvre.

1.3 Le One Time Password

Comme son nom l'indique, le One Time Password consiste à soumettre à l'utilisateur un mot de passe valable une seule fois, à un moment précis. Contrairement au cas précédent, l'utilisateur ne choisit donc pas le mot de passe, qui est en général une chaîne pseudo-aléatoire d'une longueur très variable selon les implémentations. En effet, Le fait que le mot de passe n'ait de validité qu'à un instant donné, sans droit à l'erreur, limite par nature toute attaque par force brute ou par dictionnaire. Par conséquent, il n'est pas rare qu'un OTP soit une suite de quatre ou cinq caractères qui peuvent être uniquement des chiffres. En dehors d'être unique, un OTP se doit également d'être non prédictible : la connaissance d'OTP antérieurs ne doit pas permettre à un attaquant de déduire les OTP futurs. Dans cette section, nous décrivons les implémentations les plus courantes de l'OTP, et nous montrerons que, quelle que soit son incarnation, l'OTP n'est finalement qu'une déclinaison plus élaborée du modèle d'identité « identifiant / mot de passe » sur laquelle l'attaque par Phishing devient beaucoup moins efficace, mais qui reste sensible à une attaque de type MITM.

1.3.1 Algorithmes de calcul des OTP

Les algorithmes permettant de garantir le caractère aléatoire et non prédictible des OTP sont assez variés. Comme il est rappelé dans [28], la première idée de calcul des OTP a été développée par Leslie Lamport dans les années 1980, et utilise une fonction à sens unique *f*, par exemple une fonction de hachage. Le principe proposé, implémenté par exemple dans le système S/KEY[29], est l'itération successive, *n* fois, de cette fonction *f* sur une graine *s*, qui constitue le secret originel créé du côté de l'utilisateur et qui n'est jamais transmis au serveur. La connaissance de *s* par un attaquant extérieur suffit en effet à compromettre la sécurité de l'ensemble. Une fois les itérations effectuées, le secret *s* est éliminé, et $f^n(s)$ est transmis au serveur pour référence. La liste $\left\{f^{n-1}(s), f^{n-2}(s),..., f(s)\right\}$ constitue alors une liste ordonnée d'OTP pour l'utilisateur. L'envoi de $f^{n-1}(s)$ au serveur permet en effet une authentification de l'utilisateur, puisque le serveur peut comparer, après application de *f*, la valeur reçue avec celle stockée pour référence. A la suite de cette authentification, le serveur utilise à présent $f^{n-1}(s)$ pour référence, et l'utilisateur devra alors s'authentifier avec $f^{n-2}(s)$, et ainsi de suite. Le fait que *f* soit à sens unique garantit l'impossibilité pour un attaquant de trouver la liste des OTP de l'utilisateur à partir d'une écoute passive.

Plusieurs alternatives à cette implémentation ont vu le jour. Parmi celles-ci, on peut citer le HMAC-based OTP[30], largement déployé dans le consortium OATH[32].

Une implémentation toute autre a vu le jour, notamment via des jetons hardware, avec une orientation du calcul sur la synchronisation horaire du jeton avec le serveur d'authentification. Chaque jeton est initié avec une graine *s* différente, et génère, à la demande de l'utilisateur, un nombre pseudo-aléatoire dépendant de l'heure de la demande et qui reste valable pendant une durée souvent très courte (de quelques minutes à quelques dizaines de secondes). Une normalisation IETF de cette implémentation, intitulée TOTP[31] a par ailleurs récemment vu le jour.

1.3.2 Transmission de l'OTP par courrier électronique et SMS

D'un contexte à l'autre, les déploiements d'un système d'authentification par OTP sont très variables. Le point commun entre les différentes versions existantes consiste à trouver un moyen de *transmettre* une donnée, de manière certaine, à l'utilisateur. Une première approche consiste à utiliser une donnée numérique propre à l'utilisateur, notamment son adresse mail ou son numéro de téléphone, comme moyen de lui transmettre des données – dans un cas par courrier électronique, dans l'autre cas par SMS. Par exemple, dans les systèmes bancaires, lors d'un paiement en ligne, l'utilisateur n'est, par défaut, pas authentifié auprès de sa banque. Celle-ci utilise alors en général l'adresse mail fournie par l'utilisateur, ou bien son numéro de téléphone, ou bien les deux à la fois, qui

servent de point de contact avec ce dernier, et permettent de lui envoyer un mail et/ou un SMS contenant l'OTP.

Le cas de la transmission par mail se heurte, par ricochet, au problème de l'authentification au serveur de mail, protégé par le classique « identifiant / mot de passe ». Celui de la transmission par SMS permet d'éviter cet écueil et bénéficie également, éventuellement, d'un chiffrement de type symétrique de type A5/x lors du transport jusqu'au téléphone de l'utilisateur – bien que les algorithmes A5/1 et A5/2, utilisés dans les réseaux GSM, soient depuis longtemps considérés comme cassés[33].

1.3.3 Les OTP non numériques

Une orientation différente de la conception de l'OTP conduit à établir préalablement le lien entre l'utilisateur et un élément physique qui sera générateur d'OTP. Cet élément physique peut être non numérique, c'est-à-dire sous forme papier, qui contient alors une liste d'OTP statiques. Lors d'une procédure d'authentification, il est demandé à l'utilisateur de saisir l'un de ces OTP, qui ne sera plus jamais demandé par la suite. Ce système est mis en pratique dans certaines banques, qui envoient une carte d'OTP statiques à l'utilisateur sur recommandé, c'est-à-dire sur la fois de son adresse postale. Ces OTP peuvent être demandés lors d'achats en ligne ou de virements bancaires.

Le principal défaut de ce système réside dans son format même. En effet, les OTP peuvent être visibles pour quiconque est proche de l'utilisateur, et la nécessité de posséder en permanence, sur soi, la carte d'OTP, en fait une cible facile pour un vol matériel. Ainsi, dans le cas des systèmes bancaires, il est très probable que cette carte sera conservée conjointement avec la carte de crédit – le vol de l'une impliquant le vol de l'autre avec une probabilité non négligeable. On peut considérer ce système comme une version statique et non numérique d'un système plus élaboré reposant sur un composant hardware ou, dans une version équivalente, sur un composant software dédié, installé sur un équipement mobile. Ce sont ces derniers qui vont à présent retenir notre attention.

1.3.4 La génération d'OTP via jeton ou software dédié

Comme le montrent les nombreuses déclinaisons du principe de l'OTP, les différents algorithmes de calcul ont vite été adaptés dans une perspective de mise en pratique software ou hardware qui permettrait à l'utilisateur de générer ses OTP directement sur un équipement lui appartenant, notamment un équipement portable. Cette approche élimine le problème du transport de l'OTP, puisqu'il ne quitte pas son environnement de création.

L'approche hardware repose sur des jetons dédiés, par exemple *RSA SecureID*, que l'utilisateur peut en permanence porter sur lui. Toutefois, avec le développement des environnements mobiles, il est possible de fournir exactement les mêmes fonctionnalités à partir d'une version software, par exemple embarquée sur un Smartphone.

1.3.5 La vulnérabilité des OTP face à des attaques de type MITM

Indépendamment des différents algorithmes existants, ou des modes de transmission des OTP à l'utilisateur, le principal écueil subsistant dans ce modèle, de manière générique, est la possibilité d'une attaque de type MITM.

Le principe de cette attaque consiste, pour l'attaquant, à se faire passer pour le serveur auprès de la victime, et pour l'utilisateur auprès du serveur, de manière à donner aux deux parties l'impression de communiquer naturellement de l'une à l'autre. Pour cela, il intercepte de manière active ou passive, selon le contexte, les communications qui transitent entre les deux parties. La méthode la plus répandue pour mettre en place une attaque de type MITM consiste, sur un réseau local, à réaliser une attaque *ARP Spoofing*, qui consiste à attaquer le protocole ARP[34] afin d'induire une mauvaise association entre l'adresse physique MAC, et l'adresse logique IP d'une machine. De cette manière, la victime enverra ses messages à l'adresse physique de l'attaquant, qui assure alors un rôle de *proxy* (entité intermédiaire) avec la véritable machine destination. Ainsi, dans le cas de la récupération d'un mot de passe classique, l'attaquant reçoit les données de la victime qui croit avoir affaire au serveur, et l'attaquant n'a alors plus qu'à retransmettre telles quelles ces données au véritable serveur. En retour, il peut retransmettre à la victime le jeton de session le stipulant comme étant authentifié, afin de ne pas se faire repérer. L'attaque générique est représentée sur la figure 1.

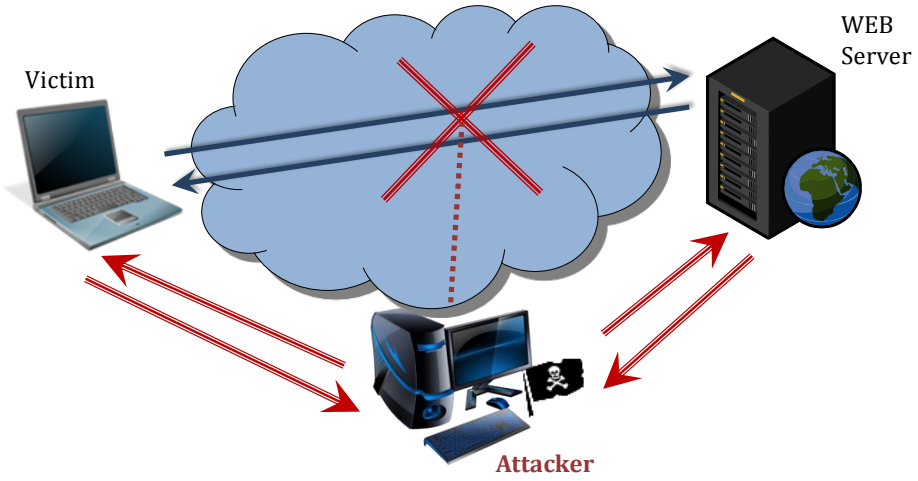

Figure 1 : Attaque Man-In-The-Middle

Dans le cas d'un mot de passe classique, une attaque par Phishing est souvent plus simple à réaliser. En revanche, l'intérêt du MITM est nettement plus fort dans le cas des OTP. Dans ce contexte, en effet, récupérer un mot de passe doit absolument aboutir à pouvoir en faire usage immédiatement, sans quoi cette donnée devient sans intérêt. De la même manière que l'attaquant

peut récupérer un mot de passe classique lors d'une attaque MITM, il peut tout aussi bien récupérer un OTP afin de le réémettre tel quel au serveur en continuant de se faire passer pour la victime.

Il est important de noter que la courte fenêtre temporelle liée à l'utilisation des OTP ne nuit pas au fonctionnement de l'attaque MITM, puisque celle-ci peut se dérouler en temps réel. Par ailleurs, la tentative de connexion à un serveur en HTTPS[35] peut être un obstacle au déroulement de l'attaque puisque, en toute logique, la victime s'attend à recevoir la preuve de l'identité du serveur via son certificat X509. L'attaquant est a *priori* incapable de faire cela. En revanche, il existe une faille qu'il lui est possible d'exploiter.

L'idée consiste à empêcher la victime de réaliser des liens HTTPS avec le serveur, c'est-à-dire que toute requête de type HTTPS sera remplacée par une requête HTTP sur intervention de l'attaquant. Pour cela, l'attaquant réalise une attaque MITM à partir de laquelle il intercepte la première requête vers le serveur sécurisé. Il profite en réalité du fait que cette première requête est très rarement du HTTPS, car la victime ne le saisit pas explicitement dans l'URL à joindre. L'attaquant, recevant cette requête, engage alors une requête HTTPS vers le serveur, reçoit en réponse la page HTML, de laquelle il ôte tous les liens HTTPS pour les remplacer par du HTTP. De cette manière, la victime communique ses données en clair à l'attaquant, tandis que ce dernier poursuit l'échange chiffré avec le serveur. Du point de vue de la victime, seules les notifications discrètes de sécurité des échanges parfois présentes dans les navigateurs (coloration de la barre d'URL, etc.) seront manquantes, ce qui n'a aucun impact auprès d'un utilisateur non averti. Cette faille a été publiée par M. Marlinspike en 2009 [36]. L'auteur a également développé un outil nommé SSLstrip [37] afin de fournir une preuve de concept de cette faille. Son fonctionnement schématique peut être représenté par la figure 2.

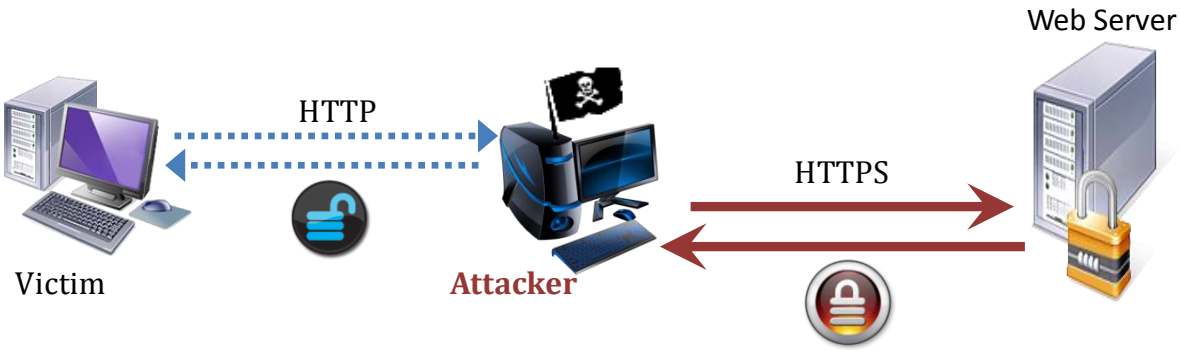

Figure 2 : Attaque MITM sur le protocole HTTPS

Nous voyons ainsi que l'OTP contient une faiblesse dans son principe même : celui d'utiliser un mot de passe, fût-il unique et sécurisé, dans un environnement dépourvu de confiance. En effet, l'utilisateur n'a pas de garanties quant à la qualité de la connexion établie avec le serveur, ni, finalement, quant à son identité, puisqu'un attaquant est capable de maîtriser les requêtes en provenance de l'utilisateur. Il faudrait, pour remédier à cela, stocker le mot de passe (usuel ou OTP) dans un environnement de confiance (comme une carte à puce), mais également pouvoir assurer

l'échec de la connexion avec le serveur dans le cas où l'échange TLS ne s'est pas déroulé normalement. Une solution à cela est de s'orienter sur une authentification mutuelle, dans laquelle l'échange TLS implique également l'authentification de l'utilisateur, et de s'assurer que l'utilisateur exécute le protocole TLS dans un environnement, lui aussi, de confiance, afin d'empêcher l'action de *malware* sur l'environnement utilisateur et sur le déroulement de la session TLS mutuelle. Cette idée a été mise en œuvre dans le cadre de cette thèse par l'utilisation d'une carte à puce EAP-TLS[12], dont le fonctionnement détaillé figure au chapitre 3.

Dans l'idée de renforcer l'authentification, il faudrait donc s'éloigner le plus possible de l'idée de mot de passe, c'est-à-dire également changer de modèle d'identité numérique. Dans la mouvance de l'authentification à deux facteurs (qui conserve, en général, le mot de passe comme premier facteur…), l'utilisation de la biométrie en tant que preuve de ce que « la personne est » s'est largement développée. Nous examinons désormais ce mode d'authentification.

1.4 La biométrie

1.4.1 Principe

La biométrie a pour objet d'authentifier un individu à partir d'une ou de plusieurs de ses caractéristiques physiques ou comportementales. Les modalités biométriques pouvant être mises en œuvre dans un protocole d'authentification sont multiples : analyse des empreintes digitales, de la géométrie de la main, scanner de la rétine ou de l'iris, analyse de la signature manuscrite, reconnaissance faciale, etc. L'identité d'une personne peut alors être vérifiée par comparaison de la modalité relevée avec un modèle figurant dans une base de données consultée par le serveur d'authentification. Deux politiques peuvent être mises en œuvre à cette fin :

- La *vérification* d'une modalité nominative, c'est-à-dire sur un seul échantillon de la base (comparaison individuelle 1 : 1), dont le principe est illustré sur la figure 3, extraite de la norme X9.84-2001[38].

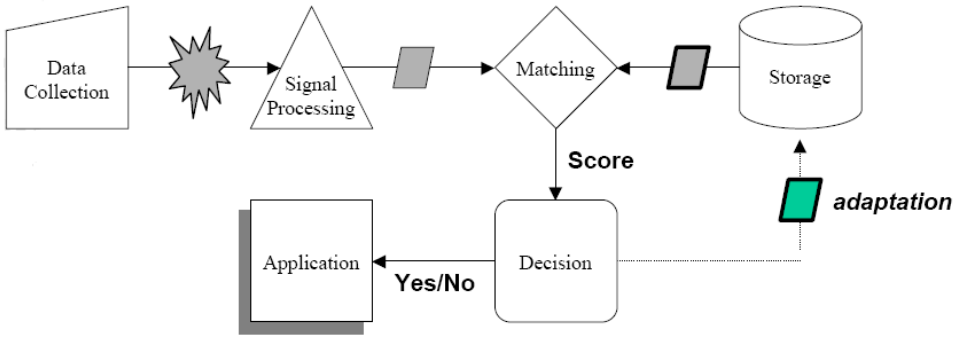

Figure 3 : Vérification biométrique

- L'*identification* d'une modalité anonyme, testée sur tous les échantillons (comparaison collective 1 : N), illustrée sur la figure 4, à nouveau extraite de la norme X9.84-2001.

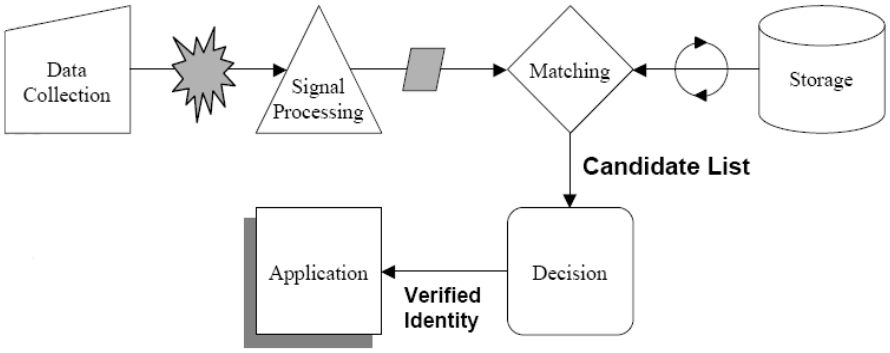

Figure 4 : Identification biométrique

Dans les deux cas, si la concordance entre la modalité et l'échantillon est jugée suffisante, l'identité de l'utilisateur est considérée comme vérifiée. L'évaluation du degré de concordance est réalisée par une analyse mathématique qui définit un seuil d'acceptation au-dessus duquel l'identité est vérifiée. De nombreux organismes ont œuvré à la standardisation des données biométriques, parmi lesquels, outre l'ISO et l'IEC, nous pouvons citer le NIST, qui a mis en place le standard CBEFF (Common Biometric Exchange Formats Framework) définissant le format numérique d'échange de données biométriques [39].

Par construction, l'authentification par la biométrie n'existe que dans un modèle symétrique. La différence essentielle avec la notion de mot de passe provient du caractère personnel, non secret, non modifiable et non imitable de la donnée. Le fait même que la biométrie s'appuie sur une caractéristique propre à l'individu rend le terme générique « authentification » impropre. En réalité, si un terme générique doit être employé, il s'agit d'*identification* : l'individu déclare *qui il est* par le moyen d'une de ses modalités biométriques, ce qui lui permet par ailleurs d'éviter d'avoir à mémoriser une information – ce qui reste l'un des principaux écueils de l'authentification par mot de passe. De plus, les caractéristiques fondant la biométrie garantissent une association naturelle entre l'information fournie (la modalité), et l'identité de l'utilisateur. De ce point de vue, dans un contexte administratif adapté (typiquement, authentification hautement sécurisée de personnel dans une infrastructure), et avec la garantie de disposer d'un lecteur biométrique de confiance (cf. section 1.4.2.3), un protocole s'appuyant sur EAP-TTLS[40] et utilisant la biométrie dans la seconde phase d'authentification, avait été proposé en tant que *draft* à l'IETF [41]. Toutefois, malgré les attraits que la biométrie possède, de nombreuses limitations existent, et beaucoup d'attaques sont réalisables, ce qui amène à penser que la définition d'un modèle d'identité numérique reposant sur ce principe relève d'une gageure.

1.4.2 Limitations et faiblesses

La principale limitation de la biométrie provient de ce qui constitue sa clé de voûte : le caractère non modifiable et non imitable des modalités. Bien que tous les algorithmes de comparaison des modalités biométriques reposent sur ce principe, il faut en relativiser les affirmations.

1.4.2.1 Evolution des modalités biométriques

Les modalités biométriques sont une partie intégrante du corps ou du comportement humain, ce qui les rend, *ipso facto*, soumises à l'évolution du temps et aux aléas de la vie – autrement dit, au changement. Une empreinte digitale peut se détériorer, l'empreinte vocale peut varier selon l'état de santé de l'individu, le visage peut évoluer avec le temps, etc. Par conséquent, il peut se produire un refus de la modalité présentée alors que l'identité de l'individu est la bonne. De plus, dans le cas où l'individu subit des sévices corporels importants (section d'un doigt ou d'une main, décollement de la rétine, globe oculaire fendu, cordes vocales arrachées, etc.), il lui est impossible de remplacer ces éléments d'identification. Il en est de même dans le cas où un attaquant parvient à imiter la modalité d'un individu : contrairement au mot de passe qui peut être changé à volonté, la modalité biométrique mise en cause doit être remplacée par une autre existante – dans la limite, donc, de dix modalités pour les doigts, deux pour les yeux, et une pour la voix, la graphie, le visage, etc.

Il est difficile, dans ces conditions, d'imaginer un déploiement à grande échelle d'un modèle d'identité numérique qui puisse gérer correctement cette limitation conséquente.

1.4.2.2 Imitation des modalités biométriques

Le paragraphe précédent faisait usage d'un exemple impliquant l'imitation d'une modalité biométrique (*spoofing*), contredisant ainsi l'un des principes de base de la biométrie. De fait, même si il est en pratique impossible de modifier sa physionomie, ou son comportement, afin d'en faire coïncider des extraits avec ceux d'un autre individu, il reste possible d'en réaliser une imitation à partir d'un échantillon qui aura auparavant été prélevé par l'attaquant [42]. Ce prélèvement peut être indirect (prélèvement d'empreintes digitales laissées par la victime, enregistrement de la voix, photographie du visage, etc.), ou direct (section du doigt de la victime, prélèvement d'un œil, etc.).

Pour parer à cela, les lecteurs biométriques mettent en place un ensemble de contre-mesures, l'une des plus importantes étant le *liveness detection*, qui vise à s'assurer du lien réel entre la modalité présentée et l'individu à identifier. Dans le cas des empreintes digitales, c'est cela, par exemple, qui permet la détection de prothèses, de faux doigts, ou de doigts sectionnés (via une mesure de la pression sanguine ou un test de la présence de transpiration naturelle, par exemple). Un état de l'art du *liveness detection* est présenté dans [43].

Par conséquent, l'authentification par la biométrie relève à nouveau de la question de la confiance, qui est ici celle que l'on peut accorder au lecteur. Si ce dernier n'implémente pas suffisamment de contre-mesures sécuritaires, les failles énoncées ci-dessus resteront exploitables, et il restera nécessairement des doutes, de la même manière que lors d'une authentification par mot de passe, quant à l'utilisateur qui aura été authentifié par le serveur.

1.4.2.3 Le problème du passage à l'échelle

Dans la perspective du déploiement d'un nouveau modèle d'identité, il est indispensable de prendre en compte le problème du passage à l'échelle. Si les utilisateurs doivent, désormais, s'authentifier par l'intermédiaire de leurs modalités biométriques, il est nécessaire que chacun dispose des outils nécessaires pour s'authentifier. Autrement dit, deux solutions sont envisageables :

- Intégrer un lecteur biométrique de confiance à tout appareil informatique de type ordinateur ou Smartphone
- Stocker les modalités biométriques de l'utilisateur dans un élément sécurisé de type carte à puce.

La première solution implique un investissement financier très fort, y compris pour l'utilisateur, puisqu'elle consiste à intégrer des lecteurs certifiés par des organismes de standardisation sur l'ensemble des équipements informatiques produits, mais aussi, éventuellement, à proposer indépendamment ces lecteurs à l'achat pour les utilisateurs. Il faudrait par ailleurs s'assurer qu'un lecteur non certifié ne puisse donner lieu à authentification, car l'attaquant recherchera alors le matériel le plus faible afin de pouvoir pratiquer des attaques par *spoofing*. Les obstacles à l'établissement d'un tel modèle auprès des utilisateurs sont donc nombreux.

Quant à la deuxième solution, elle présente le double avantage de supprimer la nécessité du lecteur biométrique et de stocker les modalités dans un environnement de confiance. Les problèmes qui se posent alors sont tout autres. Premièrement, il est nécessaire d'enregistrer les modalités biométriques désirées dans la carte, ce qui implique l'utilisation d'un lecteur biométrique certifié, comme ci-dessus. A supposer que cet enregistrement puisse être réalisé seulement auprès de l'administration (permettant d'assurer le lien entre le possesseur de la carte et les modalités biométriques enregistrées), cela implique des procédures coûteuses et fastidieuses à mettre en place. Deuxièmement, le seul moyen de détecter la présence de l'utilisateur réside dans le code PIN permettant l'activation de la carte à puce. Cela ne permet donc pas de réaliser du *liveness detection*, par exemple. Enfin, cette solution ne permet pas aux serveurs de modifier leurs exigences en termes d'authentification. S'il arrivait que les modalités à présenter ne correspondent pas à celles qui ont été enregistrées dans la carte, cela poserait les mêmes problèmes que ceux évoqués précédemment.

Ainsi, la biométrie atteint rapidement ses limites dans un contexte d'authentification très large, notamment sur le Web. Outre ses failles intrinsèques, elle est extrêmement lourde et coûteuse à

déployer, et ne peut donc faire l'objet d'une base pour la conception d'un modèle d'identité numérique qui résiste au passage à l'échelle. La biométrie fonctionne dans des cadres très restreints et des contextes bien définis, mais ne peut en aucun cas servir de socle à l'authentification au cœur de l'Internet. Si l'on ajoute à cela les nombreux problèmes éthiques soulevés par l'automatisation des prélèvements de données biométriques, cette piste ne nous semble pas raisonnable à exploiter.

La conception d'un modèle d'identité numérique doit donc éviter de s'appuyer sur un principe de mot de passe utilisateur, ce qui n'apporterait aucune avancée considérable par rapport à l'existant, et ne peut non plus s'appuyer sur la biométrie. En revanche, un autre modèle, largement déployé, mérite que l'on s'y attarde. Il s'agit du certificat X509, l'un des fondements du protocole TLS, que nous étudions à présent.

1.5 Le protocole TLS

1.5.1 Présentation générale

Le protocole TLS, anciennement dénommé SSL, a été inventé par Netscape dans les années 1990. Il s'agit d'un protocole se situant entre la couche transport et la couche applicative visant à ajouter une couche sécuritaire protégeant les données applicatives. C'est du moins son utilisation la plus courante : le protocole TLS a en effet l'avantage de pouvoir sécuriser, par un canal chiffré, et sans souci d'interopérabilité, tout protocole de niveau applicatif (FTP[44], HTTP, SMTP, etc.), tout en nécessitant une connexion fiable au niveau du transport, par exemple le protocole TCP. Le fait que les protocoles de niveau applicatif permettent, par défaut, un échange de données en texte clair a très vite attiré l'attention sur les nombreux risques qu'un tel parti pris comporte. Etablir un protocole standardisé, interopérable, qui permette d'offrir, entre autres, confidentialité et intégrité des données, était donc loin d'être un luxe et explique en grande partie le succès de TLS.

Dans l'usage quotidien, les protocoles applicatifs les plus sollicités sont sans conteste HTTP et SMTP. Ainsi, dans le cas de HTTP tous les échanges réalisés avec un serveur Web transitent sans protection dans le réseau, laissant toute liberté à un attaquant ou un espion pour observer le contenu des requêtes et réponses. Cela comprend, bien entendu, les données sensibles telles que les mots de passe utilisateur ou les numéros de carte bancaire. Dans le cas de SMTP, non seulement l'identifiant et mot de passe du client sont transmis en clair à travers le réseau, mais les mails eux-mêmes peuvent être lus en entier par tout individu malveillant, et même éventuellement modifiés.

En réponse à ces défauts de conception, le protocole TLS propose d'établir, entre deux entités du réseau, les services de sécurité suivants (la définition de ces services et les principaux types d'attaques réseau sont donnés en annexe A) :

- **Confidentialité des données** par la mise en place d'un tunnel sécurisé entre client et serveur dans lequel les données sont chiffrées par un algorithme cryptographique symétrique tel que RC4 ou AES.

- **Intégrité des données** par le calcul d'un MAC (*Message Authentication Code*) sur chaque fragment des données applicatives échangées. En général l'algorithme utilisé est un HMAC[45] qui s'appuie sur des fonctions de hachage classiques telles que MD5[46] ou bien SHA-1[47].

- **Protection contre le rejeu** par l'ajout d'un numéro de séquence pris en compte dans le calcul des MAC.

- **Authentification à sens unique ou mutuelle** par l'intermédiaire de certificats numériques X509. Ces derniers sont un maillon des infrastructures à clé publique (PKI) largement mobilisées dans le cadre du protocole TLS.

La confidentialité et l'intégrité reposent sur des algorithmes cryptographiques robustes – malgré les faiblesses découvertes dans les fonctions de hachage MD5[48] et SHA-1[49], de nouvelles fonctions ont été développées, telles que la famille de fonctions SHA-2 sur lesquelles aucune collision n'a encore été trouvée. Par conséquent, ces services relèvent purement de la sécurité, alors que l'authentification relève davantage de la confiance. C'est donc sur ce dernier point que nous allons nous attarder dans la section 1.5.3, après avoir présenté le fonctionnement de TLS de manière un peu plus détaillée. Par ailleurs, TLS a fait l'objet de plusieurs analyses de sécurité, telles que [54] et [55], qui garantissent la solidité du protocole.

1.5.2 Détail d'une session TLS

Une session TLS comporte deux phases bien distinctes : la mise en place du tunnel sécurisé, réalisée par le protocole *Handshake*, et l'échange des données applicatives chiffrées par le protocole *Application Data*. Dans tous les cas, ces deux protocoles, ainsi que *ChangeCipherSpec* et *Alert*, sont encapsulés dans un protocole appelé *Record*. Le protocole *ChangeCipherSpec* a pour objet de signaler au *Record* tout changement des paramètres de sécurité, et *Alert* sert à prévenir de tout avertissement ou erreur dans le déroulement d'une session TLS. Quant au protocole *Application Data*, il transmet au *Record* les données applicatives afin que celles-ci soient fragmentées, éventuellement compressées, et que chaque fragment soit l'objet du calcul d'un MAC avant d'être chiffré puis transmis. Nous détaillons désormais le protocole *Handshake* qui sera le centre de notre intérêt, puisque c'est au cours de cette phase qu'a lieu l'authentification.

1.5.2.1 Le Handshake

Le *Handshake* permet aux deux entités de se mettre d'accord sur les paramètres de sécurité à établir, et notamment sur la *ciphersuite* qui sera utilisée, c'est-à-dire sur les algorithmes de cryptographie symétrique, asymétrique, et les fonctions de hachage qui seront employés. Le déroulement d'une session complète (mode *full*) est représenté sur le diagramme de la figure 5.

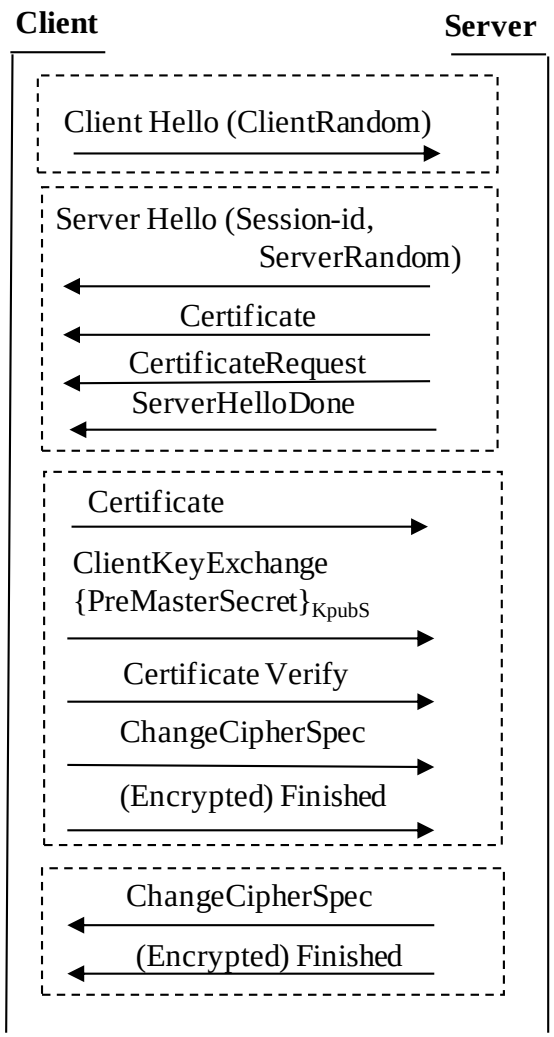

Figure 5 : Handshake TLS en mode full

Avant tout, le client envoie un message *Client Hello* qui contient, notamment, une valeur aléatoire, un *timestamp*, ainsi qu'un ensemble de *ciphersuites* proposées au serveur. Parmi celles-ci, le serveur choisit la première qu'il est capable de gérer (s'il n'en supporte aucune, le *Handshake* échoue) et renvoie un message *Server Hello* qui contient, entre autres, une valeur aléatoire, la *ciphersuite* sélectionnée, et le certificat X509 du serveur. Ce certificat contient la clé publique du serveur et permet d'authentifier ce dernier auprès du client selon le principe des PKI. Les valeurs aléatoires échangées ont pour but d'empêcher les attaques de type rejeu, et servent notamment de graine pour le calcul des clés d'intégrité et de chiffrement.

A l'issue du message *Certificate* du serveur, deux cas de figure peuvent se présenter, relativement à l'authentification. Si celle-ci est **à sens unique** (configuration par défaut dans la plupart des connexions utilisateurs à des serveurs), le client ne s'authentifie pas à ce stade, et le *Handshake* se poursuit. Si elle est **mutuelle**, le message *Certificate Request* est envoyé au client, qui doit répondre par l'envoi de son certificat en clair et par un message *Certificate Verify*, qui est une signature réalisée avec la clé privée correspondant à la clé publique du certificat. Si le client ne possède pas de certificat ou si celui-ci est invalide, le *Handshake* échoue.

A la suite de ces échanges, le client génère un nombre aléatoire de 48 octets appelé *premastersecret*, qu'il transmet au serveur en le chiffrant avec sa clé publique, dans le message *ClientKeyExchange*. Ce secret permet alors au client et au serveur de calculer le même *Master Secret*, qui est un élément de sécurité essentiel de la session TLS, notamment parce qu'il est permanent et qu'il permet de calculer les clés temporaires symétriques utilisées pour le chiffrement des données applicatives.

Enfin, le client, puis le serveur, s'envoient les messages *ChangeCipherSpec* et *Finished*. Le premier indique à son destinataire que les messages postérieurs seront chiffrés à l'aide des paramètres définis au cours du *Handshake* et du *Master Secret*. Quant au message *Finished*, il s'agit d'un hachage de tous les messages échangés au cours du *Handshake*, qui fait également intervenir le *Master Secret*, et chiffré avec les paramètres de session précédemment négociés. Il permet de s'assurer que le client et le serveur peuvent chiffrer et déchiffrer correctement les messages, mais aussi d'empêcher les attaques portant sur l'intégrité des messages du *Handshake*. La session TLS échoue si la vérification d'un au moins des messages *Finished* a échoué.

1.5.2.2 L'échange des données applicatives chiffrées

Une fois la phase *Handshake* achevée, le client et le serveur peuvent s'échanger les données applicatives qui seront chiffrées, comme le message *Finished*, selon les paramètres de session négociés.

Le chiffrement est de type symétrique, et est réalisé à partir de clés temporaires, appelées *Key Blocks*. Un *Key Block* est un ensemble de quatre clés (deux pour l'intégrité, deux pour le chiffrement) qui est calculé d'après la valeur du *Master Secret* et des valeurs aléatoires échangées dans les messages *Hello* du *Handshake*. Le client et le serveur ont chacun leur clé de chiffrement et d'intégrité, qu'ils utilisent ainsi pour la transmission sécurisée de données. La formule de calcul est décrite dans [9], où PRF désigne la *Pseudo Random Function* décrite dans ce même document :

```
key_block = PRF(SecurityParameters.master_secret,
                          "key expansion",
                          SecurityParameters.server_random +
                          SecurityParameters.client_random);
```

Le point qui retient ici notre attention est la différenciation entre les *Key Blocks* et le *Master Secret*. Comme nous l'expliquons dans le paragraphe suivant, pour une session donnée – qui peut être ré-établie plusieurs fois de suite – les premiers sont éphémères alors que le second est permanent. Cela induit des implications différentes du point de vue de la sécurité. En effet, la compromission d'une clé temporaire par un attaquant lui permettra de déchiffrer une portion très réduite des données applicatives, mais ne lui permet ni de déduire les autres clés temporaires, ni de remonter au *Master Secret*. En revanche, la connaissance de ce dernier peut permettre à un attaquant de déduire toutes les clés temporaires, et donc de déchiffrer l'ensemble des données applicatives de

la session. Le *Master Secret* est donc un élément extrêmement sensible, qu'il faudrait traiter avec la même attention que des clés privées asymétriques – à savoir, en privilégier le stockage sur des environnements de confiance tels que des cartes à puce.

1.5.2.3 Reprise d'une session existante

Afin d'améliorer les performances du protocole TLS, dont la phase de *Handshake* peut s'avérer coûteuse en termes de trafic et de calculs si elle est exécutée entièrement à chaque fois que le client envoie une requête au serveur, il est possible de négocier la reprise d'une session récente. Dans ce cas, le *HandShake* est dit en mode *resume*, par opposition au mode *full* qui a lieu lorsqu'une nouvelle session est établie. La figure 6 illustre l'échange de messages lors d'une session TLS en mode *resume*.

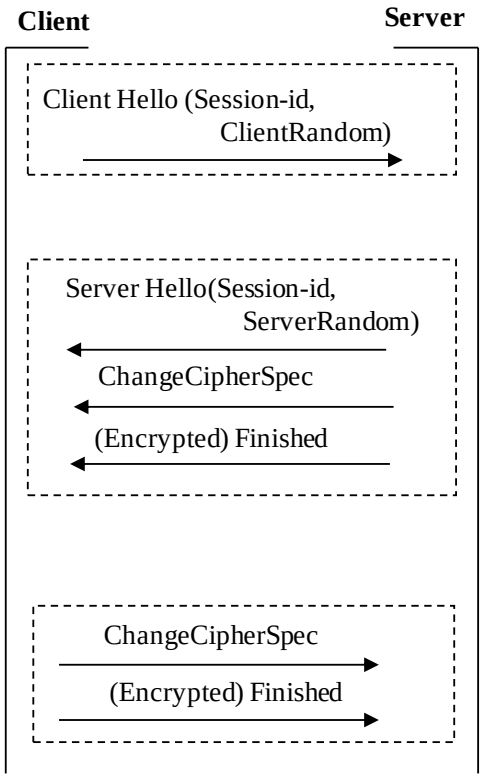

Figure 6 : Handshake TLS en mode resume

Pour cela, le client envoie dans le *Client Hello* l'identifiant de la session qu'il souhaite reprendre. Si le serveur reconnaît cet identifiant et qu'il accepte la reprise de session, la phase d'authentification et l'envoi du *premastersecret* par le client n'a pas lieu. Le serveur envoie alors, juste après le message *Server Hello* (qui contient l'identifiant de la session reprise), les messages *ChangeCipherSpec* et *Finished*. Le client n'a alors plus qu'à envoyer ces mêmes messages. La session TLS utilisera alors les mêmes paramètres de sécurité que ceux qui ont été négociés à l'origine – et donc, notamment, le même *Master Secret*.

Dans le cas où l'identifiant de session à reprendre est trop ancien, ou bien si le serveur refuse la reprise de session, il envoie un nouvel identifiant de session dans son message *Server Hello*, et la phase de *HandShake* est exécutée entièrement.

Dans tous les cas en revanche, les valeurs aléatoires envoyées par le client et le serveur sont renouvelées. Par conséquent, même dans le cas d'une reprise de session, les *Key Blocks*, et donc les clés de chiffrement, seront différents d'une requête à une autre.

1.5.3 L'identité et l'authentification dans TLS

1.5.3.1 L'authentification par certificat X509

Nous ne détaillerons pas dans ce rapport les différentes attaques spécifiquement cryptographiques dont peut être victime le protocole TLS. En général, l'utilisation d'une *ciphersuite* appropriée, avec des tailles de clé suffisamment longues, permet de s'en prémunir.

En revanche, nous nous attardons sur les mécanismes d'authentification et sur le traitement de l'identité numérique. A l'heure actuelle, en effet, l'authentification dans TLS repose exclusivement sur l'utilisation de certificats X509. Cela signifie que l'identité numérique d'un individu ou d'une machine est entièrement représentée par le certificat. Son intérêt premier, bien qu'il ne se limite pas à cela, est d'associer une dénomination (le *Common Name*, usuellement abrégé CN, par exemple l'URL d'un serveur ou le nom d'un utilisateur) à une clé publique asymétrique. L'entité qui détient le certificat détient alors également la clé privée associée, qui sera utilisée pour prouver qu'elle est bien le détenteur légitime de ce certificat.

L'obtention d'un certificat consiste, en accord avec le modèle des PKI, à en faire la demande auprès d'une autorité compétente reconnue, à savoir les Autorités de Certification (CA). Lorsque le certificat est généré par cette autorité, son contenu est signé numériquement par cette dernière, ce qui empêche toute personne de modifier le contenu du certificat, ce qui aurait pour effet de rendre la signature invalide – modulo la solidité de la fonction de hachage utilisée pour ladite signature qui peut conduire, en cas de défaillance, à des problèmes de collisions sur des certificats, comme cela a été le cas avec l'utilisation de la fonction MD5[50]. La vérification d'un certificat consiste alors à vérifier le CN (par exemple vérifier la concordance de l'URL d'un serveur avec celle qui a été saisie dans le navigateur de l'utilisateur), puis à vérifier la signature du certificat, à l'aide de la clé publique du CA. Etant donné qu'il est possible à un CA de déléguer la génération de certificats à d'autres organismes, la vérification d'un certificat nécessitera la vérification de la chaîne des certificats des CA ayant été impliqués, jusqu'à remonter à un CA reconnu, préalablement désigné comme tel pour l'entité vérificatrice. Dans le cas d'un utilisateur vérifiant le certificat d'un serveur Web par exemple, le CA racine devra figurer dans la liste de son navigateur.

Le principal point fort de l'authentification par certificat, à l'image des modèles asymétriques, tient au fait que **le modèle d'identité public (le certificat, donc) ne contient aucun élément**

susceptible d'intéresser un attaquant. Un certificat ne peut être utilisé que par celui qui en possède la clé privée, laquelle ne transite jamais sur le réseau. Il faut toutefois noter que malgré ces caractéristiques, le fait que le certificat utilisateur transite en clair, lors d'une authentification mutuelle, donnerait malgré tout à un espion des renseignements sur l'identité dudit utilisateur. Autrement dit, il n'existe pas de mécanisme de protection de l'identité cliente.

En outre, ce mode d'authentification est reconnu pour être **résistant aux attaques de type MITM**. En termes de sécurité, il nous apparaît donc particulièrement intéressant à étudier, bien qu'il possède également un certain nombre de limitations.

1.5.3.2 Les problèmes liés à l'authentification dans TLS

En excluant les problèmes de nature technique tels que le clonage de certificats, qui peuvent être résolus par l'utilisation de fonctions de hachages plus solides que MD5 ou SHA-1, le modèle d'identité déployé dans TLS peut faire l'objet des critiques suivantes :

- **L'utilisation des PKI**

L'une des critiques majeures adressées à l'authentification par le protocole TLS est liée à l'utilisation des PKI et à la complexité de gestion des certificats. Il n'est pas rare, en effet, qu'un serveur utilise un certificat périmé, ou bien que les listes de révocations des certificats ne soient pas à jour, ou encore qu'une autorité de certification ne soit pas reconnue par un navigateur, pour ne citer que quelques exemples. Dans la plupart des cas, cela amène à des avertissements ou à l'échec d'une session TLS où aucun attaquant n'est intervenu et où aucune entité n'était malveillante. Il est plus rare en revanche que cela conduise au bon déroulement d'une session qui aurait dû échouer (cas le plus dangereux, du point de vue de la sécurité), mais cela peut arriver, comme dans le cas où les listes de révocation ne sont pas à jour.

Il s'agit là de défauts intrinsèques au modèle d'identité qu'est le certificat X509, qu'il convient de prendre en compte, mais qu'il est nécessaire de relativiser, notamment si l'on compare avec la biométrie ou les mots de passe. Dans ces derniers cas, si un attaquant récupère le secret de la victime à son insu, celui-ci n'est pas davantage révoqué, aussi longtemps que la victime ou le système d'authentification n'aura pas repéré la fraude.

- **L'absence globale de déploiement de l'authentification mutuelle**

Dans la très grande majorité de ses déploiements, i.e. dans la sécurisation de serveurs accessibles au grand public, TLS met en place une authentification à sens unique, qui ne prend pas en compte l'authentification de l'utilisateur. La raison, évidente, est liée à la complexité de gestions des certificats, qu'il est difficile d'imposer à l'utilisateur sans dispositions préalables propres à lui simplifier la vie. Il n'en reste pas moins qu'en l'état, l'utilisateur peut être l'objet de vol d'identifiants malgré la protection offerte par le canal TLS, comme il a été expliqué dans les sections précédentes.

La difficulté liée au développement d'un modèle d'identité et d'authentification imposant une authentification mutuelle se heurte alors au problème de l'intégration de ce modèle dans le parc de serveurs actuel, dont il faudrait modifier la configuration de chacun d'entre eux pour mettre en place ce mode d'authentification. Toutefois, ce problème peut être contourné, au moins partiellement, par les architectures de fédération d'identité, qui seront détaillées au chapitre 2.

- **Le défaut de confiance dans l'authentification mutuelle**

Dans les cas où l'authentification mutuelle est mise en place, se pose le problème de la confiance liée à l'environnement où se déroule le protocole TLS. Notamment, les éléments cryptographiques les plus sensibles sont souvent stockés sur des machines qui n'ont pas bénéficié de contre-mesures efficaces visant à empêcher la lecture ou la modification de données. Ainsi, sur un environnement classique tel qu'un ordinateur sans contre-mesures spécifiques, une clé privée RSA a pu être retrouvée via une attaque de type *Branch Prediction* par l'observation d'un seul calcul RSA[51], et une clé symétrique AES a pu être retrouvée par une attaque sur la mémoire cache en 65 ms[52]. Mais les clés privées ne sont pas les seuls éléments à sécuriser. Comme nous l'avons souligné auparavant, le *Master Secret* d'une session TLS est extrêmement important, et doit faire l'objet d'une attention toute particulière quant à son stockage, ce qui, actuellement, n'est pas le cas.

Au-delà de ces critiques, le modèle d'identité représenté par le certificat X509 dans le cadre du protocole TLS bénéficie d'avantages certains, notamment celui de ne pas être sensible à l'exposition publique, et celui d'être déjà largement déployé et considéré comme un standard incontournable. Parmi les critiques exprimées dans ce paragraphe, il est possible de remédier au moins aux deux dernières, tout en prenant en compte les problèmes de la première afin d'en limiter la portée. Cela sera l'objet du chapitre 3 de ce rapport.

1.5.4 Le protocole TLS-PSK

Le protocole TLS bénéficie de *ciphersuites* spécifiquement dédiées à une authentification basée sur le paradigme du Pre-Shared-Key (PSK)[53]. Celui-ci consiste à partager préalablement un secret avec le serveur distant, qui servira alors à la dérivation, de part et d'autre, du *premastersecret*, puis du *Master Secret*. Dans ce modèle, le partage du secret est le seul moment où celui-ci est susceptible d'être intercepté par un attaquant. En dehors de cet instant, pour lequel des précautions adéquates peuvent être prises, le secret partagé ne transite jamais directement dans le réseau, y compris sous la forme d'un condensât, ce qui empêche la réalisation d'attaques de type MITM simples. En cela, ce mode d'authentification, bien que symétrique, diffère sensiblement de l'authentification par mot de passe. Une session TLS-PSK peut être modélisée par le diagramme représenté figure 7.

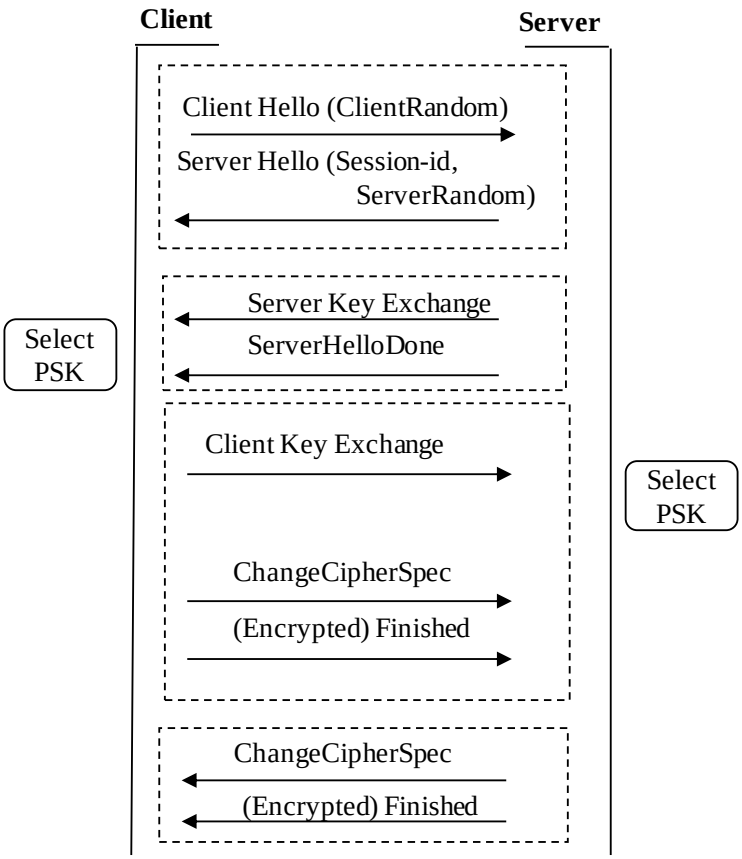

Figure 7 : Handshake du protocole TLS-PSK

L'élément le plus sensible, dans ce contexte, est bien évidemment la PSK, i.e. le secret partagé, même si les remarques formulées précédemment à propos du *Master Secret* restent valables. A nouveau, plus que la sécurité, c'est ici la question de la confiance qui se pose. Il faut en effet des garanties sur le stockage des éléments sensibles, aussi bien du côté du serveur, où la PSK est stockée nécessairement en clair, que du côté du client. En revanche, cette approche nous amène à considérer un modèle d'identité alternatif au certificat X509 dont la complexité de gestion peut être un frein à son déploiement. Dans le cas présent, le modèle d'identité est proche de celui de l'identifiant / mot de passe. Mais ici il est possible de forcer le mot de passe à être une clé aléatoire d'une longueur déterminée, ce qui permet de supprimer tous les problèmes liés à la longueur et à l'entropie des mots de passe. L'utilisateur lui-même n'a pas besoin de connaître sa PSK, qui sera stockée dans un environnement déterminé après sa génération. Un problème qui peut alors se poser est celui de la portabilité, puisqu'un changement d'environnement peut induire l'impossibilité d'utiliser sa PSK.

Comme dans le cas de l'authentification asymétrique mutuelle de TLS, l'utilisation de composants sécurisés et portables tels que des cartes à puce permettent de résoudre à la fois le problème de la portabilité et celui de la confiance.

1.6 Conclusion

L'analyse des principaux modèles d'identité et d'authentification associés a permis de dégager les principales caractéristiques à mettre en valeur ou à écarter dans la perspective de l'élaboration d'un modèle d'identité numérique. Un modèle adapté à la convergence doit en effet pouvoir bénéficier d'un haut niveau de sécurité afin de protéger les différents usages de l'identité numérique au cœur des réseaux – notamment, la préservation des données personnelles, et l'accès éventuellement payant à tous types de services et de flux de données. La garantie d'une authentification forte est donc un paramètre indispensable dans la création d'un modèle d'identité. Comme nous l'avons vu, les qualités offertes par le protocole TLS nous amènent à considérer les modèles d'identité numérique qui lui sont associés d'un œil plus attentif que les modèles à base de mots de passe ou de biométrie.

Par ailleurs, l'adaptation à la convergence implique une conformité avec les standards en vigueur. A ce titre, le protocole TLS est reconnu comme étant le protocole phare de la sécurisation du Web. Un modèle d'identité qui s'appuierait sur ce protocole tout en en corrigeant les défauts serait donc, par nature, convenable de ce point de vue. Mais il existe également des contraintes de performances : les propositions de solution aux défauts de TLS, impliquant notamment l'utilisation de composants sécurisés dédiés, doivent nuire le moins possible à la rapidité d'exécution aussi bien du *Handshake* TLS (en mode *full* comme en mode *resume*) que des opérations de chiffrement et déchiffrement des données applicatives.

Enfin, il est essentiel qu'un modèle d'identité et d'authentification impliquant une configuration particulière (i.e. normative mais non classique) du serveur d'authentification puisse être aisément compatible avec les architectures de fédération d'identité qui peuvent en accélérer considérablement le déploiement sur le Web. C'est ce point que nous détaillons dans le chapitre suivant de ce rapport.

Chapitre 2 : La fédération d'identité et le Single Sign On

2.1 Introduction

Le développement des services personnalisés sur les réseaux a naturellement amené l'identité numérique à se placer au centre des préoccupations des utilisateurs comme des fournisseurs de service. Nous l'avons vu dans le chapitre précédent, la sécurisation des accès et une authentification forte sont des enjeux majeurs du domaine, pour lesquels aucune réponse satisfaisante n'est toutefois déployée. Mais à cela s'ajoute également une autre dimension, qui est celle de la difficulté de gestion des identités numériques, aussi bien du point de vue de l'utilisateur que de celui des fournisseurs de service.

En effet, la tendance naturelle a consisté à associer une identité différente (au moins, mais parfois davantage) à chaque service, ce qui, du point de vue des utilisateurs, multiplie le nombre de comptes et d'identifiants dont il est nécessaire de se souvenir. Dans un cadre où le mot de passe tient une place importante dans les modèles d'identité utilisateur répandus, cela devient rapidement ingérable. En compensation de cette complexité pour l'utilisateur, la gestion des droits d'accès est en revanche extrêmement simple. Le service authentifie lui-même l'utilisateur et accorde en conséquence les droits associés à l'identité considérée. Autrement dit, le service est entièrement maître de la sécurisation des accès au serveur.

Toutefois, les inconvénients de cette approche de gestion « indépendante » de l'identité ont été suffisamment forts pour pousser à la création d'un autre modèle de gestion « centralisé » de l'identité numérique qui prend le contrepied de cette approche. Les architectures de Single Sign On (SSO) et de fédération d'identité, qui sont les représentants – déclinés selon plusieurs variantes sur lesquelles nous reviendrons dans ce chapitre – de cette mouvance, visent en effet à faciliter la vie de l'utilisateur en lui permettant d'associer une de ses identités à plusieurs services. Les conséquences premières en sont assez évidentes : du point de vue de l'utilisateur, la gestion des identités est extrêmement simplifiée, puisque le nombre d'identifiants à retenir est drastiquement réduit. En revanche, du point de vue des services, la question de la sécurisation des accès est nettement plus complexe à gérer, puisque l'authentification à un service donné est déléguée à un tiers, qui doit faire parvenir au dit service le succès ou l'échec de l'authentification, ainsi que, le cas échéant, certains attributs associés à l'identité authentifiée. Autrement dit, il s'agit de faire circuler des informations sensibles entre des domaines différents, dont les politiques de sécurité doivent être compatibles.

Plusieurs architectures de fédération d'identité ont vu le jour. Dans ce chapitre, nous en détaillons différents types de mise en œuvre et de technologies, notamment Windows Cardspace[56], le langage SAML[19] qui est la fondation des architectures Liberty Alliance[57] et Shibboleth[58],

ainsi qu'OpenID[18], dans le but de dégager les éléments fondamentaux communs à la majorité de ces architectures. Ainsi, il sera possible d'orienter la conception de notre modèle d'identité numérique de manière à en assurer la compatibilité avec ces dernières et à en réaliser une preuve de concept.

2.2 Fédération d'identité et SSO

Avant de nous intéresser aux architectures existantes, il convient de préciser les définitions couramment associées à la « Fédération d'identité » et au « Single Sign On ». Dans l'usage courant, ces termes tendent en effet à être utilisés, à tort, de manière interchangeable. Dans les deux cas, le paradigme de l'authentification unique reste le même, et fait intervenir trois entités principales dans les échanges : l'utilisateur, le fournisseur de service (SP, pour *Service Provider*), et le fournisseur d'identité (IdP, pour *Identity Provider*). Le schéma générique d'une authentification dans un tel contexte est représenté sur la figure 8 – laquelle ne fait pas apparaître distinctement la manière dont le SP vérifie le jeton d'authentification. En effet, ce point, ainsi que la manière dont le SP découvre l'adresse de l'IdP où l'utilisateur sera redirigé, sont ceux pour lesquels les solutions proposées divergent le plus, techniquement parlant.

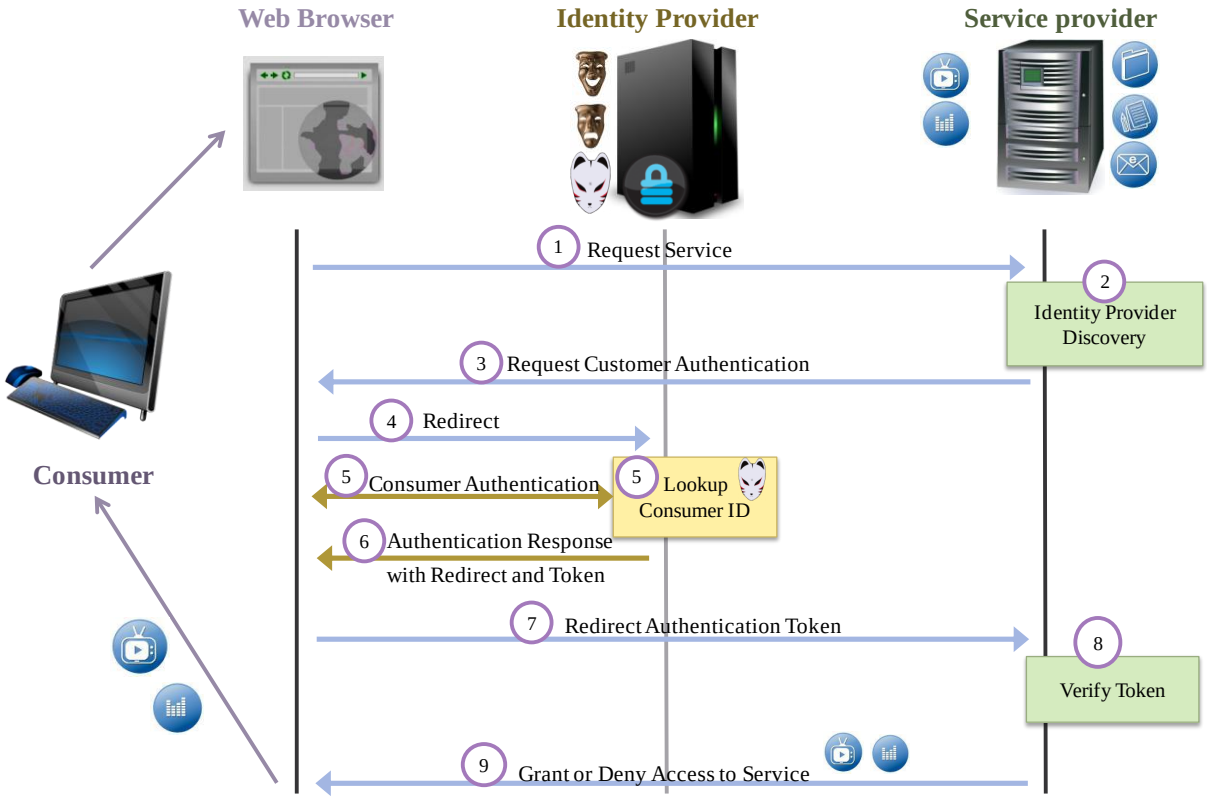

Figure 8 : Scénario d'authentification dans une architecture de fédération d'identité

En réalité, le SSO désigne une architecture permettant à un utilisateur de s'authentifier de manière unique pour avoir accès à un ensemble de services au sein d'un unique domaine. Le

fonctionnement est donc assez analogue, par exemple, à ce que propose le protocole Kerberos[59], où le serveur d'authentification jouerait le rôle de l'IdP. L'une des premières architectures SSO a été lancée en 1999 par Microsoft via son .Net Passport, aujourd'hui rebaptisé Windows Live ID, qui permettait d'obtenir, après authentification, l'accès à un ensemble de sites affiliés à Microsoft.

La Fédération d'identité a, quant à elle, une ambition plus large. Contrairement au SSO, elle vise en effet à établir l'authentification unique vers des services appartenant à des domaines différents. Une telle architecture n'est pas concevable sans discussion préalable entre les différentes parties, qui doivent notamment aboutir à la signature de contrats écrits. Il s'établit alors un *cercle de confiance*, qui comprend plusieurs IdP et SP des différentes organisations ayant conclu un accord. La fédération d'identité peut alors être considérée comme une sorte de SSO inter-domaines pouvant avoir lieu au cœur d'un cercle de confiance déterminé. Le principe du cercle de confiance est illustré par la figure 9.

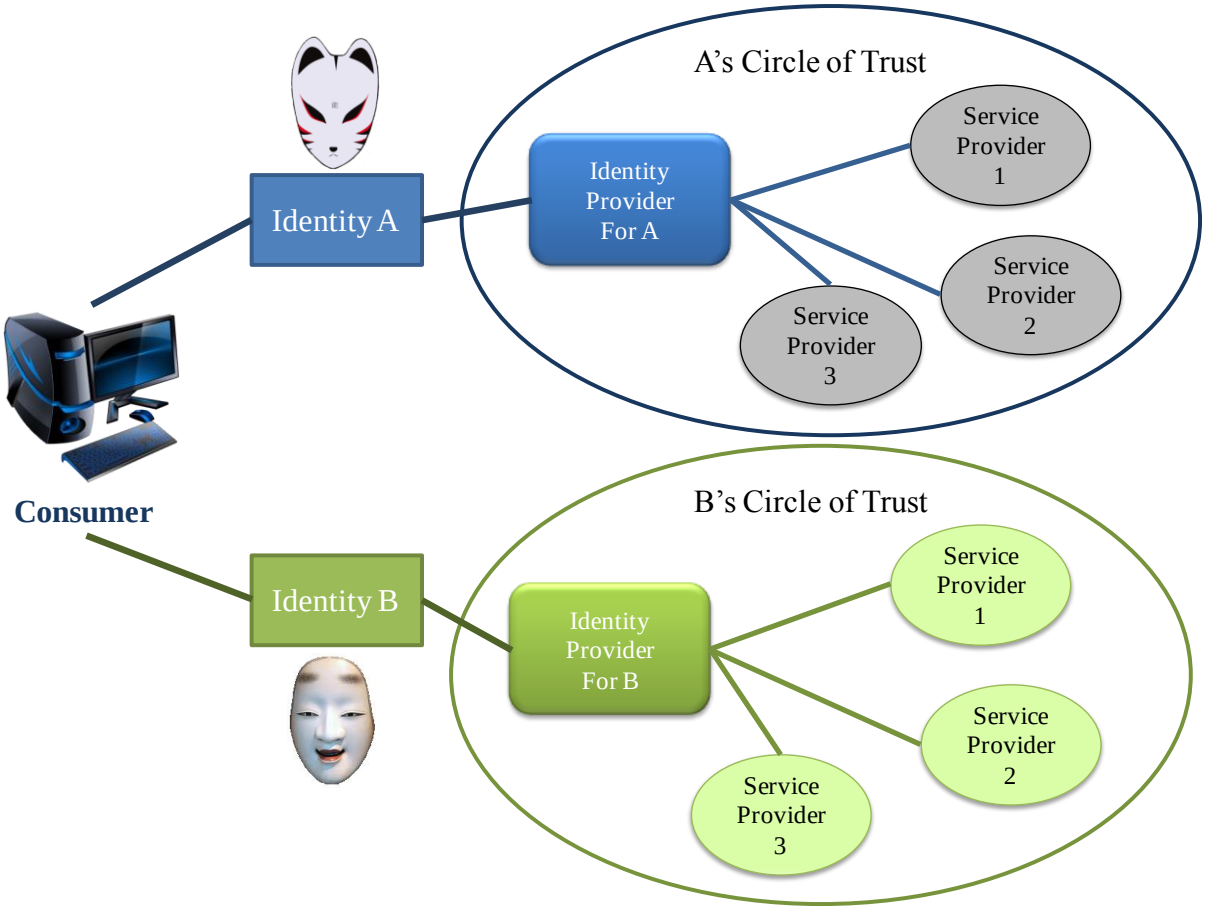

Figure 9 : Le principe du cercle de confiance

Etant donné la nature même du modèle, orienté vers la mise en place de partenariats financiers, il n'est pas étonnant que la quasi-totalité des plateformes de fédération d'identité soient propriétaires et payantes. La seule exception notable reste Shibboleth, qui est libre et gratuit.

Bien que les implémentations des architectures de SSO ou de fédération d'identité présentent des différences, certains aspects fondamentaux restent invariables, notamment deux d'entre eux qui vont retenir notre attention :

- La nécessité de transmettre entre l'IdP et le SP (qui éventuellement ne se connaissent pas et n'appartiennent pas au même domaine) un jeton contenant des données relatives à l'identité de l'utilisateur et à son authentification.

- La force de l'architecture globale repose sur le niveau de sécurité dans la gestion dudit jeton, mais aussi sur le niveau de sécurité du protocole d'authentification mis en œuvre par l'IdP.

- Un troisième point que l'on peut souligner, mais qui restera hors du champ d'étude de ce rapport de thèse, est la nécessité de résoudre la question de la découverte de l'IdP par le SP lors d'une authentification de l'utilisateur. En effet, l'authentification étant déléguée à une entité externe au SP, celui-ci doit pouvoir rediriger convenablement l'utilisateur vers l'IdP, même s'il n'en a pas connaissance au préalable. Par conséquent, quelle que soit la technologie mise en œuvre, la question de la découverte de l'IdP doit être résolue.

En général, les plateformes de fédération d'identité ont le souci de conserver le maximum d'interopérabilité, incarnée par un large choix de méthodes d'authentification proposées aux utilisateurs. Notamment, l'interdiction d'une authentification par mot de passe reste très rare. Or, les exigences de sécurité dans ce contexte devraient être notoirement plus élevées que dans le cas d'une gestion « indépendante » des identités. En effet, dans le cas d'une usurpation d'identité, c'est l'accès à un ensemble de services, et non à un seul, que gagne l'attaquant. Les réticences du milieu de la fédération d'identité envers le bannissement complet de l'authentification par mot de passe tient, en dehors des éventuels problèmes d'interopérabilité, à l'accroissement de la complexité de déploiement d'un système déjà peu aisé à appréhender – et donc à la perte de marché qui en découlerait. Toutefois, cette conséquence n'est pas nécessairement une fatalité. Elle tient son origine, comme souligné dans le chapitre 1, de l'absence de solution sécurisée standard qui pourrait être aisément déployée, y compris auprès d'un public restreint. Autrement dit, c'est à nouveau l'absence d'un modèle d'identité adapté qui est ici en cause.

2.3 Les solutions pour la fédération d'identité

Nous nous intéressons désormais à plusieurs incarnations technologiques de la théorie de la fédération d'identité. Comme nous l'avons souligné, le SSO peut être décrit comme un sous-ensemble de la fédération d'identité. Nous nous focaliserons donc par la suite davantage sur cette dernière. Dans cette section, notre but n'est pas de détailler les différences entre les produits de fédération d'identité, ni d'effectuer un inventaire exhaustif de l'existant, mais plutôt d'observer les différentes formes et technologies mises en œuvre dans des solutions de nature suffisamment différente pour

être complémentaires. Les exemples que nous avons choisis nous ont été inspirés par [60], dont la sélection permet d'étudier le domaine de la fédération d'identité sous le prisme de la diversité de ses incarnations.

2.3.1 Le protocole Infocard et Windows Cardspace

Windows Cardspace[56] est un composant .Net qui implémente le protocole Infocard[61], et qui fournit à l'utilisateur une interface de gestion des identités numériques représentées sous forme de « cartes », stockées sur le poste de l'utilisateur sous la forme de fichiers XML. Nous amorçons notre analyse par ce composant car il s'agit de la représentation la plus visuelle du principe de la fédération d'identité du point de vue de l'utilisateur.

Les cartes dont peut disposer l'utilisateur sur l'interface Cardspace sont constituées d'un ensemble d'assertions relatives à une identité, conformément à ce que préconise le protocole Infocard. Ces cartes peuvent être de deux sortes :

- Les cartes n'ayant pas fait l'objet d'une certification par un IdP externe, c'est-à-dire les cartes auto-certifiées. Ces dernières représentent des identités contenant un ensemble d'attributs restreint.

- Les cartes ayant été certifiées par un IdP externe, qui représentent des identités dont l'ensemble des attributs est extensible, et entièrement géré par l'IdP en question. En cela, ces cartes sont très similaires à des assertions SAML (cf. section suivante).

Dans les deux cas, lorsque l'utilisateur souhaite s'authentifier auprès d'un SP, il doit sélectionner une carte dont les attributs sont conformes aux exigences de ce dernier. Par ailleurs, Cardspace résout le problème de la découverte de l'IdP de la manière suivante : si la carte est auto-certifiée, le poste de l'utilisateur est lui-même considéré comme l'IdP, sinon l'adresse de l'IdP à contacter est contenue dans la carte.

Cardspace définit ainsi une version purement *software* de gestion des identités, dans un cadre très large. Il ne s'agit pas à proprement parler d'une plateforme de fédération d'identité, mais plutôt d'un moyen permettant au grand public de s'inscrire facilement dans cette mouvance. Lorsque l'utilisateur sélectionne une carte valide, l'IdP ayant certifié l'identité est contacté par le logiciel Cardspace dans le but d'obtenir, après authentification de l'utilisateur auprès de l'IdP, un jeton XML signé qui contient les informations demandées par le SP – procédé pouvant permettre à l'IdP de savoir quels SP l'utilisateur consulte, ce qui constitue un risque latent pour la vie privée. Les requêtes de jeton reposent sur les protocoles WS-* des Web Services[64], notamment WS-Trust[65]. Les moyens d'authentification de l'utilisateur sont nombreux, et contiennent notamment le couple identifiant / mot de passe. Néanmoins Cardspace propose des contre-mesures permettant d'empêcher les attaques de type *phishing*.

Malgré tout, Cardspace reste sensible aux attaques de type MITM portant sur une attaque DNS relative aux noms de domaine de l'IdP ou du SP [62]. Par ailleurs, le stockage des identités dans un environnement déterminé demeure problématique en cas de changement de poste utilisateur, ce qui limite le déploiement de cette solution. Par conséquent, Cardspace souffre d'un réel problème de portabilité qui n'a pas été résolu. Enfin, la limitation aux protocoles WS-* fait également partie des défauts qui, avec le manque de succès rencontré par ce projet, ont conduit à l'abandon du projet par Microsoft en 2011.

2.3.2 Le langage SAML

2.3.2.1 Liberty Alliance et Shibboleth

Du point de vue des technologies du Web, l'un des standards de référence de la fédération d'identité est le langage SAML[19], défini par le consortium OASIS (Organization for the Advancement of Structured Information Standards). SAML offre un cadre flexible pour l'échange d'informations touchant à l'identité et à la sécurité sous la forme de messages XML. Ce cadre, notamment, prévoit l'échange de ces informations entre des domaines distincts, ce qui répond spécifiquement à la problématique principale de la fédération d'identité. Depuis la version 1.0 de ce langage qui a vu le jour en 2002, de nombreux ajouts et extensions ont été développés en relation étroite avec Liberty Alliance, qui est un des acteurs majeurs de la fédération d'identité et des Web Services. Liberty Alliance a notamment développé, au-dessus des éléments de SAML 1.0 visant à permettre le SSO, un cadre complet pour la fédération d'identité (issu des spécifications Liberty Alliance Identity Federation Framework v1.2[57]). L'unification de l'ensemble a donné naissance, en 2005, à la version 2.0 de SAML qui est le standard en vigueur actuellement. Shibboleth[58], projet open-source du consortium Internet2, partage de nombreuses similitudes avec Liberty Alliance. Notamment, il propose une fédération d'identité qui repose également sur SAML 2.0, mais dans un cadre plus restreint que Liberty Alliance. Là où ce dernier, bien que s'appuyant sur des standards ouverts, est déployé dans un contexte commercial avec plus de 150 organisations parmi ses membres, Shibboleth est utilisé par les universités sans visée commerciale, avec des cercles de confiance nettement plus restreints.

Ces deux exemples illustrent ainsi la puissance et l'importance de SAML dans la fédération d'identité. Néanmoins, ils restent critiquables du point de vue de leur accessibilité par le grand public. En effet, le cadre restreint dans lequel fonctionne Shibboleth ne permet pas à l'utilisateur d'en faire un usage très fréquent. Quant à Liberty Alliance, sa visée commerciale et sa complexité induit davantage un ciblage des entreprises que des utilisateurs finaux. Par ailleurs, aussi longtemps que l'authentification pourra être réalisée via le classique couple « identifiant / mot de passe », la sécurité de l'ensemble de l'architecture ne peut être certifiée. Néanmoins, considéré à part entière, le langage SAML demeure un outil très fort dont il nous sera possible de tirer partie lors de la conception de la compatibilité de notre modèle d'identité avec la fédération d'identité. Nous détaillons ainsi à présent les caractéristiques et les possibilités offertes par SAML.

2.3.2.2 Caractéristiques et cas d'utilisation de SAML

Le langage SAML est construit autour de la notion d'*assertion*. Une assertion est un ensemble d'informations représentées au format XML, liées à une identité ayant entrepris une action d'authentification dans un cadre de SSO ou de fédération d'identités. Les assertions SAML sont échangées entre l'IdP et le SP, et peuvent contenir trois types de déclarations (*statements*) :

- Déclaration d'**authentification**, spécifiant le succès d'une authentification en y faisant figurer la méthode d'authentification utilisée ainsi qu'un *timestamp*. Ainsi, si le SP considère que la méthode d'authentification est invalide, il pourra refuser l'accès à l'utilisateur, même en cas de succès de l'authentification.
- Déclaration d'**attributs**, qui contient un ensemble d'informations associées à l'identité considérée.
- Déclaration de **décision d'autorisation**, qui décrit les droits que possède une identité vis-à-vis d'une ressource donnée.

La figure 10 donne un exemple d'assertion SAML contenant les deux premiers types de déclaration. Elle possède la signification suivante : l'assertion « abcdef » déclare que le sujet « 123456abcd », qui est un élève de l'organisation concernée, s'est authentifié par mot de passe transporté de manière sécurisée auprès de l'IdP « idp.issuer.org » afin d'accéder au seul SP « sp.audience.net ». Des conditions relatives au temps de validité de cette assertion sont également précisées.

SAML propose une méthode de découverte de l'IdP reposant sur un cookie rattaché à un domaine commun, mais l'utilisation de SAML au sein de cercles de confiances amène généralement les administrateurs à préférer la saisie directe des informations relatives à l'IdP dans chacun des SP concernés par le cercle.

```
<saml:Assertion
   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ID="abcdef"
   Version="2.0"
   IssueInstant="2012-03-12T19:20:12Z">
   <saml:Issuer>https://idp.issuer.org/SAML2</saml:Issuer>
   <ds:Signature
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
   <saml:Subject>
     <saml:NameID
       Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
       123456abcd
     </saml:NameID>
   </saml:Subject>
   <saml:Conditions
     NotBefore="2012-03-12T19:15:12Z"
```

```
    NotOnOrAfter="2012-03-12T19:25:12Z">
    <saml:AudienceRestriction>
      <saml:Audience>https://sp.audience.net/SAML2</saml:Audience>
    </saml:AudienceRestriction>
  </saml:Conditions>
  <saml:AuthnStatement
    AuthnInstant="2012-03-12T09:20:07Z"
    SessionIndex="abcdef">
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
  <saml:AttributeStatement>
    <saml:Attribute>
      <saml:AttributeValue
        xsi:type="xs:string">student</saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

Figure 10 : Exemple d'assertion SAML

Le formatage des messages SAML et leur compatibilité avec les protocoles de transport standards sont définis dans les *bindings*. SAML 2.0 en définit plusieurs, mais à l'origine un seul existait, qui préconisait ainsi l'encapsulation des messages SAML dans un paquet SOAP[63], lui-même encapsulé dans du HTTP.

Les cas d'utilisation de SAML sont prédéfinis via des *profiles*, qui décrivent la manière dont seront combinés, selon les contextes, les assertions, protocoles, et *bindings*. Conformément à son orientation première, de nombreux *profiles* de SAML 2.0 ont été spécifiés pour le SSO. Le plus important d'entre eux est le *Web Browser SSO Profile*, qui spécifie les différentes possibilités de mise en œuvre du SSO à partir d'un navigateur utilisant le HTTP comme protocole de communication de base. La liste exhaustive des *profiles*, et donc des cas d'utilisation, figure dans les spécifications de SAML 2.0.

Les avantages de SAML tiennent à la maturité de son développement et des cas d'utilisation envisagés qui a permis, entre autres, de traiter efficacement des questions telles que la sécurité et la vie privée, tout en offrant un canevas réutilisable par des projets d'envergure, comme nous l'avons vu. Par ailleurs, SAML est compatible avec les spécifications WS-* des Web Services, que nous ne détaillerons pas dans ce rapport. Il est toutefois intéressant de noter que la spécification WS-Federation[66], datant de 2006, et visant à assurer la fédération d'identité au sein des Web Services, reprend de nombreuses idées à SAML.

Du point de vue du respect de la vie privée, il est à nouveau possible pour l'IdP de connaître, et donc de suivre, les SP consultés par l'utilisateur. SAML offre la possibilité de s'opposer à ce comportement, mais demandera alors une volonté particulière de la part des développeurs (ou de leurs dirigeants) pour aller dans ce sens.

De manière paradoxale, le plus gros désavantage de SAML est la conséquence directe de ce qui fait sa force. En effet, sa maturité l'a amené à devenir une technologie assez lourde et complexe, peu propice à un déploiement massif auprès des utilisateurs finaux. Son but consiste davantage à cibler le développement de projets inter-entreprises, sans pour autant interdire des déploiements plus modestes, mais pour lesquels des technologies plus simples seront souvent préférées, à l'image d'OpenID, que nous détaillons à présent.

2.3.3 OpenID

La technologie OpenID[18] est un standard ouvert, qui a originellement été développé dans une perspective beaucoup plus restreinte que SAML, afin de permettre aux utilisateurs d'une communauté restreinte de s'authentifier de manière décentralisée auprès d'une tierce partie. Il s'agit d'une technologie extrêmement facile à déployer tant du point de vue du SP que de l'IdP, et très accessible aux utilisateurs même non spécialistes du domaine.

Le principe de l'authentification dans OpenID consiste à fournir un identifiant au SP qui est, en général, une URL du type http://idp-address/identity, mais tout type de XRI (eXtensible Resource Identifier) peut être accepté. Cet identifiant doit être saisi sur l'interface d'authentification OpenID du SP, qui redirige alors l'utilisateur sur le site de l'IdP où il peut s'authentifier, selon la méthode en vigueur chez ce dernier. Le résultat de l'authentification subit un HMAC dont la clé est un secret qui a été partagé entre le SP et l'IdP au moment où l'utilisateur est redirigé sur le site de l'IdP, puis ce HMAC est transmis au SP avec éventuellement des données relatives à l'identité de l'utilisateur. A partir de ces éléments, le SP gère alors lui-même l'accès de l'utilisateur à ses services. Les différentes étapes d'une authentification avec la technologie OpenID sont représentées sur la figure 11.

Comme l'illustre la description précédente, OpenID s'est avant tout attaché à offrir un maximum de simplicité. En effet, la question de la découverte de l'IdP est résolue par le format même de l'identifiant OpenID, qui amène naturellement l'utilisateur à fournir lui-même l'adresse de ce dernier. Par ailleurs, si l'authentification unique permet d'accéder à des services appartenant à des domaines différents, il ne s'agit pas de fédération d'identité au sens usuel du terme, qui implique un cercle de confiance préétabli entre différentes organisations. Or, OpenID n'en fait pas usage. Le seul « cercle » de confiance est construit par l'utilisateur au fur et à mesure de ses interactions avec différents SP, qui lui permet ensuite, après une procédure d'authentification en bonne et due forme auprès de l'IdP, de ne saisir que son identifiant OpenID auprès d'un des SP de sa liste en guise d'authentification auprès de ce dernier.

En réalité, il est possible de dire qu'OpenID s'appuie sur le principe de l'*absence* de confiance. Qu'il s'agisse du classique cercle de confiance formalisé comme dans le cas de Liberty Alliance, ou bien d'une simple relation de confiance entre différents SP, ou même entre les SP et l'IdP, aucun de ces liens n'est censé préexister. Le seul lien de confiance établi dynamiquement est celui qui est

matérialisé par le secret partagé entre le SP et l'IdP, mais cela ne garantit en rien une réelle relation de confiance entre ces identités qui, le plus souvent, ne se connaissent pas.

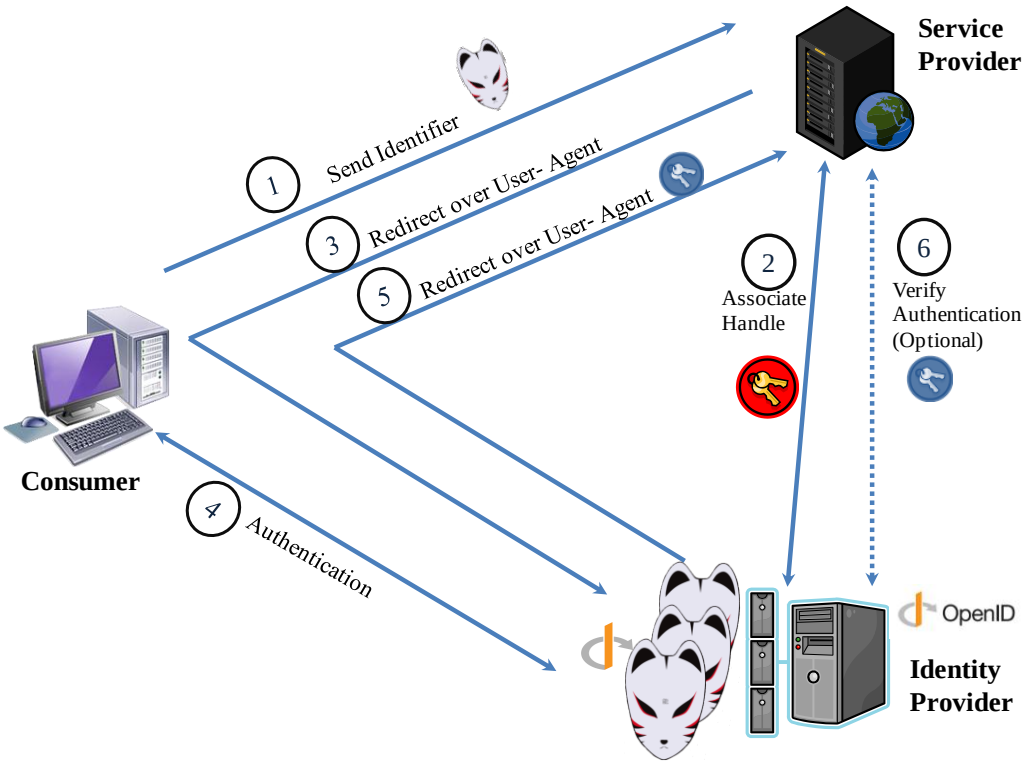

Figure 11 : Authentification avec OpenID

La contrepartie essentielle des avantages qu'offre OpenID en termes de simplicité et de facilité de déploiement tient, fatalement, aux problèmes de sécurité et de confiance. En effet, l'absence de relation de confiance, mais également d'un quelconque lien préalable, entre le SP et l'IdP, conduit à un grand flou dans la phase d'authentification et à la force de cette dernière. Pour tenter de pallier à ce manquement, les spécifications PAPE v1.0[67] permettent d'une part aux IdP de communiquer aux SP leur politique d'authentification, et d'autre part aux SP de demander des garanties spécifiques à ce niveau pour considérer l'authentification valide. Mais ces spécifications datent de 2008, alors qu'OpenID est né en 2005, ce qui conduit à de nombreux cas de SP ou d'IdP qui ne prennent pas ces spécifications en compte. Autrement dit, l'authentification déléguée à l'IdP reste majoritairement du type « identifiant / mot de passe », sans contre-mesure anti-phishing, et la plupart des SP s'en satisfont.

Par ailleurs, du point de vue de la vie privée, non seulement l'IdP connaît, par construction, l'ensemble des SP visités par un utilisateur, mais il est possible à différents SP de recouper les informations personnelles associées à un même identifiant OpenID, i.e. à une même identité – bien que le consentement de l'utilisateur soit nécessaire et explicitement demandé lors de la transmission

de données personnelles vers un SP. La version 2.0 d'OpenID, qui date de décembre 2007, propose une possibilité pour l'utilisateur de donner au SP l'identifiant de l'IdP plutôt que le sien, ce qui limiterait cette tendance, mais une telle solution fait l'objet de trop peu de promotion pour estimer qu'elle puisse avoir un réel impact.

Malgré ces défauts importants qui empêchent OpenID d'être considéré, tel quel, comme une architecture acceptable dans des projets impliquant des aspects légaux et des partenariats importants (pour lesquels la notion de cercle de confiance s'avère indispensable), son principe a été repris et intégré par des acteurs majeurs de l'identité comme Yahoo ou Google, qui sont désormais devenus des IdP de type OpenID. De fait, déployée de manière adéquate, une architecture de type OpenID peut remédier à ses faiblesses tout en conservant ses qualités, c'est-à-dire proposer la délégation d'une authentification unique vers des services différents tout en imposant très peu de complexité tant à l'utilisateur qu'aux SP ou aux IdP.

Pour cette raison, nous avons choisi la technologie OpenID afin d'établir une preuve de concept de la compatibilité du modèle d'identité que nous avons élaboré avec des architectures de type SSO ou fédération d'identité. Ce point sera détaillé dans la deuxième partie de ce rapport de thèse.

2.3.4 Comparatif des différentes solutions

Les différentes solutions à la problématique de la fédération d'identité que nous avons explorées précédemment ne prétendent nullement à l'exhaustivité, mais visent à dépeindre des facettes suffisamment différentes de ce paysage afin d'en faire ressortir, pour chacune, les avantages et les inconvénients, mais également un socle commun. Le comparatif des fonctionnalités qu'effectue [60] donne une vision intéressante des convergences et divergences fonctionnelles de ces trois solutions, que nous résumons dans le tableau 1.

D'un point de vue technologique plus que fonctionnel, nous pouvons également dégager au moins deux principes communs à chaque solution : le caractère primordial de la force de l'authentification, et la sécurisation et la maîtrise des jetons de session à travers plusieurs domaines. C'est notamment ce dernier point qui s'avère être le défi propre à la fédération d'identité. Si de manière générale, la gestion de l'identité numérique est indissociable du processus d'authentification, la fédération d'identité ajoute le problème de l'échange inter-domaines d'informations et d'autorisations relatives à une identité, là où les architectures classiques n'ont qu'un domaine de sécurité à gérer. Par ailleurs, il est important de noter que les différents standards mettent en jeu la même notion de *jeton de session*, dont la constitution, d'un standard à l'autre, diffère uniquement sur la forme. Ainsi, il est possible de réaliser, comme cela est proposé dans la deuxième partie de ce rapport, une architecture convergente prenant en compte cette proximité, et permettant de s'adapter, par exemple, aussi bien à la technologie SAML qu'au standard OpenID.

	Windows Cardspace (basé sur Infocard)	Liberty Alliance (basé sur SAML)	OpenID
Cercle de confiance	Non, mais nécessite une compatibilité des politiques de sécurité	Oui	Non
Public visé	Utilisateurs finaux	Entreprises	Utilisateurs finaux
Compatibilité WS	Oui	Oui	Non
Centré utilisateur	Oui	Non, et les opérations de transmission de données personnelles sont transparentes	Oui, et consentement demandé avant transmission de données personnelles
Sécurité (Authentification)	Résistant au phishing, mais faiblesse potentielle du protocole d'authentification utilisé	Dépend du protocole d'authentification	Dépend du protocole d'authentification, en général mot de passe donc peu sécurisé
Découverte de l'IdP	Simple	Difficile	Simple
Préservation des données personnelles	Risques de suivi des SP visités par l'IdP	Risques de suivi des SP visités par l'IdP	L'IdP connaît tous les SP visités. Risque de partage d'informations entre différents SP

Tableau 1 : Comparaison de trois solutions de fédération d'identité

2.4 Elaboration d'un modèle d'identité et fédération d'identité

Dans le cadre de notre objectif qu'est l'élaboration d'un nouveau modèle d'identité, la compatibilité avec les différents moyens de lier l'identité entre SP et IdP doit être la plus grande et la plus aisée possible. La fédération d'identité représente en effet un vecteur extrêmement conséquent permettant le déploiement rapide d'un modèle émergent qui ne nécessite pas de changement de configuration d'un parc entier de serveurs d'authentification ou de technologies usuellement supportées et déployées.

Inscrire l'élaboration d'un modèle d'identité dans la réalité suppose, entre autres contraintes, la prise en compte d'un déploiement effectif en dehors de serveurs de tests personnels. Pour cela, la

démarche consiste à implémenter un IdP s'appuyant sur l'une des technologies citées précédemment, puis à tester l'authentification auprès d'un SP sur la base de cette technologie. D'après les remarques formulées dans les sections précédentes, la technologie OpenID nous a semblé la plus adéquate à cet effet, grâce notamment à sa simplicité, sa légèreté, et l'absence de coûts que son déploiement engendre.

D'un point de vue strictement industriel, adhérer à un acteur majeur de la fédération d'identité tel que Liberty Alliance, en s'inscrivant notamment dans un cercle de confiance, pourrait avoir beaucoup plus de poids, mais nécessite un investissement prohibitif pour une simple preuve de concept. Toutefois, la compatibilité de la solution élaborée avec la technologie SAML sera illustrée dans la troisième partie de ce rapport qui traitera, dans le chapitre 6, de la mise en place d'un serveur de jetons SAML dans le contexte du Cloud Computing.

Cette recherche d'interopérabilité du modèle d'identité avec ces différents standards interdit par nature l'élaboration d'un format d'identité totalement nouveau – travail qui appartient aux consortiums et organismes de standardisation à même de tenter de l'imposer au sein de l'Internet actuel. Le cadre d'un travail de thèse impose nécessairement des vues plus modestes, sans quoi l'ambition de demeurer réaliste ne tiendrait pas. Par conséquent, l'élaboration d'un modèle d'identité numérique s'appuiera nécessairement sur l'utilisation de standards reconnus mais combinés de manière originale, en recherchant la nouveauté dans l'architecture de gestion sous-jacente davantage que dans le format lui-même. Une preuve de concept impliquant le déploiement d'un IdP de type OpenID s'inscrit donc entièrement dans cette orientation.

2.5 Conclusion

Nous avons pu mettre en évidence à travers ce chapitre différents types d'architecture de gestion de l'identité numérique, qui notamment s'opposent à l'architecture classique associant une identité à un serveur. Les apports en termes de simplicité de gestion pour l'utilisateur sont réels et conséquents, mais la complexité de développement et de déploiement ne l'est pas moins, bien que cela dépende du type d'architecture considérée. Les contraintes en termes de coûts et de risques relatifs à la sécurité et à la vie privée doivent être traitées avec beaucoup de rigueur, les enjeux d'une usurpation d'identité étant extrêmement importants.

Les différents produits et technologies mis au point par différents acteurs de l'identité permettent de fournir à l'heure actuelle la possibilité de répondre à des besoins précis par rapport à ces contraintes, c'est-à-dire à un compromis entre le niveau de sécurité et de garantie de vie privée offerts aux utilisateurs, et le prix à payer pour cela. Depuis OpenID, gratuit et simple mais offrant peu de garanties, jusqu'à Liberty Alliance, ou d'autres produits de fédération d'identité propriétaires, qui offrent une gestion généralement plus fine des problématiques de l'identité numérique mais qui sont nettement plus coûteux et plus lourds, les solutions en place sur le marché couvrent une grande partie des cas d'usage.

Indépendamment de cela, et malgré les enjeux majeurs qu'elles se proposent de traiter, les architectures de fédération d'identité restent toutefois critiquables d'un point de vue plus large. En effet, la création de cercles de confiance induit l'existence de communautés fermées et par conséquent difficiles d'accès, dont la finalité consisterait davantage à la mise en place d'un partenariat liant ses différents membres qu'à une véritable volonté de simplifier la gestion des identités pour les utilisateurs. Mais la suppression totale du principe du cercle de confiance n'est pas non plus une solution, car le caractère inter-domaines des informations échangées et des politiques de sécurité ne permet pas, sans relation préalable de confiance, d'accepter en toutes circonstances la délégation de l'authentification à un IdP dont le SP ne sait rien quant au sérieux de son administration. Dans tous les cas, la fédération d'identité se heurte à un paradoxe qui n'a pas trouvé, à l'heure actuelle, de solution satisfaisante.

Dans le cadre de ce travail de thèse, ainsi qu'il a été spécifié, nous préférons tirer partie de la technologie la plus aisée à déployer sur le Web, à savoir OpenID. Ses défauts, certes nombreux, peuvent trouver des réponses satisfaisantes, notamment du point de vue de la sécurité lors de la procédure d'authentification. Plus précisément, une authentification de type mutuelle par le protocole TLS serait vue d'un très bon œil par l'ensemble des SP souhaitant s'assurer de la mise en place de contre-mesures anti-phishing, tout en permettant à l'utilisateur une gestion plus aisée de son compte par la suppression du mot de passe. Par conséquent, les conclusions que nous avons tirées du premier chapitre de ce rapport restent valables dans ce contexte, gagnant même en pertinence. L'orientation amenée par l'ensemble de cette analyse, tant sur les modèles d'identité et d'authentification, que sur les architectures de fédération d'identité, nous amène donc à creuser plus en détail un modèle d'identité reposant sur le protocole TLS tout en bénéficiant d'un environnement de confiance propre à remédier aux réserves énoncées dans le premier chapitre. A cet effet, la carte EAP-TLS, qui sera le fondement de notre modèle d'identité, constitue le sujet du chapitre suivant.

Chapitre 3 : La carte à puce EAP-TLS

3.1 Introduction

Comme nous l'avons montré dans le Chapitre 1, le protocole TLS possède de nombreuses qualités, souvent insuffisamment exploitées. Son déploiement majeur, sous la forme d'une authentification à sens unique de la part du serveur vers le client, permet de réaliser le chiffrement des flux de données applicatifs et de réduire l'efficacité du phishing. Mais en omettant l'authentification mutuelle, qui soumet la création du tunnel sécurisé au succès de l'authentification de l'utilisateur via son certificat X509, ce déploiement n'élimine ni la faiblesse de l'authentification par mot de passe, ni le phishing – qui fonctionnera aussi longtemps qu'une donnée non éphémère tel qu'un mot de passe, transitant sur le réseau, servira à l'authentification – ni les attaques par MITM, dont nous avons vu qu'elles pouvaient compromettre de nombreuses procédures d'authentification, y compris les OTP.

Toutefois, nous avons également mis en évidence à propos de TLS un fait nettement moins répandu dans les analyses des faiblesses du protocole. Il s'agit de son exécution en environnement peu adapté à la conservation d'éléments sensibles tels que des clés secrètes ou privées. Or, l'absence d'environnement de confiance peut, au même titre que les attaques mentionnées plus haut, compromettre l'ensemble de la sécurité offerte par le protocole. L'utilisation de composants sécurisés tels que des cartes à puce permet alors de combler cette faille, au moins en partie. Les bénéfices apportés par ces composants dépendent de leurs propriétés, d'une part, mais aussi, et surtout, de la manière dont ils sont mis à contribution.

Dans la pratique, en effet, l'utilisation des environnements de confiance se limite le plus souvent au stockage d'éléments à longue durée de vie, comme une clé privée asymétrique. C'est notamment l'orientation prise par les « cartes PKI » qui inondent le marché des cartes à puce, et que nous étudierons plus spécifiquement dans ce chapitre. Néanmoins, comme nous l'avons vu au premier chapitre, une telle approche n'est pas suffisante. Pour aller au bout du raisonnement et assurer un très haut niveau de confiance, c'est l'ensemble du protocole TLS qui doit s'exécuter dans le composant dédié.

L'ensemble de ces observations nous a amenés à considérer ce que nous appellerons désormais la « carte EAP-TLS », qui désigne un composant sécurisé, sans limitation aux cartes à puce, embarquant notre implémentation du protocole EAP-TLS. Malgré tout, la carte à puce représente la très grande majorité des supports des implémentations que nous avons effectuées. C'est pourquoi nous nous attarderons plus particulièrement, dans ce rapport, sur les propriétés de la carte à puce et sur les différents types de cartes que nous avons testées.

Le but de ce chapitre est de décrire formellement la technologie EAP-TLS embarquée, de la différencier des traditionnelles « cartes PKI » -- qui désignent généralement des composants compatibles avec le standard PKCS #15[78] – et enfin de montrer que les performances de cette technologie sont compatibles avec une utilisation régulière de la part des utilisateurs finaux. Nous terminerons ainsi la première partie de ce rapport par une brève description du modèle d'identité que nous avons envisagé, qui reposera très largement sur cette technologie, et qui prendra en compte les conclusions tirées à l'issue des deux chapitres précédents.

3.2 La carte à puce

3.2.1 Généralités sur la carte à puce

L'histoire de la carte à puce remonte aux années 1970 où l'implication industrielle dans la microélectronique (Bull CP8, Schlumberger, etc.) a permis de concevoir les prémices d'un type de composant qui devait succéder aux cartes à bande magnétique, sur lesquels il était aisé de lire ou d'écrire des données. A l'heure actuelle, la carte à puce[68] peut être définie comme un type de micro-contrôleur sécurisé à bas prix mettant en place des contre-mesures visant à empêcher la modification de données stockées. Ces composants comportent un CPU, dont les architectures vont de 8 à 32 bits, quelques centaines de Ko de ROM qui contient le système d'exploitation du composant, de la RAM à hauteur d'environ 10 Ko, et une mémoire non volatile E²PROM dont la taille varie de 32 Ko à 1 Mo qui stocke la partie software (e.g. des Applets Javacard) ainsi que les clés cryptographiques. La carte à puce s'est aujourd'hui déployée à grande échelle (environ 7 milliards d'unité) dans de nombreux secteurs tels que la monétique (cartes EMV), la téléphonie mobile qui représente près de 75% du marché global de la carte à puce (cartes SIM, USIM), la médecine (cartes Vitale), le transport (cartes sans contact Navigo), etc.

La puce des cartes à contacts ne peut fonctionner qu'à l'aide d'un CAD (Card Acceptance Device), usuellement un lecteur de cartes, qui lui délivre de l'énergie et une horloge, ainsi qu'un lien de communication avec un terminal. Elle est constituée de huit contacts à même de gérer ces différents éléments, selon le standard ISO 7816-2[69]. La communication avec le terminal peut être un lien série ou bien une interface USB, avec un débit variant entre 9600 bits/s à quelques Mbits/s. Les paquets échangés entre la carte et le terminal s'appellent des APDUs (Application Protocol Data Units), d'une taille maximale de 256 octets, et dont la spécification détaillée est définie dans le standard ISO/IEC 7816. (cf. Annexe B)

La sécurité dans les cartes à puce est mise en place par un ensemble de contre-mesures physiques et logiques qui permettent d'empêcher, notamment, des attaques de type injection de fautes, consistant à introduire des erreurs volontaires dans l'exécution d'un programme afin d'en tirer des informations censées être secrètes, ou encore des attaques par canaux cachés (*side channel attacks*) qui visent à exploiter des données physiques ou temporelles produites par le composant (mesure de temps d'exécution, de consommation d'énergie pour un calcul donné par exemple) afin de

déduire, par exemple, un secret cryptographique. Ces attaques sont connues pour fonctionner sur des environnements classiques tels que les ordinateurs, dont les composants hardware ne sont en général pas pourvus de telles contre-mesures. Pour cette raison, les attaques à base de malware ou de Trojan ont beaucoup plus de latitude pour nuire dans ces environnements que dans une carte à puce, où de nombreux verrous supplémentaires sont posés.

Les cartes à puce fournissent ainsi un environnement de *confiance*, dans le sens défini par le Trusted Computing Group : « *Trust is the expectation that a device will behave in a particular manner for a specific purpose* » [70]. Ce consortium, fondé en 2003, a notamment défini les spécifications d'un composant nommé TPM (*Trusted Platform Module*) prévu pour être attaché à la carte mère des terminaux afin d'exécuter des fonctionnalités telles que le relevé de mesures d'intégrité ou le stockage sécurisé d'informations. Néanmoins, contrairement aux cartes à puce, les TPM ne sont pas prévus pour stocker des données applications spécifiques.

De fait, la plupart des cartes à puce sont équipées d'une machine virtuelle, de type Java ou .NET. Par conséquent, les langages de développement sur carte à puce étant très proches des langages de programmation courant, l'écriture d'applications critiques visant à être embarquées, puis exécutées dans cet environnement de confiance, est extrêmement aisé. Dans le cadre de ce travail de thèse, c'est la technologie Javacard 2.X qui a été privilégiée pour le développement de nos applications.

3.2.2 La technologie Javacard

Le langage Javacard est un sous-ensemble du langage Java, qui en limite certains types – comme les tableaux multidimensionnels ou les type *long*, *float*, *double* et *char* -- mais aussi la gestion de la création d'objets, qui ne peut avoir lieu au *runtime*. De fait, à ces restrictions près, le développement d'une application Javacard est très similaire à celui d'une application Java.

3.2.2.1 Compilation et installation des applications

La compilation d'une application comporte deux phases : la première est une compilation classique des différents fichiers qui produit des *.class*. Ensuite, un programme nommé convertisseur, qui implémente une partie de la machine virtuelle de la carte à puce, effectue les vérifications de compatibilité avec la version du langage Javacard supporté par la carte, puis lie l'ensemble des fichiers *.class* dans un unique fichier nommé *.cap*, qui constitue l'application à charger et installation dans la carte.

L'installation de l'application est réalisée par l'intermédiaire d'un logiciel présent sur le terminal dont la fonction est de segmenter l'application en APDUs, éventuellement chiffrées et signées, afin de transmettre les données à la carte, sur laquelle un *on-card installer* se chargera de traiter les APDUs reçues et d'installer l'application en conséquence.

3.2.2.2 L'Applet Javacard

Une application Javacard, représentée par l'ensemble des classes présentes dans le fichier *.cap*, est définie comme une extension de la classe Applet. Conformément au standard ISO 7816-5, elle possède obligatoirement un identifiant unique nommé AID (Application Identifier), composé de 5 octets obligatoires (*National Registered Application Provider*, RID) et d'un champ facultatif comportant de 0 à 11 octets (*Proprietary Application Identifier Extension*, PIX). Elle doit également comporter plusieurs méthodes parmi lesquelles (nous ne donnons pas les prototypes complets) :

- *install()* : Il s'agit de la méthode appelée par le *Javacard Runtime Environment* (JCRE) lors de l'installation de l'application. L'AID correspondant est passé en paramètre de la méthode. C'est à ce stade qu'est effectuée la création des objets définis pour cette application.

- *process()* : Il s'agit de la méthode permettant de traiter les APDUs reçues par la carte une fois que l'application a été sélectionnée. C'est dans cette méthode, véritable colonne vertébrale de l'application, que la gestion des différentes commandes propres à l'application peut être définie.

La norme Javacard permet par ailleurs le partage d'objets entre Applets. Un ensemble d'Applets appartenant à un même paquetage constitue un *contexte*. Le partage d'objets est autorisé au sein d'un même contexte, mais pas entre des Applets appartenant à des contextes différents, pour des raisons de sécurité.

3.2.2.3 Le Javacard Runtime environment

Le JCRE est un ensemble de composants présents dans la carte avec lequel le dialogue est établi par des APDUs. La figure 12 illustre et résume l'architecture générique d'une Javacard, et met en valeur les différents éléments constitutifs du JCRE, à savoir :

- *Installer* : Permet le chargement d'un Applet dans la carte.

- APIs : ensemble de classes permettant de fournir un cadre indispensable à l'exécution d'un Applet.

- Des extensions spécifiques à une ou plusieurs applications particulières.

- Un ensemble de classes systèmes (*system classes*) permettant de fournir des services de base, notamment la gestion des Applets et des transactions.

- La machine virtuelle (et méthodes natives associées).

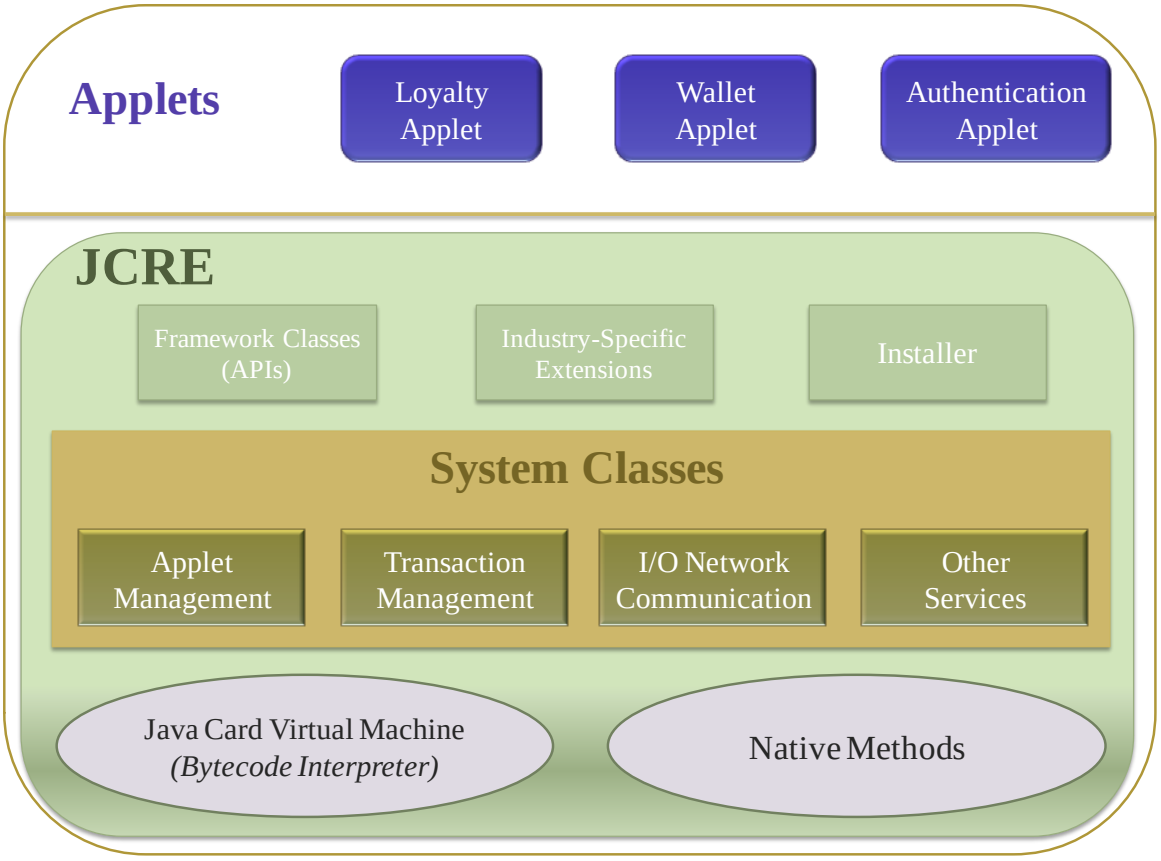

Figure 12 : Les composants du JCRE

3.2.3 Carte à puce et identité numérique

La carte à puce a permis de faire apparaître un modèle d'identité original par rapport à ceux qui ont été étudiés au cours du premier chapitre, et qui s'intéressaient avant tout à l'identité numérique au cœur du Web, et non aux réseaux accessibles via les terminaux mobiles. Or, ces modèles d'identités ont en commun le fait que l'utilisateur est pleinement détenteur et gestionnaire de son identité : c'est lui-même qui la crée et, en accord avec le modèle adopté, c'est lui-même qui sait comment fournir la preuve de son identité lorsque cela lui est demandé.

Mais des cartes à puce du type USIM, émises par des opérateurs mobiles de réseaux 3G, représentent un modèle où l'identité de l'utilisateur est gérée par une tierce partie. Dans ce paradigme, l'utilisateur accepte l'identité qui a été créée pour lui et sur laquelle il n'a aucune connaissance, notamment sur le secret incarné par une clé stockée dans la carte. La preuve de l'identité est réalisée sans l'intervention de l'utilisateur, qui n'a même pas connaissance de la procédure d'authentification.

L'orientation que nous avons prise, et qui nous amène sur un modèle d'identité reposant sur l'utilisation d'une carte à puce, nous permet assez naturellement de nous inscrire dans la convergence, puisque la carte à puce peut, par nature, se greffer aisément sur la plupart des terminaux modernes, i.e. des plateformes dites convergentes. Toutefois, pour être entièrement cohérents avec nos

ambitions, il nous faut également prendre en considération le modèle dont nous venons de parler, et nous assurer que le modèle d'identité que nous élaborerons reste compatible à la fois avec un type de gestion centrée utilisateur, et avec un type de gestion déléguée à une tierce partie.

3.3 L'architecture de la carte EAP-TLS

Le protocole EAP-TLS est défini comme un standard IETF [20], et vise à assurer le transport des messages TLS selon un mode datagramme. Contrairement au déploiement usuel de TLS, ce n'est donc pas la pile TCP/IP qui sert de support, ce qui présente de nombreux avantages relatifs à la simplicité de traitement et de déploiement.

Malgré cela, il reste possible de réaliser l'implémentation de TLS sur TCP/IP dans une carte à puce, comme le montre le brevet [71]. Ce choix se heurte malgré tout aux limites inhérentes aux protocoles TCP et IP, dont les failles de sécurité sont à présent bien connues[72]. Par conséquent, les avantages apportés par l'environnement de confiance qu'est la carte à puce sont remis en cause par la faiblesse des protocoles embarqués. Pour cette raison, qui s'ajoute aux précédentes, il nous semble d'autant plus judicieux de viser une implémentation de TLS sur EAP.

3.3.1 Rappels sur le protocole EAP

EAP[73] est un protocole réalisant une enveloppe permettant de transporter différents protocoles d'authentifications, et a été développé à l'origine pour être utilisé dans des connexions de type point à point avec le protocole PPP[74], mais il est à l'heure actuelle largement répandu dans les architectures sans fil de type 802.1X[75]. Dans cette architecture visant à restreindre l'accès des flux d'utilisateurs non authentifiés, plusieurs éléments essentiels doivent être déployés :

- Le *Supplicant* : Il s'agit la machine cherchant à accéder au réseau et qui désire s'authentifier.
- L'*Authenticator* : Il s'agit de l'entité réalisant le contrôle d'accès au réseau par blocage de port. Tant que l'authentification n'a pas été validée par le serveur, seuls les messages EAP visant à authentifier le *Supplicant* peuvent transiter sur le réseau. Au cours de cette phase, son rôle est simplement d'effectuer le relais des paquets transitant entre le *Supplicant* et le serveur d'authentification.
- L'*Authentication server* : Il s'agit d'un serveur, généralement de type RADIUS[76], dédié à l'authentification centralisée des utilisateurs du réseau. C'est cette entité qui est à même d'autoriser l'accès d'un utilisateur après validation de son authentification. Les messages EAP issus du *Supplicant* sont encapsulés dans le protocole RADIUS par l'*Authenticator* avant d'être transmis au serveur, et vice versa.

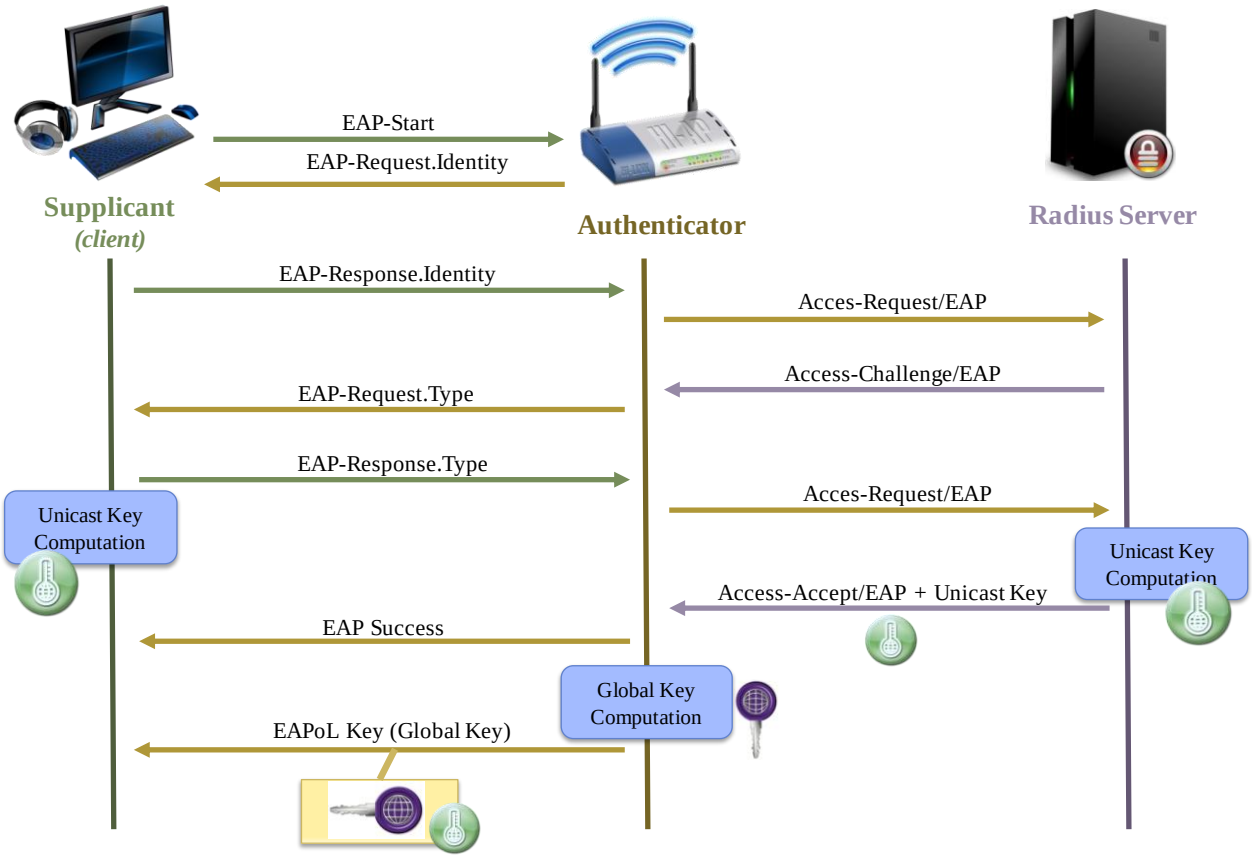

Figure 13 : Authentification dans une architecture 802.1X

Le schéma générique d'une authentification dans une architecture de type 802.1X est présenté dans la figure 13. Les échanges entre le serveur d'authentification RADIUS et l'*Authenticator* sont sécurisés par un secret partagé RADIUS utilisé pour adjoindre à chaque message RADIUS un HMAC-MD5, placé dans l'attribut RADIUS de type 80. Dans le cas où l'authentification est réussie, le *Supplicant* et le serveur d'authentification calculent une clé Unicast. Cette clé est retransmise à l'*Authenticator* avec le message *Access-Accept* du serveur d'authentification, et est utilisée pour la transmission au *Supplicant* d'une clé Globale nécessaire au chiffrement des paquets échangés entre le *Supplicant* et l'*Authenticator*.

Les méthodes d'authentification transportées dans EAP sont très nombreuses. Celle qui nous intéresse ici est TLS, mais la description formelle et exhaustive du protocole EAP-TLS n'apporterait que peu d'informations par rapport à l'étude respective de EAP et de TLS qui a déjà été effectuée. Nous nous intéressons par conséquent davantage aux modifications architecturales à apporter au protocole afin d'en faire une application pour l'embarqué.

3.3.2 Les contraintes du TLS embarqué

EAP-TLS impose quelques contraintes supplémentaires par rapport à TLS, et notamment celle de mettre en place une authentification mutuelle de manière obligatoire. Par ailleurs, il est nécessaire de fragmenter les messages TLS, dont le *Record*, habituellement limité en taille à 16 Ko, dépasse potentiellement la taille maximale des messages RADIUS (4 Ko) ou des trames 802.11 (2312 octets).

De manière générale, l'architecture 802.1X, dans le cadre de laquelle le protocole EAP-TLS est le plus souvent déployé, impose de nombreuses contraintes pour un déploiement visant le grand public. Pour cette raison même, nous renonçons à réaliser une implémentation complète de EAP-TLS en tant que protocole d'authentification de notre modèle d'identité. En revanche, nous conservons de EAP-TLS l'idée de transporter des messages TLS avec un mode datagramme, sans la lourdeur ni les failles de la pile TCP/IP, qu'il serait difficile, voire contre-productif, d'implémenter en sus de TLS pour un système embarqué. Nous en conservons aussi l'esprit, à savoir le fait d'imposer, et non de simplement proposer, une authentification mutuelle via TLS.

Embarquer le protocole TLS consisterait donc, d'après les remarques précédentes, à en implémenter une version transportée par des datagrammes EAP, sans pour autant déployer une architecture de type 802.1X. Par conséquent, si les paquets TLS sont encapsulés dans l'EAP depuis la carte à puce jusqu'au poste client, il faut ensuite que ce dernier envoie du TLS simple au serveur afin que la communication entre client et serveur reste standard et que l'utilisation du TLS embarqué soit transparente pour le serveur Web. Par ailleurs, il est nécessaire de pouvoir établir une liaison entre la carte à puce et le poste client. Cela concerne notamment les demandes de connexion en HTTPS de la part de l'utilisateur via son navigateur, qui doit appeler le TLS embarqué et non le TLS implémenté sur le poste client. L'ensemble de ces considérations amène donc logiquement à l'ajout d'un logiciel supplémentaire, que nous appellerons *proxy*, et qui doit tourner sur le terminal de l'utilisateur en tâche de fond.

Enfin, du point de vue du serveur, une configuration spécifique visant à obliger l'authentification mutuelle lors d'une session TLS doit être mise en place. Parmi l'ensemble des contraintes d'implantation du TLS embarqué, celle-ci représente clairement le frein le plus important à un déploiement massif de cette solution. Il n'est en effet pas possible d'imposer, de manière réaliste, une telle contrainte aux différents serveurs Web existants, du fait de l'impact que cela aurait sur l'ensemble des utilisateurs des sites concernés, utilisateurs alors incapables de s'authentifier sans certificat X509. C'est donc sur ce point, très précisément, que les architectures de fédération d'identité sont d'un grand intérêt pour nous : en déléguant l'authentification à un IdP, seul ce dernier doit être équipé de la configuration adéquate, les SP n'ayant pas besoin de modifier la leur.

3.3.3 L'identité EAP-TLS

L'intégration de TLS dans une carte à puce n'a de sens qu'avec l'adjonction de données correspondant à une identité, qui dans la plupart de nos cas d'utilisation est cliente, mais qui peut aussi bien être une identité serveur. Dès lors, nous avons défini un format d'identité, que nous appellerons par la suite « identité TLS » ou « conteneur d'identité », et qui contient les champs suivants :

- Type de l'identité (client ou serveur)
- Nom associé à l'identité
- Nom associé à l'EAP-Identity
- Ensemble des clés nécessaires à l'établissement de la session TLS (par exemple une clé privée RSA en mode CRT[13])
- Certificat associé à l'identité
- Optionnellement, le certificat du CA

Cette identité est stockée dans la mémoire non volatile E²PROM de la carte. Etant donné qu'elle contient des données critiques, la carte doit être protégée par un code PIN, qui doit être demandé à l'utilisateur avant toute opération menée avec la carte.

En tant que tel, ce format d'identité nous fournit une base extrêmement solide et concrète par rapport au modèle d'identité que nous souhaitons élaborer, à savoir un format reposant sur un certificat X509 visant à un déploiement d'une authentification mutuelle par TLS tout en bénéficiant de l'environnement de confiance qu'est la carte à puce. Autrement dit, l'ensemble des caractéristiques que nous avons posées comme étant souhaitables ou indispensables pour un modèle d'identité numérique sont vérifiées. Il reste à exploiter cette base afin de lui donner une existence réelle, i.e. imaginer la gestion d'un tel modèle et le cycle de vie de la carte à puce, mais aussi à s'assurer de la viabilité dudit modèle en mesurant les performances d'une authentification reposant sur ce dernier.

Auparavant, nous achevons la description de l'architecture de la carte EAP-TLS en explicitant le rôle du logiciel proxy ainsi qu'un scénario type de connexion TLS impliquant le TLS embarqué dans notre carte.

3.3.4 La gestion de l'EAP-TLS par la carte

Afin de pouvoir être pleinement opérationnelle, plusieurs commandes ISO 7816 ont été programmées dans la carte embarquant le protocole EAP-TLS ainsi qu'un conteneur d'identité, tel que décrit précédemment. Outre les commandes habituelles relatives à la sélection de l'Applet, au code

PIN et aux lectures/écritures de la mémoire E²PROM, l'octet INS (cf. annexe B) a ainsi permis de différencier les différents types d'APDUs spécifiques devant être traitées par la carte. Parmi ces commandes, nous citerons notamment :

- *Get Current Identity* (INS = 0x18) : Renvoie l'identité courante, i.e. le nom de l'identité chargée et convoquée lors d'une session TLS mutuelle.

- *Set Identity* (INS = 0x16, P1 = 0x80) : Spécifie l'identité devant être considérée comme l'identité courante. Le nom correspondant à un conteneur d'identité ayant été préalablement chargé doit être passé en paramètre de l'APDU.

- *Reset* (INS = 0x19, P1 = 0x10) : Réinitialise la machine à états EAP-TLS.

- *Process-EAP* (INS = 0x80, P1 = 0x00 ou 0x01) : APDU encapsulant un paquet EAP. Le paramètre P1 est fixé à 1 lorsque le paquet est fragmenté, et à 0 sinon. Le contenu EAP est traité conformément au standard en vigueur.

- *Get Ciphersuite* (INS = 0x82, P1 = 0xCC) : Renvoie la ciphersuite courante. Cette commande peut être désactivée selon les droits mis en œuvre dans la carte.

- *Get Key-Bloc* (INS = 0x82, P1 = 0xCA) : Renvoie le Key Bloc courant. Cette commande peut être désactivée selon les droits mis en œuvre dans la carte.

L'ensemble de ces commandes a permis de réaliser une gestion efficace du protocole EAP-TLS codé en tant que machine à états. Les deux dernières commandes, qui peuvent être jugées dangereuses du point de vue de la sécurité, sont dépendantes des droits fixés par l'administrateur de la carte, mais sont indispensables à la mise en œuvre de l'EAP-TLS embarqué tel qu'il a été conçu ici. Les raisons en sont données dans les paragraphes suivants de ce chapitre.

3.3.5 Le logiciel proxy

Le logiciel proxy a un rôle architectural essentiel dans le fonctionnement du TLS embarqué. Il s'agit du composant réalisant la glu logique entre le poste client et la carte à puce, mais aussi entre le serveur Web et la carte. Dans le cadre du travail que nous avons effectué, la communication avec la carte à puce est le plus souvent réalisée avec une interface PC/SC[77], qui est une API supportée aussi bien par Linux que par Windows et qui est dédiée à la découverte et l'utilisation de lecteurs de carte à puce. Cependant, tout type d'interface (notamment AT-CSIM (cf. section 3.5), port série, sans contact) est supporté.

Affecté à un port spécifique de la boucle locale du poste client (typiquement le port 8080), il réagit aux requêtes du navigateur Web du type *http://127.0.0.1:8080/~url=www.server.com* qu'il interprète comme une demande d'ouverture de session TLS vers server.com utilisant le TLS embarqué. Autrement dit, l'URL précédente est strictement équivalente, sémantiquement, à l'URL

https://www.server.com. Simplement, la pile TLS exécutée ne sera pas la même dans les deux cas. Par ailleurs, afin d'assurer la communication entre la carte à puce, qui embarque EAP-TLS, et le serveur, qui communique avec TLS, le logiciel proxy assure la traduction des messages qui transitent entre ces deux entités. L'architecture globale impliquant les différents éléments que sont la carte à puce, le poste client muni de son navigateur et du logiciel proxy, et le serveur Web distant, est représentée sur la figure 14. Seuls les échanges entre le logiciel proxy, présent sur le poste client, et le navigateur client, ne sont pas représentés

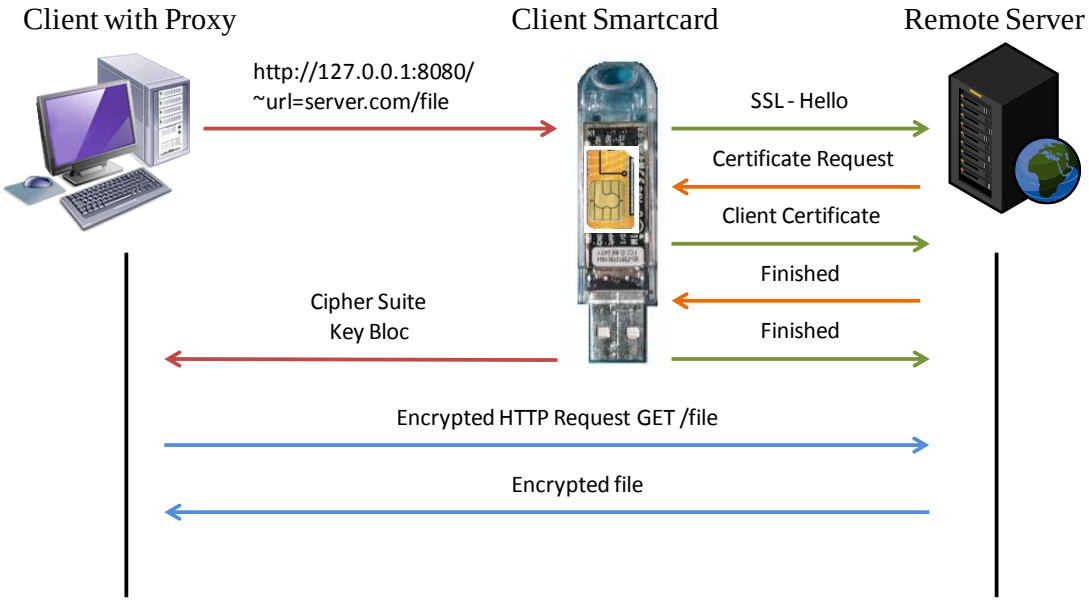

Figure 14 : Architecture pour le TLS embarqué

Le scénario de fonctionnement du logiciel proxy est alors le suivant :

1. Démarrage du logiciel proxy en tant que tâche de fond sur le port 8080, détection de la carte à puce cliente EAP-TLS et demande du code PIN à l'utilisateur. Une fois celui-ci fourni, récupération de l'identité TLS associée à la carte. Si aucune carte n'est détectée, si le code PIN est invalide, ou si aucune identité n'est trouvée, le démarrage du logiciel échoue.

2. Attente d'une demande de connexion de la part du navigateur client sur une URL au format spécifié précédemment. Envoi d'un message EAP-TLS-Start à la carte à puce, laquelle envoie ensuite un *Client Hello* encapsulé dans de l'EAP-TLS.

3. Réécriture du *Client Hello* en ôtant la partie EAP puis envoi de ce message au serveur Web distant par socket.

4. Relai de l'ensemble des messages du *Handshake* TLS entre le serveur et la carte à puce, jusqu'aux messages *ChangeCipherSpec* et *Finished*.

Une fois le *Handshake* effectué, la phase de chiffrement des données applicatives commence. Deux choix se posent alors :

- Réaliser le chiffrement et le déchiffrement des données à l'intérieur de la carte à puce, de manière à ce qu'aucune clé ne sorte de la carte, mais au détriment des performances dans le cas d'une taille importante de données applicatives.

- Exporter les clés temporaires (*Key Blocks*) une à une, avec la *ciphersuite* associée, suivant les connexions utilisateurs afin de réaliser le chiffrement et le déchiffrement des données sur le poste client, ce qui permet de profiter d'une plus grande puissance de calcul.

En réalité, les deux solutions sont équivalentes en termes de sécurité. En effet, le terminal client aura, dans les deux cas, accès aux données HTTP à la fois en clair et chiffrées. Avec un algorithme de chiffrement symétrique tel que RC4, cela permet même au terminal de retrouver la clé temporaire de session : si C_i désigne le message chiffré correspondant au clair M_i avec une clé de chiffrement RC4 K_i, le terminal connaît :

$$C_i = M_i \oplus K_i \quad \text{et} \quad M_i = C_i \oplus K_i$$, il peut donc en déduire $K_i = C_i \oplus M_i$.

Par ailleurs, avec un débit de moins de 2 Ko/s, les cartes à puce actuelles ne permettent pas de réaliser le chiffrement des données applicatives pour une utilisation courante sur des serveurs Web au contenu souvent chargé.

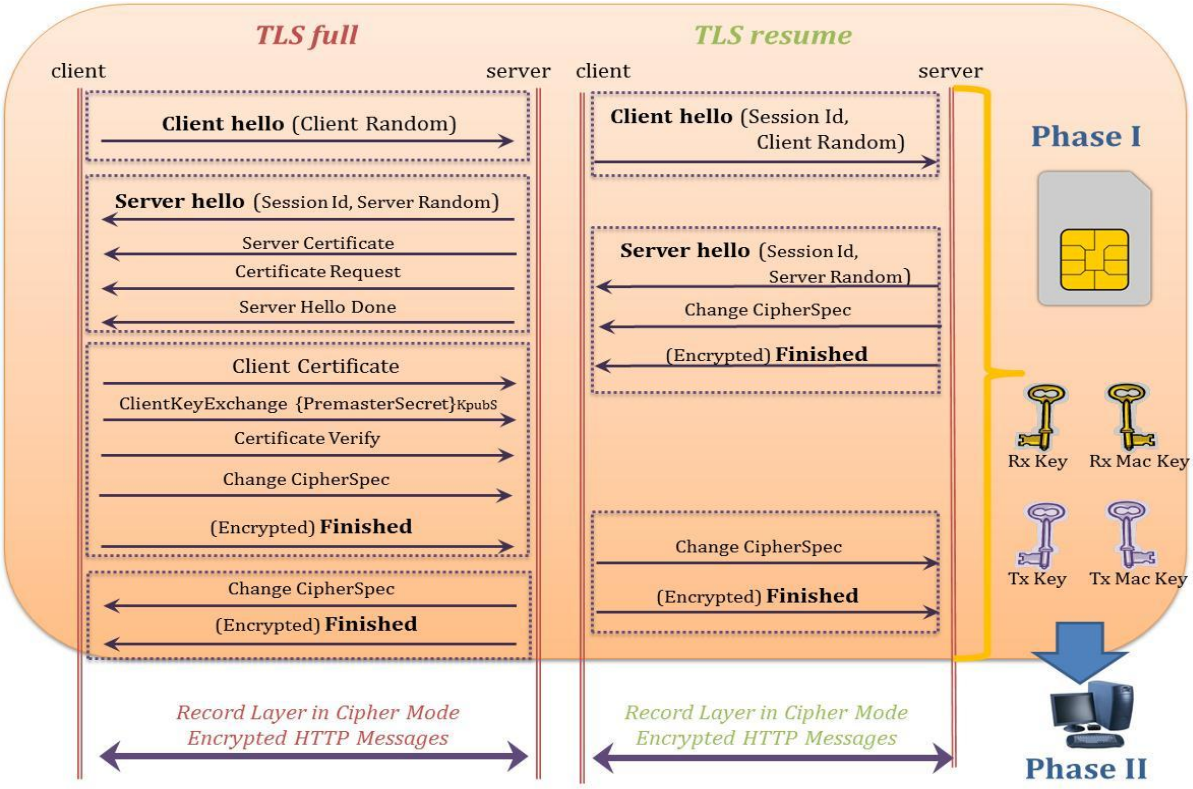

Figure 15 : La solution « TLS-Tandem »

Par conséquent, la seconde option est la plus avantageuse, et a fait l'objet d'un dépôt de brevet. Cette solution a été baptisée « TLS-Tandem », puisqu'elle tire avantage de la séparation en deux phases du protocole TLS : la première phase, le *Handshake*, est intégralement réalisée dans la carte, et la seconde phase, le chiffrement et déchiffrement des données applicatives, est déléguée au poste client qui récupère les clés de session temporaires au fur et à mesure des paquets échangés avec le serveur. La figure 15 donne une vue détaillée du déroulement d'un scénario d'une session TLS sur la base de la solution « TLS-Tandem », en mode *full* (à gauche) et en mode *resume* (à droite).

Le logiciel proxy a été écrit de manière à s'approcher le plus possible du paradigme de *OpenSSL*, dont l'API pour l'établissement d'une session TLS comporte trois fonctions principales de haut niveau qui sont : *SSL_connect()*, *SSL_read()* et *SSL_write()*. Le proxy implémente les mêmes fonctions. Simplement, la première demande une ouverture de session à la carte et laisse le proxy jouer le simple rôle d'intermédiaire, les deux autres fonctions étant en revanche exécutées sur le terminal client.

Il faut par ailleurs noter que, bien que la description précédente s'attache au cas d'une carte EAP-TLS en mode client, il est bien évidemment également possible de réaliser une carte EAP-TLS serveur, dont le principe reste similaire. Ainsi, notre application EAP-TLS contient systématiquement le code des versions client et serveur, de manière à pouvoir fournir une plus grande flexibilité à nos cartes. La troisième partie de ce rapport décrit ainsi plusieurs architectures faisant intervenir les deux modes à travers des cas d'utilisation concrets.

3.4 Comparaison avec les architectures classiques

3.4.1 Carte EAP-TLS et composants PKCS#15

Habituellement, la recherche d'un environnement de confiance dans le déploiement du protocole TLS repose sur l'adjonction d'un composant sécurisé calculant et stockant des clés privées avec éventuellement les certificats X509 correspondants. Il s'agit notamment des composants compatibles avec le standard PKCS #15[78].

Ce paradigme reflète une compréhension précoce des cartes à puces, dans laquelle la philosophie était de fournir l'environnement de confiance pour le stockage et le calcul des éléments cryptographiques sensibles dans un mode sans état[79]. Toutefois, depuis l'arrivée de la Javacard en 1996, les cartes à puce sont capables de supporter des protocoles à états tels que TLS. Malgré tout, les architectures classiques persistent dans la philosophie première, et fournissent à l'utilisateur des composant sécurisés spécialement dédiés à la cryptographie, et liés à la pile TLS du poste client par l'intermédiaire de périphériques logiciels spécifiques. De fait, de nombreuses contraintes relatives à la nature du système d'exploitation, du navigateur, et du composant lui-même rendent cette solution extrêmement hétérogène. Le manque de standardisation relatif à ce type de solutions rend difficile un déploiement massif auprès de millions d'utilisateurs. En comparaison, une architecture reposant sur

un logiciel proxy possède l'avantage de ne pas connaître ces différentes dépendances. Le poste client n'a pas davantage besoin de configuration préalable, puisque le logiciel proxy s'exécute sans installation. La figure 16 permet de visualiser la comparaison entre une architecture classique et celle que nous proposons.

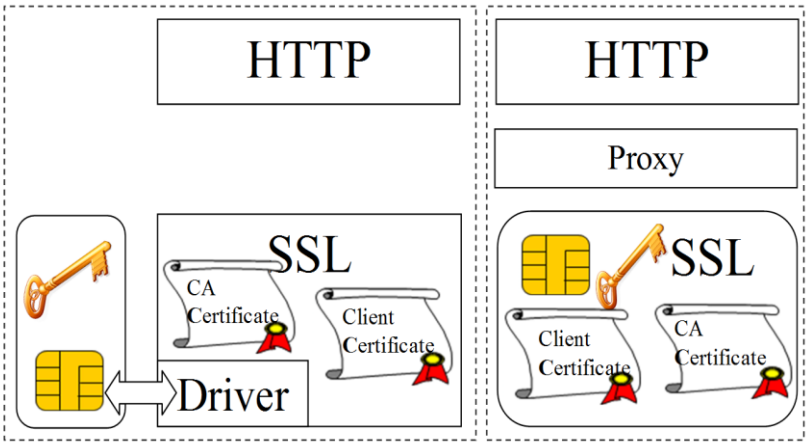

Figure 16 : Composant PKCS #15 comparé à la carte EAP-TLS

3.4.2 Les avantages sécuritaires de la carte EAP-TLS

Du point de vue de la sécurité, la différence essentielle entre les deux architectures est celle relative à la mise à contribution de l'environnement de confiance. Dans le cas des architectures classiques, le protocole TLS s'exécute par définition sur le poste client, et il existe dès lors une scission entre l'entité qui s'authentifie (i.e. le poste client, vulnérable) et l'élément sécurisé prouvant l'identité (i.e. le composant contenant la clé privée et réalisant les calculs cryptographiques). Par conséquent, l'authentification mutuelle n'est pas parfaite. A cela s'ajoute le problème, déjà mentionné dans le premier chapitre, du stockage du *Master Secret* d'une session TLS sur le poste client, alors qu'il s'agit d'une donnée capitale pour un attaquant. Par contraste, l'architecture que nous proposons réalise une authentification mutuelle complète entre la carte EAP-TLS de l'utilisateur et le serveur Web distant. Le poste client ne peut profiter de cette connexion sécurisée que s'il reçoit de la carte les clés temporaires successives et la *ciphersuite* associée, autrement dit si le lien entre le poste client et la carte EAP-TLS est établi, ce qui est précisément le rôle du code PIN utilisateur – qui assure au logiciel proxy que l'utilisateur du poste client est bien le possesseur de la carte EAP-TLS.

Par ailleurs, la réécriture complète du protocole TLS permet d'y ajouter des options ne s'opposant pas strictement au standard en vigueur, telles que la protection de l'identité du client mise en place par le chiffrement du certificat X509 de l'utilisateur, décrite dans [80]. De la même manière, l'échange de clés par l'algorithme RSA implémente le plus souvent la norme PKCS#1 v1.5[81]. Or, comme il a été démontré dans [83], cette norme présente des failles importantes permettant notamment à un attaquant de réaliser une opération impliquant la clé privée de la victime sans pour autant l'avoir récupérée. Il serait alors possible d'implémenter la norme PKCS#1 v2.1[82], la plus

récente à ce jour, et qui est résistante à cette attaque. Cela n'a toutefois pas été réalisé, et ce pour deux raisons principales : avant tout, l'échange de clés par RSA tend à être délaissé au profit de l'utilisation de la cryptographie par courbes elliptiques (ECC), ce qui limite l'intérêt d'un tel ajout pour les perspectives futures. Par ailleurs, cette norme n'est pas préconisée par l'IETF à travers les dernières versions du protocole TLS, dans lesquelles la référence à la norme PKCS #1 v1.5 est explicite – ce qui limite les possibilités de déploiement de l'ajout en question. Il en va de même des calculs MD5-SHA-1 de la PRF, qui ont été officiellement remplacés par l'algorithme SHA-256 dans TLS v.1.2[84], version que peu de serveurs implémentent. Dans ces derniers exemples, le cas d'utilisation le plus adapté reste celui du M2M, où deux composants embarquant la même application peuvent communiquer avec des standards récents sans craindre de problèmes d'incompatibilité. Dans la troisième partie de ce rapport, nous détaillerons deux exemples où une telle architecture, qui reste marginale mais qui possède un véritable sens, est déployée.

Malgré tout, il est important de souligner que notre architecture comporte également une faiblesse. L'utilisation d'un logiciel proxy implique un problème de confiance lié à l'intégrité du logiciel. Le risque pour l'utilisateur consiste ainsi à se retrouver avec un logiciel contrefait se faisant passer pour le logiciel proxy. Etant donné qu'il s'agit d'un élément central de l'architecture, qui notamment gère l'interface avec la carte à puce, un logiciel vérolé ou contrefait peut conduire dans le pire des cas à un détournement de session. Des solutions reposant sur une vérification de signature du logiciel ou sur un téléchargement à partir de serveurs sécurisés pourraient par exemple permettre de contourner ce problème.

3.5 Déploiements de la carte EAP-TLS

La carte EAP-TLS peut faire l'objet de deux types de déploiement selon le contexte d'utilisation :

- Déploiement pour ordinateurs fixes ou portables, notebooks, etc.
- Déploiement pour terminaux mobiles tels que les Smartphones.

Dans le premier cas, la carte doit être associée au poste client par l'intermédiaire d'un lecteur dédié, que l'utilisateur doit donc posséder. Néanmoins, il est irréaliste d'attendre qu'un tel paradigme puisse se répandre, car les utilisateurs n'utilisent jamais, en pratique, de lecteur de carte. Par conséquent, la forme la plus adaptée à un emploi quotidien dans le contexte d'ordinateurs fixes ou portables est celle d'un jeton USB intégrant un lecteur de cartes au format SIM et muni de mémoire Flash, typiquement pour stocker le logiciel proxy que l'utilisateur n'a plus qu'à exécuter. La carte EAP-TLS fonctionne alors selon un mode *plug-and-play*.

Dans le second cas, nous tirons partie du fait que la plupart des terminaux mobiles intègrent une carte SIM, dont une large partie (estimée à environ 1 milliard) embarque une machine virtuelle Java. Dès lors, notre application EAP-TLS, codée en Javacard, est naturellement compatible avec ces

cartes. Les cartes SIM peuvent être divisées en deux catégories : les cartes SIM dédiées au réseau GSM, et les cartes dédiées aux réseaux 3G ou 4G, nommées USIM. Les équipements relatifs au réseau ne sont pas gérés de la même manière dans les deux cas :

- Les cartes SIM gèrent les fonctionnalités réseau dans l'OS de la carte. Une application Javacard peut s'exécuter sans gêner les fonctionnalités GSM, mais deux applications Javacard ne peuvent être exécutées en même temps.

- Les cartes USIM gèrent les fonctionnalités réseaux par l'intermédiaire d'une application Javacard. Toutefois, la notion de « canal logique » introduite dans le standard ISO7816 permet jusqu'à l'exécution parallèle de quatre applications Javacard.

Il se pose alors la question de la communication entre le logiciel proxy, embarqué cette fois sur le terminal mobile en tant que *Middlet*, et la carte (U)SIM. Les systèmes d'exploitation orientés Java, tels que Symbian, prennent en charge des API comme la *Java Specification Request* (JSR) 177 (« *Security and Trust Services API for J2ME* ») permettant l'échange d'APDUs entre le terminal mobile et la carte. Par ailleurs, plusieurs terminaux mobiles et certains types de jetons USB pour la 3G (par exemple un modem HSDPA comportant une carte USIM) supportent les commandes AT-CSIM[85], qui sont une extension optionnelle des fonctionnalités AT offertes par la plupart des modems. Les commandes AT-CSIM permettent notamment l'échange d'APDUs avec les cartes (U)SIM.

Dans le cadre de nos travaux, nous avons développé des logiciels proxy adaptés à chacun des deux cas d'utilisation énoncés ci-dessus, aussi bien avec une carte SIM qu'avec une carte USIM, cette dernière étant équipée de deux canaux logiques.

3.6 Analyse de performances

3.6.1 Mesures sur Javacard GX4 et carte USIM

Notre implémentation de EAP-TLS repose sur la *ciphersuite* TLS_RSA_WITH_RC4_128_MD5 induisant l'échange du *premastersecret* par un chiffrement asymétrique RSA 1024 bits. Ce paramètre étant fixé, nous avons pu tester les performances de deux composants commerciaux, l'un étant de type Javacard Gemalto TOP IM GX4[112], carte à processeur 32 bits mais limitée à 800 octets de RAM, l'autre étant une carte Morpho USIM 32 bits. Les temps de calcul pour les différentes opérations cryptographiques élémentaires sont donnés dans le tableau 2.

Dans le cas d'une session TLS avec authentification mutuelle en mode *full*, les calculs cryptographiques effectués lors du *Handshake* comprennent 230 blocs MD5 et 230 blocs SHA-1, une opération de chiffrement à clé privée RSA (signature), une opération de chiffrement à clé publique RSA et une opération de déchiffrement à clé publique RSA. Par ailleurs, environ 2500 octets de données sont échangés. A l'aide du tableau 2, nous pouvons alors écrire la décomposition du temps global nécessaire pour une session en mode *full* (où N désigne le nombre d'octets de données

échangées, et T_{SF} et T_{SR} les temps de calcul résiduels liés à l'exécution des instructions de notre application Javacard, respectivement en mode *full* et en mode *resume*) :

$$T_{full} = 230 (T_{MD5}+T_{SHA1}) + T_{RSAPRIV} + 2 T_{RSAPUB} + N T_{IO} + T_{SF}$$

Dans le cas du mode *resume*, les calculs se réduisent à 75 blocs MD5 et 75 blocs SHA-1. Les données échangées n'excèdent alors pas 250 octets environ. Le temps global nécessaire pour une session en mode *resume* se décompose alors ainsi :

$$T_{resume} = 75 (T_{MD5}+T_{SHA1}) + N. T_{IO} + T_{SR}$$

Les mesures réalisées nous donnent alors les temps suivants :

- Pour la Javacard :
  T_{full} = 2,0s (et T_{SF} = 0,7s)
  T_{resume}= 0,50s (et T_{SR}= 0,35s)
- Pour la carte USIM :
  T_{full} = 4,3s (et T_{SF}= 2,65s)
  T_{resume}=1,65s (et T_{SR}= 1,45s)

Device	T_{MD5} /bloc 64 B	T_{SHA1} /bloc 64 B	$T_{RSA\ PUB}$ 128 B	$T_{RSA\ PRIV}$ 128 B	T_{3DES} /bloc 8 B	T_{IO} /byte
Javacard	0,49	0,93	25	560	2,10	0,17
USIM	2,60	3,70	150	290	5,20	0,24

Tableau 2 : Performances de composants commerciaux sur des algorithmes basiques (ms)

Ces résultats, obtenus après optimisation du code de l'application EAP-TLS – notamment par la maximisation de l'utilisation de la RAM plutôt que de l'EEPROM – montrent que l'utilisation d'une carte EAP-TLS est réaliste, y compris dans le contexte d'un usage courant dans la vie quotidienne des utilisateurs. En effet, même si l'exemple de la carte USIM montre un temps relativement élevé pour une session en mode *full*, la très grande majorité des sessions TLS réalisées sont en mode *resume*, où les temps restent très acceptables dans les deux cas.

3.6.2 Mesures de connexions TCP parallèles

Une autre mesure, présentée dans [86] montre le traitement de plusieurs connexions TCP en parallèle. L'utilisation d'une page de test contenant une dizaine d'images amène naturellement, après une première connexion sur la page, à l'émission de requêtes au serveur sur ces différents objets, chacun faisant l'objet d'un préalable *Handshake* via le TLS embarqué, avec donc pour chacun un numéro de session. Les mesures obtenues sont exposées sur la figure 17. La session 1 correspond au chargement de la page de test, et les sessions 2, 3, 4 et 5 correspondent aux 4 premières

connexions TCP traitées par le logiciel proxy. Usuellement, les navigateurs lancent un nombre de connexions TCP parallèles compris entre 2 et 4. Dans le cas présent, les mesures indiquent qu'il n'existe pas plus de deux connexions simultanées.

Figure 17 : Mesures de performances sur 5 connexions TCP parallèles

Conformément au scénario explicité dans la section 3.3.5, après réception d'une demande de connexion TCP par le logiciel proxy, celui-ci attend la disponibilité de l'application de la carte à puce (noté par 1 – *Card waiting*), puis il soumet une ouverture de session TLS à la carte et agit comme relais avec le serveur pour la réalisation du *Handshake* (noté par 2 – *Card computing*), et enfin, il récupère le *Key Block* courant afin de réaliser le chiffrement et le déchiffrement des données applicatives, directement sur le poste client (noté par 3 – *Data transfer*). L'allocation par le proxy de la carte EAP-TLS aux différentes sessions initiées par le navigateur est réalisée de manière aléatoire : dans notre cas, l'ordre d'allocation est ainsi 1,2,4,3,5. Les mesures montrent que la très grande partie du temps consommé tient aux deux premières phases (11,4s sur 12,1s de temps total, soit 94%), et que la durée des opérations réalisées par le logiciel proxy est négligeable du point de vue des performances.

Il faut noter que le composant utilisé pour cette expérience est une Javacard dont les performances, moins bonnes que celles que nous avons mises en valeur précédemment, sont proches de celles de la carte USIM que nous avons testée : T_{full} = 4,2s et T_{resume}= 1,8s.

3.7 Transposition à un composant 8 bits ST23XXXX et comparaison de performances

La technologie constituée par le TLS embarqué ne se limite pas à des supports de type cartes à puce, mais peut être implantée dans tout type de microcontrôleur possédant des contre-mesures sécuritaires similaires aux cartes à puce. Bien que la question du passage à l'échelle de la solution

présentée dans ce rapport relève davantage de l'industrie que de ce travail de thèse, l'adaptation du code de l'EAP-TLS, à l'origine porté sur des Javacards, à des composants à bas coût permettrait de soutenir la possibilité d'un déploiement massif, pour peu que les performances mesurées restent intéressantes. Ainsi, un travail de transposition vers un microcontrôleur sécurisé 8 bits ST23XXXX, suivi de mesures de performances, a permis de prendre en considération l'intérêt de disposer d'un tel composant relativement à ceux étudiés dans la section précédente.

Ce microcontrôleur possède les caractéristiques suivantes :

- Un processeur 8 bits de type ST23, possédant toutefois également des registres de 16 bits
- Un OS intégré dans une ROM de 210 Ko
- Une RAM de 6,2 Ko, dont environ 5,0 Ko utilisables pour une application externe
- Une EEPROM de 66 Ko
- Un accélérateur cryptographique disposant d'une RAM dédiée de 2,0 Ko
- Plusieurs contre-mesures physiques et logiques, ainsi qu'un TRNG (*True Random Number Generator*) certifié

L'application EAP-TLS trouve naturellement sa place parmi la mémoire EEPROM du composant, bien assez conséquente pour accueillir un code d'environ 20 Ko. Le développement y est réalisé en langage C, avec plusieurs extensions permettant de gérer le plus finement possible les différents segments de la mémoire. Un émulateur dédié permet de tester l'exécution du code avant son chargement dans le composant proprement dit, afin de protéger ce dernier.

La difficulté la plus importante rencontrée dans ce travail a consisté à mettre à profit l'ensemble de la RAM et à utiliser le moins possible l'EEPROM, beaucoup plus coûteuse en temps pour les opérations d'écriture, pour y stocker des variables volatiles par manque de place dans la RAM. En réalité, sur les 5,0 Ko disponibles, plusieurs centaines d'octets étaient déjà utilisés par une petite application de démonstration déjà présente dans le composant. Environ 500 octets sont également dédiés à la pile. Enfin, une session TLS en mode *full* nécessite la conservation de l'ensemble des messages échangés durant le *Handshake*, jusqu'au calcul du message *Finished*, ce qui représente environ 3,0 Ko. Par conséquent, il n'est possible d'allouer que très peu d'espace pour le reste de l'application.

Malgré cela, le code a été optimisé de manière à ne pas utiliser l'EEPROM de manière inadéquate, ce qui a permis d'obtenir les meilleures performances à l'exécution. En effet, à titre de comparaison, une opération d'écriture dans l'EEPROM (par bloc de 64 octets) nécessite 1,5 ms, alors que la même opération dans la RAM nécessite de l'ordre de 6,4 µs pour 64 octets, et 100 ns pour 1 octet – soit *a minima* une différence d'un facteur 200. Néanmoins, il n'a pas été possible de modifier l'accélérateur cryptographique du composant, naturellement apte à optimiser les calculs des algorithmes RSA et SHA-1 (déjà implémentés dans l'application de démonstration), mais non adapté à MD5, qui a dû être entièrement codé à la main afin de tirer le plus possible avantage de l'accélérateur.

Les caractéristiques temporelles du composant sont résumées dans le tableau 3. De manière générale, les caractéristiques cryptographiques se situent dans la moyenne des deux composants étudiés dans la section précédente, à l'exception du temps de calcul pour une opération RSA avec la clé privée, particulièrement court ici.

T_{MD5} /bloc 64 B (ms)	T_{SHA1} /bloc 64 B (ms)	$T_{RSA\ PUB}$ 128 B (ms)	$T_{RSA\ PRIV}$ 128 B (ms)	$T_{WRITE\ EEPROM}$ /64 B (ms)	$T_{WRITE\ RAM}$ /byte (ns)	T_{IO} /byte (ms)
1,32	1,48	42	96	1,5	100	0,21

Tableau 3 : Performances d'un microcontrôleur sécurisé 8 bits (ms)

Les performances mesurées pour l'établissement d'une session TLS, respectivement en mode *full* et *resume*, sont les suivantes :

- T_{full} = 1,9s
- T_{resume} = 0,5s

Par conséquent, ce composant fournit des performances équivalentes aux meilleurs temps obtenus avec la Javacard GX4. Ce résultat illustre donc bien, d'une part, la pertinence du choix d'un composant spécialisé à bas coût tel que celui-ci dans le cas d'un passage à l'échelle conséquent impliquant un déploiement massif, et d'autre part, la possibilité d'obtenir des performances satisfaisantes à partir de composants dotés de registres et de mémoires de faible tailles, moyennant un effort particulier dans l'écriture du code.

3.8 Conclusion

Dans l'optique d'offrir une solution crédible aux défauts relevés dans l'état actuel des systèmes d'authentification et des modèles d'identités numériques, la carte EAP-TLS réunit de nombreux avantages. S'appuyant sur une architecture relativement simple, elle permet de tirer le meilleur parti de l'environnement de confiance que constitue le composant sécurisé et prend en compte les capacités des cartes à puce actuelles capables de réaliser des calculs cryptographiques dans un mode à états, en embarquant un protocole entier, par opposition avec les architectures classiques telles que celles reposant sur des composants compatibles avec le standard PKCS #15.

Les contraintes d'implantation sont très peu présentes du point de vue du poste client, où la carte fonctionne en mode plug-and-play, mais sont plus importantes du point de vue du serveur, dont la configuration empêcherait ce dernier de rester compatible avec un déploiement classique de TLS, i.e. avec une authentification à sens unique. Les architectures de fédération d'identité, ou plus précisément le fondement technologique sur lequel elles s'appuient, à savoir un jeton de session inter-

domaines, permettent de s'affranchir de cette contrainte et contribuent à ancrer le déploiement d'une telle solution dans la réalité. De plus, la personnalisation de l'implémentation du protocole TLS nous permet de renforcer le protocole en proposant une protection de l'identité client via le chiffrement du certificat.

Les performances affichées suite aux tests menés sur divers composants montrent par ailleurs la compatibilité de la carte EAP-TLS avec les contraintes d'une utilisation courante de cette technologie. Le surcoût temporel engendré par la réalisation du *Handshake* TLS à l'intérieur du composant sécurisé est surtout pénalisant dans les cas où la plupart des sessions TLS réalisées sont en mode *full*, mais dans le cas de la navigation usuelle sur Internet, cela arrive très rarement. Les requêtes vers plusieurs pages sécurisées d'un même serveur sont en effet généralement traitées par le mode *resume*, pour lequel le surcoût est très faible, voire négligeable, selon le composant utilisé.

Du point de vue de l'identité numérique, cette technologie dispose de l'ensemble des caractéristiques jugées souhaitables pour le déploiement d'un modèle d'identité numérique associé à une authentification forte. L'adaptation à la convergence est assurée, d'une part, grâce à l'utilisation du standard extrêmement répandu qu'est le protocole TLS, et d'autre part grâce au support de type carte à puce qui permet de viser aussi bien les réseaux fixes que mobiles. Il reste alors à définir de manière plus formelle un système de gestion associé à ce modèle d'identité, afin de dégager un cycle de vie associé au composant utilisateur dans lequel seront stockées ses différentes identités. L'objet de la deuxième partie de ce rapport est précisément de répondre à cette problématique tout en montrant que les solutions proposées ont été testées dans des situations réelles avec des performances acceptables.

Partie II : Définition et gestion d’un modèle d’identité numérique fondé sur la carte EAP-TLS

Chapitre 4 : Système de gestion de l'identité TLS

4.1 Introduction

L'environnement des réseaux fixes et mobiles permet aux utilisateurs d'accéder à de nombreux services par l'intermédiaire de leur identité numérique. Comme il a été souligné auparavant, la nature de cette dernière, ainsi que la procédure d'authentification afférente, ont de nombreux impacts quant à la qualité de l'accès à ces services. De plus, la place prépondérante occupée par les réseaux sociaux, et l'explosion du nombre de comptes utilisateurs – et de mots de passe associés – auprès des divers fournisseurs de services, tend à conduire les utilisateurs à abandonner le pseudonymat, qui était pourtant en vogue depuis le début des années 2000. Facebook soutient cette mouvance au point d'interdire, dans ses conditions d'utilisation, la création d'un profil personnel associé à un pseudonyme. Ce comportement s'explique par une raison évidente : les données personnelles des utilisateurs ont acquis une valeur extrêmement forte, commercialement parlant. Les enjeux liés à l'usurpation d'identité ont donc évolué. Si à l'origine, les conséquences pour l'utilisateur pouvaient avant tout prendre la forme d'actions, d'écrits, ou d'achats non désirés, à présent l'accès aux données personnelles est devenu une fin en soi.

Par conséquent, les modèles d'identité reposant sur une authentification faible sont, plus encore qu'auparavant, exposés à ces risques. Si les méthodes d'authentification à deux facteurs visant à protéger des opérations sensibles, par exemple des virements bancaires, permettent de mettre un frein à l'impact de l'usurpation d'identité, elles ne prennent pas en compte l'importance qu'ont acquise les fichiers de données personnelles des utilisateurs. Pour reprendre l'exemple précédent, un attaquant qui pourrait accéder aux comptes bancaires d'un utilisateur possède une information extrêmement sensible quand bien même il serait incapable de réaliser une transaction financière. Dans ce contexte, il est clair que l'élaboration d'un modèle d'identité numérique n'a de sens que par une association avec une authentification forte, comme nous l'avons déjà souligné, mais également par une gestion des identités sous-jacente extrêmement solide, et donc peu propice à la fuite de données personnelles.

Nous décrivons dans ce chapitre notre proposition de modèle d'identité, basé sur la carte EAP-TLS décrite au chapitre précédent. Nous en décrivons également la gestion par un serveur d'identités qui centraliserait les informations des utilisateurs, tout en en sécurisant l'accès par l'assurance que seul le possesseur de la carte est en train d'effectuer la demande de connexion. Après avoir largement insisté sur la définition de l'identité numérique comme l'association entre un format de données et une procédure d'authentification, il nous faut à présent, en effet, prendre en compte une dimension supplémentaire, à savoir la gestion de l'identité. Cette dimension nous amène à prendre en compte, dès la conception du modèle d'identité, la définition des services proposés par

ledit modèle et son système de gestion, mais aussi la manière dont les identités seront créées, mises à jour, supprimées, en relation avec le support du format d'identité. Une mauvaise gestion des identités peut être la source de failles de sécurité permettant à un attaquant d'usurper des identités, ou bien d'accéder frauduleusement à des données personnelles – ce dont nous chercherons à nous prémunir, dans un cas comme dans l'autre.

En termes de services, le modèle d'identité que nous proposons vise bien entendu à fournir une authentification forte, mais également à concilier les contraintes légales, demandant une association entre l'identité numérique et l'identité réelle, et la promotion du pseudonymat auprès des utilisateurs, à présent débarrassés de la difficulté de gestion de comptes à base de mots de passe. L'élément sensible qui perdure alors, le seul capable de faire le lien entre les différentes identités d'un utilisateur, et le seul capable d'accéder à l'ensemble des informations personnelles, est alors le serveur d'identités. Bien que, dans le cadre de ce travail de thèse, celui-ci soit presque entièrement développé sous la forme d'un serveur Web, les formes qu'il est susceptible de prendre sont nombreuses, et dépendent de la manière dont il est administré. Ce chapitre étant dédié à une description des fonctionnalités de ce serveur, nous illustrerons dans le chapitre suivant ce point par un cas concret, celui d'une gestion par OTA.

Enfin, nous décrivons le lien réalisé avec la fédération d'identités par l'intermédiaire d'une plateforme OpenID sur laquelle le seul mode d'authentification acceptable est celui de l'authentification mutuelle TLS réalisée à partir de notre carte EAP-TLS. De cette manière, il est possible de réaliser une preuve de concept fonctionnelle chez tout fournisseur de service intégrant une interface d'authentification du type OpenID.

4.2 La carte à identités TLS

4.2.1 Cas d'une carte à une identité

Nous élaborons notre modèle d'identité numérique en nous appuyant sur le principe de la carte EAP-TLS, d'après les arguments que nous avons déjà exposés dans le chapitre précédent. Le format exact de cette identité est celui du conteneur explicité à la section 3.3.3. Néanmoins, en tant que tel, ce modèle est difficilement exploitable pour une gestion approfondie des identités utilisateurs. En effet, si une carte ne contient qu'un seul conteneur susceptible de fournir une identité TLS, c'est-à-dire un certificat X509 et les clés associées, deux cas peuvent se présenter :

1. L'utilisateur reçoit une carte EAP-TLS vierge, i.e. sans conteneur déjà chargé. Dans ce cas, la gestion consistera à le laisser générer une identité qui devra être chargée dans sa carte.

2. L'utilisateur reçoit une carte EAP-TLS dans laquelle un conteneur *racine* a déjà été chargé, sans nécessairement de rapport avec la véritable identité de l'utilisateur, et dont la fonction serait de pouvoir certifier la carte auprès du serveur d'identités.

Chacun de ces deux cas conduit à une impasse. En effet, dans le premier cas, aucune contre-mesure n'est mise en place afin de vérifier l'authenticité de la carte utilisateur. Il est alors aisé de réaliser une contrefaçon en chargeant simplement dans un composant une application réalisant les mêmes fonctionnalités que la carte EAP-TLS. De même, un attaquant pourra aisément charger lui-même une identité dans son composant, une fois le format de celui-ci analysé, sans avoir à déclarer cette dernière au serveur d'identités. Bâtir une gestion des identités sur un tel modèle semble donc loin d'être optimal.

Dans le second cas, le conteneur *racine* a pour vocation de résoudre le problème précédent. Il contiendrait, en guise de CN du certificat X509, un numéro de série unique qui serait placé dans une liste blanche, de manière à ce que le serveur puisse vérifier la validité de la carte fournie par l'utilisateur. Pour autant, cette vérification étant effectuée, il faudrait, pour rester cohérent, interdire à l'utilisateur de réécrire ce conteneur pour y placer une identité qu'il aurait générée. En effet, une telle action ne permettrait pas au serveur de vérification ultérieure de la validité de la carte utilisateur. Bien qu'il soit possible de mettre en place une vérification de l'identité générée sur la base, par exemple, de la clé publique du certificat associé, il est possible à l'utilisateur de charger cette identité dans d'autres composants que le sien. Autrement dit, le lien entre l'identité réelle de l'utilisateur et l'identité générée, et donc entre l'utilisateur et sa carte, est rompu. Par conséquent, la gestion des identités utilisateurs ne peut être opérée convenablement selon ce modèle. Quant à considérer l'interdiction de générer une identité afin d'empêcher l'effacement du conteneur *racine*, cela revient à considérer l'absence pure et simple de gestion d'identités utilisateur, puisque ce dernier serait définitivement lié à une unique identité dont il n'aurait pas même eu le loisir de choisir le CN. Bien qu'il s'agisse, à proprement parler, d'un modèle d'identité numérique viable, son intérêt est extrêmement limité, et ne répond pas à notre ambition de laisser à l'utilisateur le choix du pseudonymat. Nous ne retiendrons donc pas un tel modèle.

4.2.2 Cas d'une carte à plusieurs identités

L'analyse précédente nous amène à envisager un modèle reposant sur une carte EAP-TLS pouvant accueillir en même temps **plusieurs** identités. Dans un tel paradigme, nous pouvons concilier les différents éléments soulevés précédemment. Notamment, il est possible de disposer d'un conteneur pré-enregistré dans la carte lorsque cette dernière parvient à l'utilisateur, tout en laissant à ce dernier le loisir de charger d'autres identités dans des emplacements dédiés. Ce conteneur *racine*, comme nous l'avons appelé (de même que nous appellerons respectivement par la suite identité *racine* et certificat *racine*, l'identité et le certificat associés à ce conteneur), permet à l'origine de réaliser le lien entre l'émetteur de la carte, l'utilisateur, et le serveur d'identité. Il occupe donc une fonction essentielle du point de vue de la gestion des identités. Néanmoins, il est possible d'objecter qu'un simple numéro de série, unique, et écrit de manière permanente dans la carte, est suffisant pour remplir cette fonction. A cela, les arguments suivants peuvent être opposés :

- La présence d'un conteneur d'identité originel, contenant par exemple le numéro de série dans le CN du certificat X509, permet d'utiliser le TLS embarqué afin d'authentifier, notamment, la carte au serveur. Par ce moyen, ce n'est pas uniquement le numéro de série qui est vérifié, mais également la validité du certificat associé.

- La présence de clés asymétriques dans le conteneur constitue une opportunité pour sécuriser le chargement des identités générées par l'utilisateur.

Le deuxième point, notamment, pousse la logique du conteneur *racine* à assurer également le lien entre l'identité réelle de l'utilisateur et les identités générées. Ce lien, pour exister, suppose l'unicité et le caractère non interchangeable de la carte EAP-TLS utilisateur, et l'assurance que les conteneurs d'identité ne peuvent pas être copiés d'une carte à l'autre. Une telle procédure de sécurisation implique la présence d'une clé résiduelle au sein de la carte, sur laquelle l'utilisateur n'a aucune possibilité d'action. Cette sécurisation, détaillée dans la section 4.3.4, met ainsi précisément à contribution les clés asymétriques du conteneur *racine*.

L'utilité de ce conteneur étant reconnue, au moins un autre emplacement est nécessaire afin de laisser à l'utilisateur la possibilité de générer, et charger, une identité dont il aura choisi la dénomination. Du point de vue du serveur d'identités, le lien entre le conteneur *racine* et les identités générées par l'utilisateur est assez simple à réaliser, pourvu que la génération d'identités soit conditionnée à une préalable authentification par l'identité *racine*. Un tel modèle permet donc de répondre aux ambitions que nous avions énoncées en termes de service, à savoir l'établissement de ce lien, d'une part, et la promotion du pseudonymat, d'autre part.

Enfin, dans un contexte où plusieurs conteneurs d'identités peuvent cohabiter dans un même composant, l'intérêt de la commande *Set-Identity*, décrite au chapitre précédent (section 3.3.4) et jusqu'ici assez artificielle, est à présent entièrement palpable, d'autant que la présence d'un conteneur racine confère une différence de statut aux identités présentes, différence dont il sera nécessaire de tenir compte.

4.3 Le serveur d'identités

Le serveur d'identité est l'élément central du système de gestion que nous avons conceptualisé. Il vise à fournir les fonctionnalités suivantes : authentification de la carte utilisateur via son identité *racine*, création d'un compte lié à cette identité *racine*, permettant alors une libre gestion par l'utilisateur de ses identités (création, modification, suppression), le chargement d'identités dans la carte par l'intermédiaire de conteneurs sécurisés, la mise à jour des informations personnelles associées au compte *racine* d'une part, et à chacune des identités générées, d'autre part.

4.3.1 Création d'un compte *racine*

La section précédente a expliqué l'intérêt de disposer d'un conteneur *racine*. D'après ce que nous en avons dit, ce conteneur *racine*, généré et placé dans la carte à l'étape de la fabrication en usine, se présente sous la forme suivante (l'ECC pouvant fort bien remplacer le RSA, que nous avons déployé dans notre implémentation) :

- Type de l'identité : client
- Nom associé à l'identité : root
- Nom associé à l'EAP-Identity : root
- Clé privée RSA en mode CRT
- Certificat X509 dont le CN est le numéro de série de la carte

Nous détaillons à présent la manière dont ce conteneur est exploité par le serveur d'identités. Avant tout, dans l'optique de conditionner l'accès de l'utilisateur à la gestion de ses identités par la vérification de la validité de sa carte, il est nécessaire de réaliser une authentification TLS mutuelle impliquant la carte utilisateur. De cette manière, le numéro de série peut être vérifié, de même que le certificat X509. Il sera également nécessaire au serveur de récupérer ainsi un élément de nature cryptographique. Le numéro de série étant par nature un élément symétrique public, il est impossible de l'employer à cet effet. En revanche, la clé publique RSA du conteneur *racine* est un excellent candidat.

Cette première authentification mutuelle est réalisée après la soumission par l'utilisateur d'informations personnelles sur son identité réelle dans le but de créer un compte sur ce serveur d'identités. Dans le cadre de la mise en place d'une preuve de concept, aucun moyen de vérification de la véracité de ces informations n'a été mis en place (en dehors de l'adresse e-mail), mais un opérateur d'identités reprenant l'administration d'un tel serveur serait libre de le faire. Néanmoins, l'orientation de notre modèle d'identité encourage à laisser suffisamment d'informations véridiques afin de parer à l'éventualité de la perte de la carte par l'utilisateur, qui sera décrite dans la section 4.4.3. Une fois les informations saisies, l'utilisateur peut accéder à l'interface de gestion des identités proprement dite – le serveur étant alors capable de lier, dans une base de données, l'ensemble des identités TLS de l'utilisateur avec son compte *racine*. Bien évidemment, si l'utilisateur possède déjà un compte sur ce serveur, une simple authentification mutuelle à partir de son identité *racine* sera suffisante pour accéder à cette interface.

4.3.2 Gestion des identités

Une fois l'utilisateur authentifié à partir de son conteneur racine, il peut à loisir générer des identités dont il choisit librement le nom – en dehors de *root*, qui est associé à l'identité *racine*. Ces identités ne sont pas, à ce stade, des conteneurs prêts à être chargés dans la carte, mais de simples comptes supplémentaires gérés par la base de données du serveur d'identités. Ces comptes sont tous affiliés au même compte « maître », qui est celui de l'identité *racine*. Le choix de séparer la création d'identité et le chargement de conteneurs dans la carte s'explique par un nombre d'emplacements potentiellement limité dans la carte, qui ne doit pas entraîner la suppression d'identités moins usitées que d'autres. La figure 18 présente une capture d'écran du serveur d'identités sur lequel l'utilisateur authentifié par sa carte a généré trois identités : Alice, Bob et Eve. Seules les deux premières ont été chargées sous forme de conteneur d'identité TLS dans la carte (cf. section suivante), par conséquent l'utilisateur peut choisir de les sélectionner comme identité courante de leur carte.

Figure 18 : Identités générées sur le serveur dédié

Chaque identité possède un ensemble d'informations personnelles librement éditables par l'utilisateur, hormis le nom ou pseudonyme saisi à la création. L'utilisateur seul peut décider quelles informations sont publiques ou privées. Par ailleurs, une page personnelle, publique, peut être générée pour chaque identité sur la base des informations publiques saisies par l'utilisateur. Ces informations auront également un intérêt réel lors d'une authentification via OpenID auprès d'un SP, car il est fréquent que des informations liées à l'identité authentifiée doivent être transmises de l'IdP au SP.

La suppression des identités créées par l'utilisateur est libre, mais conditionnée à l'absence du conteneur d'identité correspondant dans la carte de l'utilisateur – situation qui conduirait obligatoirement à une incohérence.

4.3.3 Chargement des identités TLS dans la carte

La création d'identités sur le serveur permet à l'utilisateur de charger le conteneur correspondant dans sa carte à puce. Pour cela, une interface lui permet d'afficher le contenu actuel de sa carte, afin de déterminer s'il reste un emplacement libre, comme le montre la capture d'écran représentée figure 19. Si c'est le cas, il peut sélectionner l'emplacement libre de son choix afin d'y charger le conteneur. Sinon, il peut librement effacer tout conteneur existant, sans que cela n'affecte l'existence de cette identité sur le serveur ; en revanche, il ne pourra pas en faire usage pour une authentification TLS mutuelle.

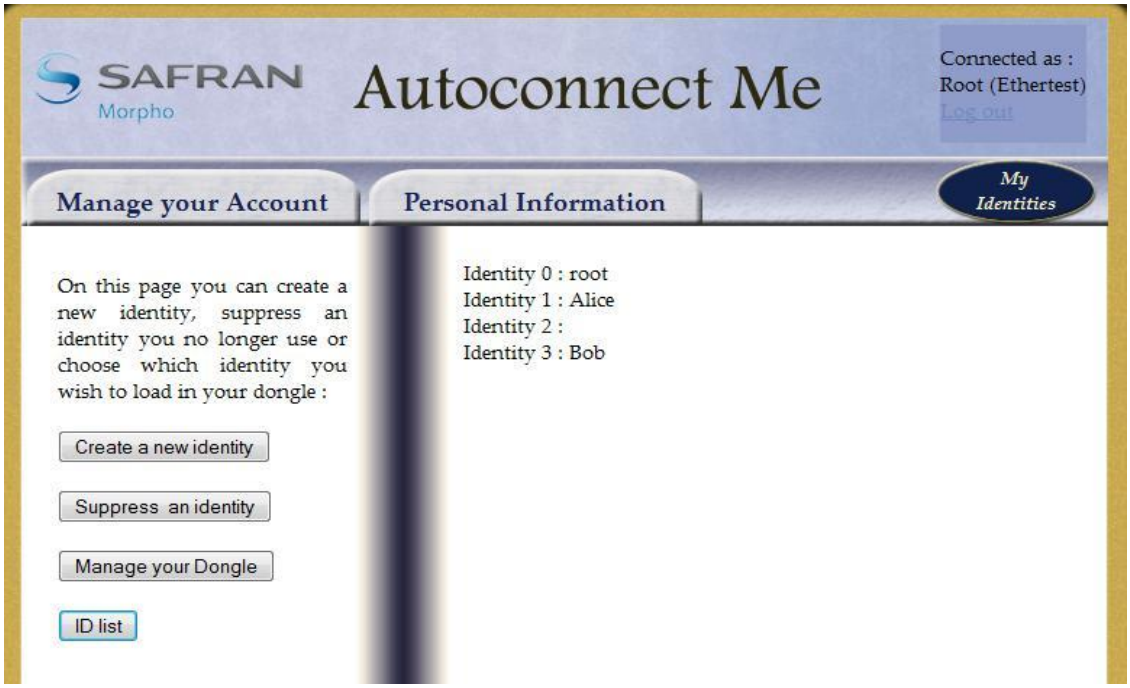

Figure 19 : Liste des identités TLS chargées dans la carte

Dans notre implémentation, jusqu'à 3 conteneurs d'identité peuvent être stockés simultanément, en sus du conteneur *racine*. Dans la Javacard, un espace mémoire de 8 Ko a été alloué en E²PROM à cet effet. Chaque conteneur a une taille d'environ 1,5 Ko. Par conséquent, un offset positionné tous les 2 Ko indique le début des 4 conteneurs éventuellement présents dans la carte. A titre d'illustration, le format du conteneur d'une identité TLS générée au nom « Alice » est le suivant :

- Type de l'identité : client
- Nom associé à l'identité : Alice

- Nom associé à l'EAP-Identity : Alice
- Clé privée RSA en mode CRT
- Certificat X509 dont le CN est Alice

La cohabitation de plusieurs conteneurs d'identité au sein de la carte peut conduire à des ambiguïtés quant à l'identité courante, telle que vue par l'application Javacard. La sélection d'une identité peut être réalisée par l'envoi de l'APDU *Set-Identity* à la carte. Cette APDU peut être générée sur le serveur par un bouton dédié, mais l'ajout d'une ligne de commande à cet effet dans le logiciel proxy permet à l'utilisateur de changer facilement d'identité à tout moment, y compris pour revenir à l'identité *racine*, utile notamment pour accéder au serveur d'identités. L'ensemble de l'architecture de gestion des identités, représentant à la fois la partie associée à la carte et celle associée au serveur d'identités, est représenté figure 20. Trois identités générées par l'utilisateur y apparaissent, chacune ayant été chargée sous forme d'un conteneur dans la carte. Néanmoins, il serait tout à fait possible que l'une des identités générées au niveau du serveur ne figure pas dans la carte – alors que l'inverse est en revanche impossible. La clé privée de l'identité *root* est par ailleurs indispensable au chargement dans la carte des conteneurs d'identité générés par l'utilisateur, comme cela sera expliqué dans la section 4.3.4.

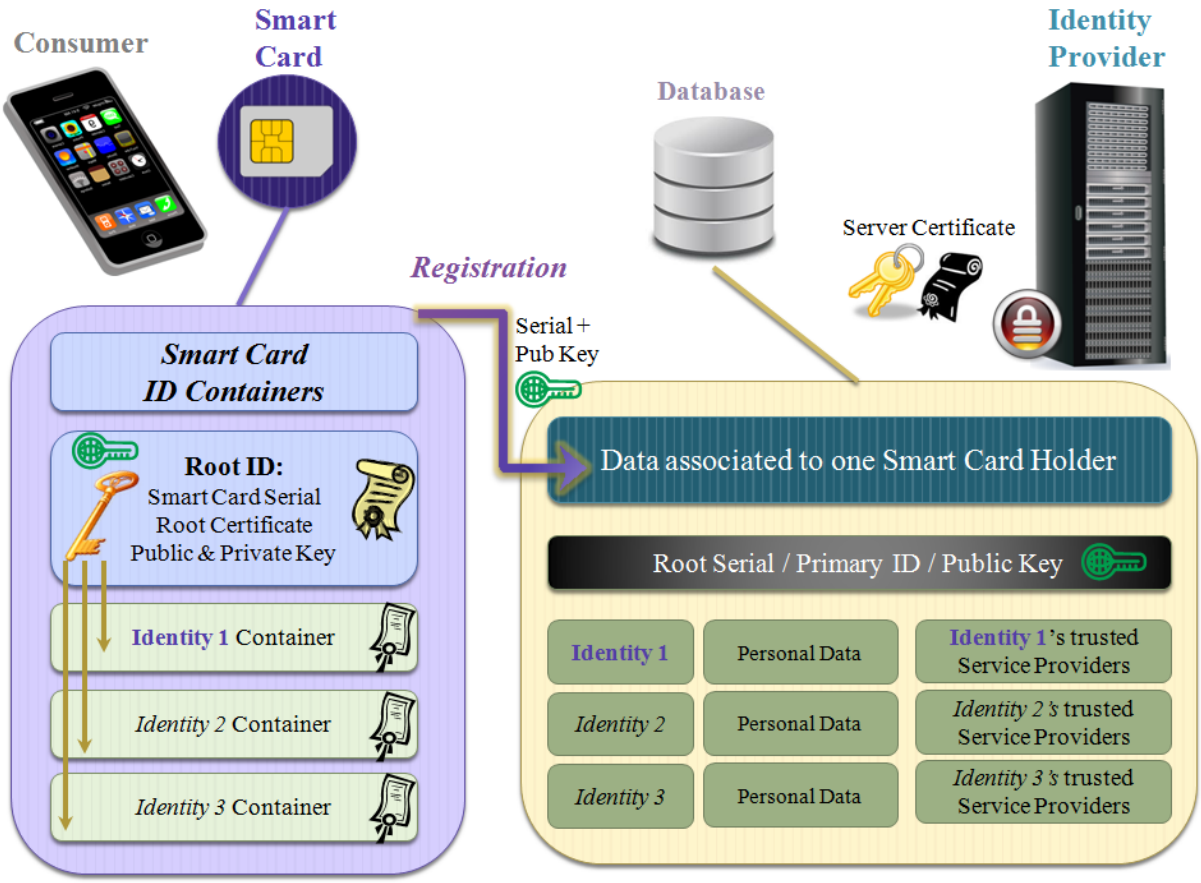

Figure 20 : Gestion des identités sur le serveur et dans la carte

Le certificat présent dans un conteneur d'identité est, dans notre modèle, délivré par le serveur d'identité lui-même, ce qui signifie que le serveur d'identité a une fonction d'autorité de certification. Toutefois, selon le contexte, il peut s'agir d'une autre entité (ou même de plusieurs), à laquelle ce rôle serait délégué. Pour que l'architecture globale fonctionne, il est nécessaire que les certificats délivrés à l'utilisateur soient reconnus dans le contexte où ils sont mis à contribution. Ainsi, dans le cas d'un déploiement de ces certificats pour une authentification par serveur OpenID interposé, seuls le serveur d'identité et le(s) serveur(s) OpenID concerné(s) doivent avoir connaissance du CA. En revanche, si nous nous trouvions face à un déploiement pour une authentification directe auprès de fournisseurs de services, il faudrait que le CA soit reconnu par chacun d'entre eux. Réciproquement, il est nécessaire que l'application Javacard réalisant l'EAP-TLS et stockant les conteneurs d'identités ait une connaissance préalable du ou des CA signant les certificats des serveurs d'authentification, ce qui n'est pas problématique pour le serveur d'identité ou pour le cas de serveurs OpenID, mais qui peut l'être dans le cas où les serveurs Web de fournisseurs de services indépendants possèdent chacun un certificat dont le CA n'est pas nécessairement connu à l'avance.

Néanmoins, il est raisonnable de penser, dans un premier temps, qu'un déploiement par l'intermédiaire d'un serveur OpenID est suffisamment représentatif du potentiel de la technologie. Si un cercle de fournisseurs de services se montrait intéressé par la carte EAP-TLS, la question du CA se règlerait probablement par un accord avec l'entité gérant le serveur d'identités – mais ces considérations sortent du cadre de ce travail de thèse.

Enfin, dans la perspective de ne pas rompre le lien entre les identités générées et éventuellement chargées par un utilisateur, et l'utilisateur lui-même (représenté par sa carte, unique), il est nécessaire, comme il a été souligné auparavant, d'empêcher qu'un utilisateur puisse de lui-même écrire un conteneur dans sa carte, ou bien copier une de ses identités dans une autre carte que la sienne, sans quoi l'ensemble de la logique de gestion mise en place jusqu'ici s'écroulerait.

4.3.4 Sécurisation des conteneurs d'identité

Le procédé de sécurisation des conteneurs d'identité s'appuie sur de la cryptographie, d'une part, et sur une APDU d'écriture dédiée, d'autre part. Il est en effet indispensable, pour les raisons déjà soulevées, de bloquer toute APDU permettant à l'utilisateur d'écrire directement dans la mémoire E²PROM de la carte. En revanche, il est possible de laisser la permission sur une APDU d'écriture conditionnelle, qui fonctionne de la manière suivante : si le conteneur d'identité chiffré reçu par la carte a été validé et correctement déchiffré, alors il peut être écrit en clair dans l'E²PROM à l'emplacement souhaité par l'utilisateur. Sinon, un message d'erreur est renvoyé par la carte et le conteneur est rejeté.

Du point de vue de la cryptographie, nous mettons à profit l'existence dans la carte d'un conteneur *racine* permanent, et donc de clés asymétriques résiduelles, que nous nommerons K_{pub} et K_{priv}. Dans notre implémentation, l'algorithme asymétrique déployé est toujours RSA, mais tout type de

cryptographie asymétrique, notamment l'ECC, fonctionnerait de même. Comme le montre la figure 21, au moment de la génération du conteneur en clair par le serveur d'identités, ce dernier adjoint un en-tête de longueur fixe comportant plusieurs informations relatives au conteneur (nom de l'identité, emplacement souhaité, *timestamp*, etc.), mais également une clé symétrique aléatoire *K*. Cette clé sert à chiffrer le corps du conteneur d'identité par un algorithme standard, par exemple AES[]. Nous pouvons ainsi représenter l'état du conteneur en clair sous la forme simplifiée suivante :

$$C = (K \mid Padding\ Bytes \mid Identity)$$

A la suite du chiffrement symétrique de l'identité par *K*, l'en-tête est chiffré avec K_{pub}, et l'ensemble est signé, par exemple selon la norme PKCS#1, en utilisant la clé privée du CA. Le conteneur d'identité chiffré peut alors être représenté sous la forme suivante :

$$EC = (\{K \mid Padding\ Bytes\}_{Kpub} \mid \{Identity\}_{K} \mid PKCS\#1\ Signature)$$

Il est important de préciser qu'à nouveau, nous supposons ici que le serveur d'identités est également le CA pour les certificats générés. Ce n'est pas nécessairement le cas, et les détails précédents peuvent être adaptés au cas où le CA est une entité tierce. C'est cette entité, dans ce cas, qui devra générer les conteneurs sécurisés, après requête de l'utilisateur au serveur d'identités.

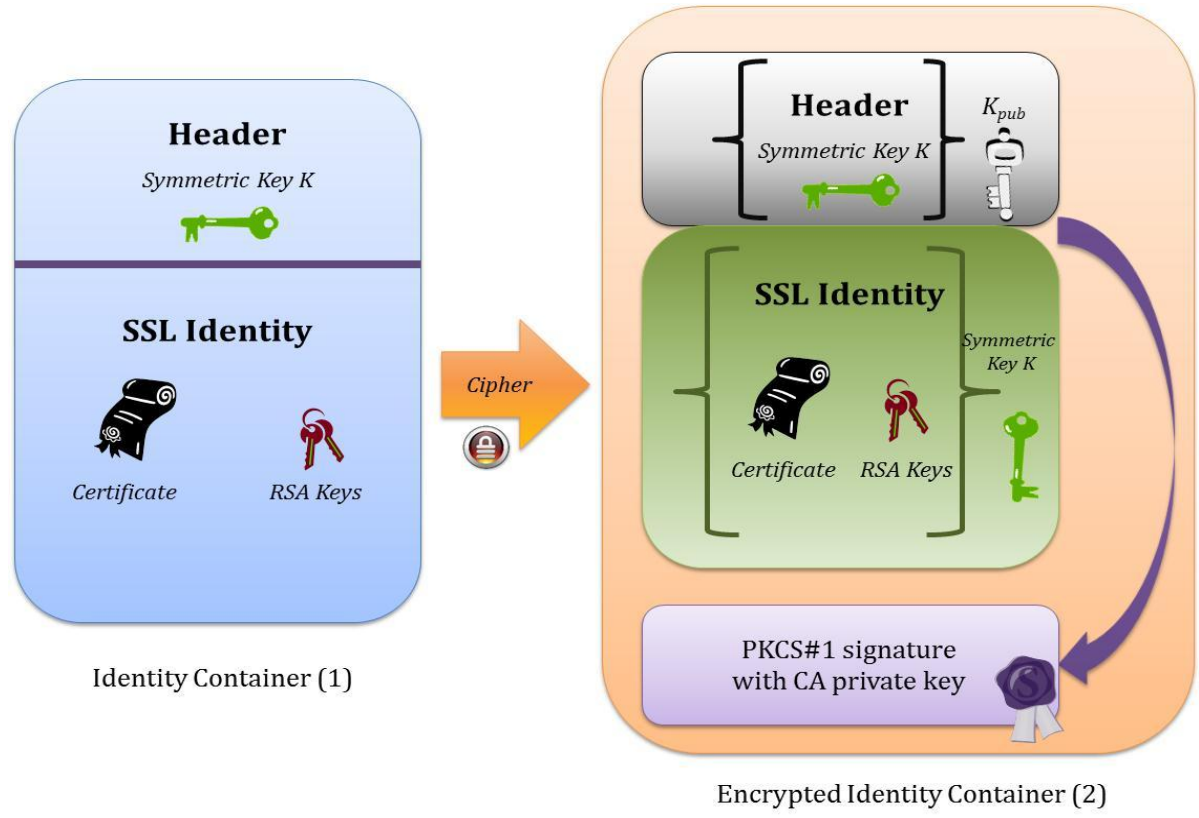

Figure 21 : Format du conteneur d'identité en clair et chiffré

Une fois le conteneur transmis à la carte, celle-ci commence par vérifier la signature du conteneur. Si elle est incorrecte, celui-ci est rejeté directement. Sinon, la carte tente de déchiffrer l'entête à l'aide de K_{priv}, puis enfin l'identité à stocker à l'aide de la clé symétrique K retrouvée dans l'entête. Avant de finalement réaliser l'écriture du conteneur d'identité en E²PROM, le certificat associé à cette identité sera également vérifié.

Ainsi, cette sécurisation assure que seule une carte donnée sera capable de réaliser l'écriture d'un conteneur d'identité valide donné. Le lien entre l'utilisateur, sa carte, et ses identités est donc intégralement préservé.

4.4 Le cycle de vie de la carte d'identités TLS

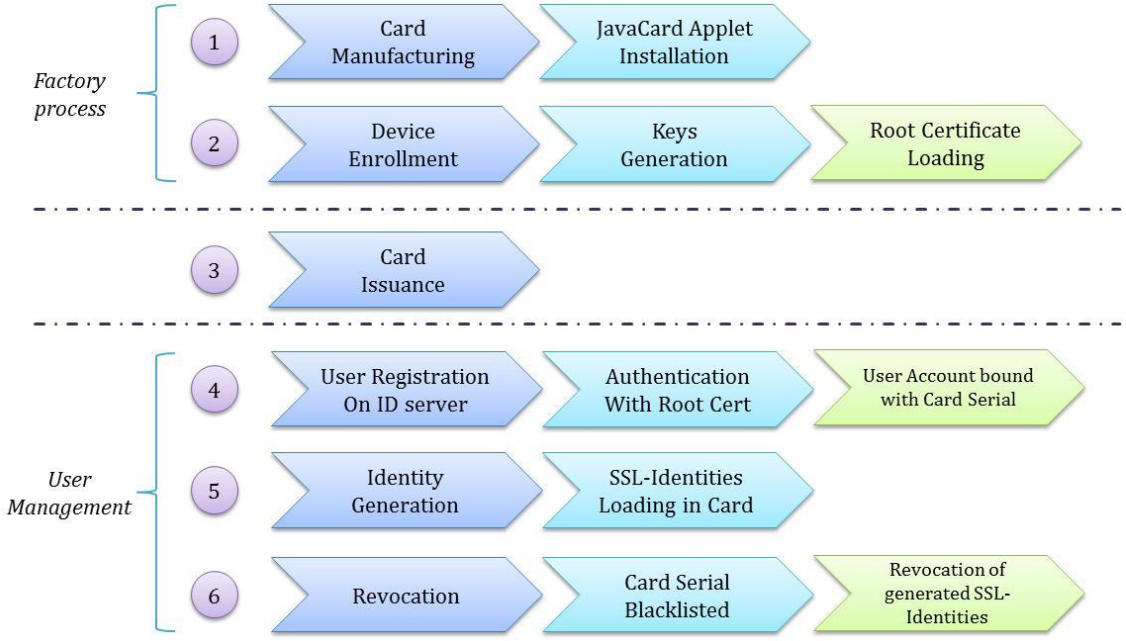

Figure 22 : Cycle de vie de la carte d'identités TLS

L'ensemble des fonctionnalités associées au modèle d'identité envisagé et au serveur associé peut être résumé par le cycle de vie de la carte EAP-TLS de l'utilisateur, représenté sur la figure 22, qui est très similaire à celui d'un composant de type carte bancaire. Ce cycle de vie donne une vision synthétique des différentes étapes qui interviennent depuis la création de la carte, en usine, jusqu'à sa perte ou à sa destruction par l'utilisateur. Il est possible de diviser ce cycle de vie en deux phases : celle de la création et de la configuration préliminaire de la carte – étapes qui ont lieu en usine – et celle de la gestion de la carte par l'utilisateur, à compter de l'instant où il fait l'acquisition de la carte. Nous y ajoutons également une troisième étape, plus spécifique, qui s'attache à répondre à la problématique de la fin de vie de la carte.

4.4.1 Création de la carte en usine

L'étape 1 du cycle de vie, tel que représenté sur la figure 22, consiste en la fabrication du composant sécurisé, lequel peut être typiquement une carte à puce, qui peut à son tour être embarquée dans divers éléments tels qu'un jeton USB, un jeton 3G, ou un Smartphone. Le cycle de l'identité est alors amorcé par la configuration de la carte, étape 2, qui consiste à charger l'application EAP-TLS, à définir les autorités de certification mobilisées pour la création des certificats des identités des utilisateurs, et dont les clés publiques correspondantes seront chargées dans la carte. Enfin, il sera nécessaire de générer une paire de clés asymétriques propre au composant et dont la clé publique sera associée à un certificat X509 qui sera l'identité *racine* du composant. Le sujet (CN) du certificat sera le numéro de série, unique, du composant. Une liste blanche recensant les numéros de série des identités *racines* existantes, accessible depuis le serveur d'identités, permettra à ce dernier de déterminer la validité du composant présenté. Cette configuration préalable ayant été réalisée, l'étape 3 du cycle de vie, à savoir l'émission du composant, peut avoir lieu.

4.4.2 Gestion de la carte par l'utilisateur

Une fois l'utilisateur en possession du composant, il doit se rendre sur le serveur d'identités afin de s'enregistrer auprès de ce dernier, afin d'amorcer l'étape 4 du cycle de vie. L'enregistrement nécessite à l'utilisateur de fournir des données personnelles (nom, prénom, adresse, numéro de téléphone, et un secret qui ne sera utilisé qu'en cas de perte du composant), qui seront de fait mises en relation par le serveur avec l'identité *racine*. La phase d'enregistrement n'est achevée qu'après la mise en place d'une session mutuelle TLS avec le serveur d'identités impliquant la carte de l'utilisateur, et notamment son identité *racine*, dont le certificat X509 et le numéro de série (CN du certificat) doivent être validés par le serveur. Par ailleurs, il n'est pas possible à l'utilisateur de s'enregistrer avec un autre composant que le sien, car la présentation d'un code PIN est indispensable pour lancer le logiciel proxy, et donc pour amorcer une authentification mutuelle avec un serveur distant.

L'utilisateur enregistré peut alors entamer l'étape 5, qui consiste en la génération d'identités à partir du serveur dédié. Comme il a été expliqué dans la section 4.3.2, l'utilisateur génère dans un premier temps des identités dans la base de données du serveur. Il peut alors, s'il le souhaite, en éditer les informations personnelles associées. En fonction de l'espace restant sur sa carte, l'utilisateur peut alors générer un conteneur d'identité associé pour le charger dans son composant. Il pourra alors sélectionner l'une des identités chargées afin de s'en servir pour s'authentifier auprès d'un serveur acceptant l'authentification mutuelle TLS. A tout moment, il peut également libérer un emplacement sur sa carte ou écrire un conteneur par-dessus un conteneur préexistant. La seule contrainte de gestion subsistant auprès de l'utilisateur est celle interdisant la suppression d'une identité de la base de données du serveur si le conteneur correspondant est, à cet instant, chargé dans la carte de l'utilisateur.

4.4.3 Fin de vie de la carte

Le cycle de vie de la carte s'achève lorsque l'utilisateur perd cette dernière, ou le matériel qui l'embarquait, par exemple un Smartphone. Il peut également s'agir de la fin de vie naturelle de la carte, notamment lorsqu'un trop grand nombre d'écritures (de l'ordre de 100 000 cycles) a eu lieu dans une zone mémoire EEPROM mise à contribution par l'application EAP-TLS, typiquement lors des *Handshakes* TLS. L'étape 6 est alors la révocation du composant considéré, qui se déroule de la manière suivante. Lorsque l'utilisateur se rend compte de la perte ou du dysfonctionnement de sa carte, il doit le signaler aux administrateurs du serveur d'identité ou à l'entité qui aura été désignée responsable, selon le cas d'utilisation. Il pourra par exemple s'agir d'un opérateur téléphonique. L'utilisateur doit alors fournir les informations personnelles correspondant à son compte sur le serveur d'identité, de manière à ce que les administrateurs puissent prendre en compte cette information. Ces derniers placeront alors le numéro de série du composant dans une liste noire, interdisant toute utilisation ultérieure du composant avec le serveur d'identités. Par ailleurs, tous les certificats générés pour les identités créées par l'utilisateur sont révoqués, de manière à ce que personne ne puisse utiliser le contenu de la carte auprès de serveurs acceptant l'authentification mutuelle TLS.

Afin d'éviter l'inconfort d'une suppression complète du compte utilisateur associé, celui-ci est conservé sur la base de données du serveur. Un nouveau composant, dont le numéro de série aura été préalablement associé par les administrateurs du serveur d'identité à l'ancien compte de l'utilisateur, est alors envoyé à l'adresse qui avait été saisie lors de la phase d'enregistrement, et que seul l'utilisateur était en mesure de modifier lorsque le composant, fonctionnel, était en sa possession. Il peut alors s'authentifier sur le serveur d'identités avec ce nouveau composant afin de lier ce dernier (et notamment la nouvelle clé publique du composant) avec son ancien compte utilisateur, et ainsi récupérer l'ensemble des identités déjà générées ainsi que les informations personnelles associées. Selon les cas d'utilisation, un secret donné par l'utilisateur lors de l'enregistrement, et/ou un secret émis à l'utilisateur en même temps que son composant initial, pourront servir d'éléments certifiant de manière plus forte l'identité de l'utilisateur déclarant la perte de son composant, de manière à éviter du déni de service de la part d'un attaquant connaissant suffisamment de données personnelles de la victime pour prétendre parler en son nom. Bien que l'attaquant ne puisse pas recevoir le composant à la place de la victime, il lui est ainsi possible de l'empêcher de s'en servir, ce qui, à défaut d'être un réel danger, représente tout de même une nuisance certaine.

4.5 Analyse de la sécurité de l'architecture

Du point de vue de la sécurité, l'architecture que nous proposons comporte plusieurs points à examiner, notamment l'authentification, l'enregistrement, l'accès aux services de gestion d'identité et aux serveurs Web, et la révocation.

4.5.1 Authentification

Avant tout, l'authentification préconisée dans l'ensemble de cette architecture repose exclusivement sur le TLS embarqué, ce qui bannit donc toutes les déclinaisons de mots de passe, avec les inconvénients que ces derniers comportent. L'étude détaillée du TLS embarqué, menée au chapitre précédent, montre que l'authentification résiste aux attaques classiques du type MITM, rejeu, et phishing. Aucune donnée d'authentification sensible ne transite par le logiciel proxy, à l'exception des *Key Blocks*, dont la portée reste limitée. En revanche, il subsiste certaines failles, notamment liées à l'intégrité du proxy (cf. chapitre 3), mais également à la garantie de l'identité réelle de l'utilisateur. Celle-ci est en effet certifiée par la saisie d'un code PIN que seul l'utilisateur est censé connaître. Bien qu'il ne transite pas sur le réseau, ce code PIN est une donnée de type mot de passe, qui reste susceptible d'être connu par un tiers. Néanmoins, un attaquant nécessiterait, pour usurper l'identité d'un utilisateur, à la fois son composant et son code PIN. Or, la perte d'un composant est prévue dans notre cycle de vie et fait l'objet d'un traitement adéquat, ce qui limite largement les possibilités et les conséquences d'une usurpation d'identité.

4.5.2 Enregistrement de l'utilisateur

Au niveau de l'enregistrement de l'utilisateur, les cas d'utilisation réels peuvent imposer la mise en place de procédures plus formelles pour la saisie d'informations. En effet, dans notre modèle, l'utilisateur peut, s'il le souhaite, saisir des données personnelles erronées. Bien que cela ne porte pas directement à conséquence, sinon pour l'utilisateur lui-même qui risque des problèmes lors de la fin de vie de son composant, un cadre administratif strict ne pourra souffrir la saisie d'une identité fantaisiste. Dans ce cas, il reste possible d'envisager la délivrance d'un composant uniquement sur présentation d'une pièce d'identité auprès d'une autorité compétente, ou bien uniquement par des organismes possédant déjà les informations personnelles d'un utilisateur, comme un opérateur téléphonique. Ces données seront alors préalablement associées au numéro de série du composant correspondant, sans que l'utilisateur doive ainsi remplir un quelconque formulaire.

4.5.3 Accès à des ressources protégées

Comme il a déjà été souligné, l'accès aux services de gestion d'identité ne peut avoir lieu qu'après une authentification utilisant l'identité *racine*. Néanmoins, même si l'authentification est sécurisée, l'accès à l'interface de gestion doit faire l'objet de contre-mesures visant à se protéger contre les risques liés à l'inactivité de l'utilisateur. Bien que l'utilisation de sessions limitées dans le temps permette de limiter les risques, cela reste insuffisant. Si, en effet, un utilisateur quitte son poste lorsque sa carte est en cours d'utilisation sur le serveur d'identités, le code PIN ayant déjà été saisi, il est possible à un individu d'effectuer des opérations au nom de l'utilisateur. Le problème se pose également pour l'accès à des serveurs Web acceptant ce mode d'authentification, pour lesquels la

certification de l'identité de l'utilisateur repose sur le code PIN. Afin de limiter ces cas, la contre-mesure la plus efficace consiste à redemander régulièrement son code PIN à l'utilisateur. Il s'agit alors d'un équilibre à trouver entre les nuisances de navigation engendrées par une saisie trop fréquente du PIN, et les risques de nuisances liées l'exécution d'opérations au nom de l'utilisateur. Ces dernières ne peuvent toutefois s'évaluer que dans un contexte précis. Par conséquent, elles n'ont pas fait l'objet d'une étude détaillée dans le cadre de ce travail de thèse.

4.5.4 Révocation

La révocation est certainement l'étape du cycle de vie qui est la plus sensible par rapport à l'ensemble de notre architecture. Elle permet notamment à un attaquant de réaliser du déni de service contre d'autres utilisateurs, si les informations demandées pour entamer la révocation sont insuffisantes. Par ailleurs, la réception d'un nouveau composant repose sur l'adresse de l'utilisateur. Si les failles relevées au paragraphe précédent ont permis à un attaquant de modifier furtivement cette adresse, cela induit un risque d'usurpation d'identité. La demande du code PIN doit donc être obligatoire pour les opérations de modification des données personnelles. Le problème du déni de service peut quant à lui être résolu par l'adjonction de secrets, par exemple des OTP, aux données personnelles, afin de compliquer la tâche de l'attaquant souhaitant désactiver temporairement un compte.

4.6 Implémentation du serveur d'identités

4.6.1 Technologies déployées

Nous avons implémenté le serveur d'identité, baptisé *autoconnect.me*, en langage PHP, et en utilisant comme composant sécurisé une carte à puce de type Javacard, dont les performances ont été données au chapitre 3. L'utilisation de cookies de session classiques permet de maintenir une connexion authentifiée au serveur d'identité, bien que le cookie d'authentification soit associé au domaine 127.0.0.1, du fait de la redirection obligatoire vers le logiciel proxy afin de lancer une authentification mutuelle TLS avant le chargement d'une page protégée. Par conséquent, ce n'est pas seulement l'authentification qui est protégée par le TLS mutuel, mais l'ensemble des pages contenant les fonctionnalités du serveur.

Par ailleurs, l'utilisation des fonctionnalités AJAX[87] est indispensable pour l'échange de données avec la carte à puce. En effet, la technologie AJAX permet d'envoyer en Javascript des requêtes asynchrones vers un serveur afin de mettre à jour dynamiquement une partie de la page courante sans avoir à recharger entièrement celle-ci où à subir une redirection vers une autre page. La limite de cette action réside dans l'application de la politique de même origine, pilier de la sécurité

des applications Web, et qui stipule qu'une requête dont l'utilisateur final n'est pas consciemment l'auteur ne peut pas avoir pour cible un serveur différent de celui dont est parti la requête.

Or, pour des opérations telles que le chargement d'un conteneur d'identité dans la carte, les données sont envoyées par le navigateur grâce à une commande du type : http://127.0.0.1:8080/~apdu=APDU1&APDU2&...&APDUn, qui permet d'envoyer simultanément plusieurs APDUs, grâce au traitement de cette commande par le logiciel proxy. Du point de vue d'AJAX, et bien qu'il s'agisse ici d'un léger artifice, la politique de même origine est respectée, puisque l'utilisateur est à ce moment sur une page sécurisée, i.e. dont l'URL est du type http://127.0.0.1:8080/~url=autoconnect.me/page.html, et la requête AJAX, qui vise à atteindre la carte à puce et non le serveur d'identités, a pour cible le domaine 127.0.0.1:8080, qui n'est autre que le logiciel proxy, lequel redirigera la requête vers la carte. Après traitement de ces APDUs par la carte, l'ensemble des réponses est renvoyé au proxy qui les représente dans un format XML. Si les données étaient retransmises au navigateur sans davantage de traitement, cela conduirait à l'affichage d'une page XML, peu propice à satisfaire l'utilisateur final. Par conséquent, l'utilisation de fonctionnalités AJAX permet la mise à jour d'une partie de la page courante après analyse en Javascript des réponses de la carte. Par exemple, à la suite de l'envoi d'un conteneur, toutes les réponses (Status Words) renvoyées par la carte doivent être 9000. Dans ce cas, un message de succès du chargement est affiché sur la page courante, sinon un message d'erreur apparaîtra.

La génération des conteneurs d'identités est effectuée par du CGI, faisant intervenir plusieurs exécutables codés en C ainsi que le logiciel OpenSSL. Afin d'éviter une surcharge du serveur, l'ensemble des fichiers ayant trait aux identités générées sont aussitôt supprimés après chargement dans le composant de l'utilisateur. De plus, une attention particulière a été portée quant à la gestion de la concurrence, qui pourrait causer des interférences au cours de l'exécution du CGI. Les tests réalisés montrent qu'une génération simultanée de plusieurs identités ne rencontre aucun problème particulier.

4.6.2 Performances mesurées

Les performances observées lors de la navigation sur les pages protégées du serveur, qui nécessitent une authentification mutuelle TLS préalable, correspondent à celles qui ont été mesurées sur la carte à puce correspondante. Autrement dit, la première session TLS, réalisée lors de l'authentification par l'identité *racine*, sera probablement en mode full, avec un surcoût d'environ 2,0 secondes. En revanche, l'accès aux autres pages induit uniquement des sessions TLS en mode *resume*, avec un surcoût négligeable de 0,5 seconde qui ne perturbe aucunement la navigation.

La génération d'un conteneur d'identité peut être décomposée, du point de vue de l'utilisateur, selon l'équation suivante, où T_{CONT} représente le temps de génération d'un conteneur en clair, T_{ENC}

représente le temps de chiffrement d'un conteneur, et T_{TLS} représente le temps nécessaire à la réalisation d'une session TLS avec le serveur d'identités, qui utilise l'identité *racine* :

$$T_{generation} = T_{CONT} + T_{ENC} + T_{TLS}$$

Les mesures effectuées ont permis d'obtenir le détail des différents termes indépendamment les uns des autres, via le temps d'exécution des différents exécutables CGI. Nous nous sommes placés dans le cas défavorable d'une session TLS en mode *full*, ce qui a donné les résultats présentés dans le tableau 4.

T_{CONT}	T_{ENC}	T_{SSL}	$T_{generation}$
0,5s	0,3s	2,5s	3,2s

Tableau 4 : Mesures temporelles pour la génération d'un conteneur d'identité TLS

Ainsi, les performances du serveur relatives à la création d'un conteneur d'identité chiffré sont tout à fait acceptables. Toutefois, il faut y ajouter le temps nécessaire au chargement du conteneur dans la carte de l'utilisateur, qui suit immédiatement la création du conteneur. Ce chargement est réalisé par une interface AJAX qui permet d'envoyer une douzaine d'APDUs à la carte. La moitié d'entre elles sont des commandes d'écriture du conteneur chiffré, et l'autre moitié est constituée de commandes de gestion. Le temps de chargement $T_{download}$ mesuré, d'environ 1,5s, peut être décomposé selon l'équation suivante, où *n* représente le nombre d'APDUs de type WRITE, T_{MGMT} représente le temps lié au traitement des APDUs de gestion, et T_{DEC} représente le temps nécessaire à la vérification de la validité du conteneur chiffré et au déchiffrement de ce dernier, avant son stockage dans la mémoire non volatile de la carte :

$$T_{download} = n.T_{WRITE} + T_{MGMT} + T_{DEC}$$

En moyenne, le cas $n = 6$ est le plus fréquent. Nous obtenons alors, en nous plaçant dans ce cas, les résultats présentés dans le tableau 5.

T_{WRITE}	T_{MGMT}	T_{DEC}	$T_{download}$
75ms	50ms	1050ms	1550ms

Tableau 5 : Mesures temporelles pour le chargement d'une identité TLS dans un composant

Le surcoût dû aux opérations réalisées du côté client est donc très faible, ce qui montre l'aspect réaliste de ce modèle de gestion de l'identité numérique. Ainsi, et bien que le cadre de ce travail de thèse, guidé par les contraintes de l'activité industrielle, n'ait pas permis de le réaliser, un passage à l'échelle est tout à fait envisageable, car le serveur limite son activité à la réalisation de sessions TLS et, parfois, à la génération de conteneurs d'identités chiffrés. Or, cette opération gère

correctement les problèmes de concurrence et reste très peu coûteuse en temps de calcul, comme l'ont montré les mesures effectuées.

4.7 Conclusion

Au cours de ce chapitre, nous avons énoncé et détaillé les principes d'un nouveau modèle d'identité numérique, et nous avons proposé une architecture permettant d'en réaliser la gestion. Le caractère novateur de ce modèle repose spécifiquement sur sa capacité à exploiter efficacement les avantages sécuritaires de plusieurs éléments à l'origine indépendants (notamment, le protocole TLS, l'authentification mutuelle, le composant sécurisé), tout en possédant un format adapté, par nature, à la convergence.

Les multiples incarnations des composants sécurisés offrent autant de possibilités de déploiement de ce modèle, qu'il s'agisse d'une réplique de la carte bancaire, ou bien d'une carte (U)SIM embarquée dans un Smartphone. Le serveur d'identités associé peut alors être supervisé par une autorité administrative, mais également par un opérateur téléphonique, qui possède un lien préétabli avec l'utilisateur via sa carte (U)SIM et les données personnelles relatives à son abonnement. Dans ce contexte, il est possible, par exemple, d'adjoindre au serveur Web une mise à jour des identités par l'intermédiaire d'un serveur OTA, qui pourrait avoir vocation à remplacer l'interface Web de génération et de chargement des identités – AJAX n'étant pas supporté par tous les appareils mobiles.

Malgré ces atouts, le déploiement d'un tel modèle d'identité numérique est conditionné, au moins en partie, par son aisance à s'insérer dans le paysage du Web. C'est la fédération d'identité qui nous permet, le plus simplement possible, de parer à cet écueil. De manière similaire, l'orientation du serveur d'identités vers les opérateurs mobiles n'a de sens qu'avec le déploiement d'un logiciel proxy pour chaque type de Smartphone existant à l'heure actuelle – dans la limite des modèles pouvant accueillir sans difficulté l'installation d'une application externe. Nous détaillons ces aspects dans le chapitre suivant, afin d'aboutir à la description des plateformes complètes fournissant une preuve de concept viable de notre architecture et de notre modèles d'identité numérique.

Chapitre 5 : Intégration de la fédération d'identités et des réseaux mobiles

5.1 Introduction

Le contexte actuel des technologies de l'information confère un cadre naturel dans lequel vient s'insérer l'identité numérique, et visant à promouvoir l'accès à des ressources du Web depuis tout type de plateforme, fixe ou mobile. Le modèle d'identité que nous avons présenté dans le chapitre précédent reste à ce titre très théorique. Bien qu'il assure, de ce point de vue, de nombreuses garanties en termes de sécurité, son développement doit toutefois être approfondi afin d'offrir une réponse valable et solide aux enjeux propres à ce contexte.

Avant tout, il est nécessaire de résoudre le problème du déploiement de notre modèle d'identité auprès des ressources du Web. Comme nous l'avons déjà souligné, la réalisation d'une authentification mutuelle TLS ne peut avoir lieu sans configuration spécifique du serveur, aussi bien d'un point de vue purement protocolaire, que du point de vue de la gestion des autorités de certification. Il n'est donc pas raisonnable d'imaginer un déploiement reposant sur l'adoption du modèle par chacun des acteurs du Web, peu propices à bousculer un modèle d'identité critiquable mais confortable. Dans le chapitre 2, nous évoquions les architectures de fédération d'identité en montrant qu'elles nous permettaient de sortir, en partie, de cette impasse. En effet, de nombreux fournisseurs de service acceptent de déléguer l'authentification à des fournisseurs d'identité, suivant le principe du SSO ou de la fédération d'identité. En mettant à profit ce schéma, nous avons alors implémenté notre propre fournisseur d'identité afin de construire une plateforme de démonstration directement fonctionnelle dans le paysage du Web.

Parallèlement, l'accès aux ressources du Web depuis tout type de plateforme impose des contraintes en termes de développement et d'équipement. Le problème de l'équipement a été discuté et résolu précédemment, en tirant parti du format particulier de notre modèle d'identité, capable de s'insérer à la fois sur des postes fixes (via un CAD dédié ou un jeton USB approprié) que mobiles (la carte (U)SIM équipant une grande partie des Smartphones existants). L'aspect développement reste toutefois encore en suspens. En effet, si le serveur d'identités est pour sa part aisé à orienter aussi bien pour des ordinateurs que pour des appareils mobiles, il reste indispensable d'assurer le bon fonctionnement du logiciel proxy sur chacune de ces plateformes. Celui-ci étant originellement développé en langage C, l'adaptation à des Smartphones du type Android ou Blackberry était indispensable afin de s'inscrire intégralement dans la mouvance actuelle de la mobilité.

Dans ce chapitre, nous décrirons les principales étapes visant à l'élaboration d'une démonstration opérationnelle intégrant la fédération d'identité et la mobilité à notre modèle d'identité numérique.

5.2 Intégration de la technologie OpenID

5.2.1 Choix et implémentation de OpenID

Suite à l'analyse réalisée au chapitre 2, des architectures et technologies proposant la fédération d'identités ou le SSO, le choix le plus judicieux pour la mise en œuvre d'une preuve de concept nous a semblé être OpenID. L'absence de cercle de confiance représente en effet, dans notre cas, un avantage notoire, tandis que de réelles perspectives industrielles nous amèneraient sans aucun doute à prendre le parti pris inverse. Nous avions souligné que la faiblesse principale de la technologie OpenID résidait dans ce qui constitue sa force même, à savoir sa simplicité et sa légèreté de mise en œuvre, qui induit de fait de sérieuses questions de sécurité à résoudre. Ces questions se ramènent en grande partie au niveau de sécurité offert par l'authentification mise en place par l'IdP, et pour laquelle notre modèle d'identité fournit d'excellents atouts.

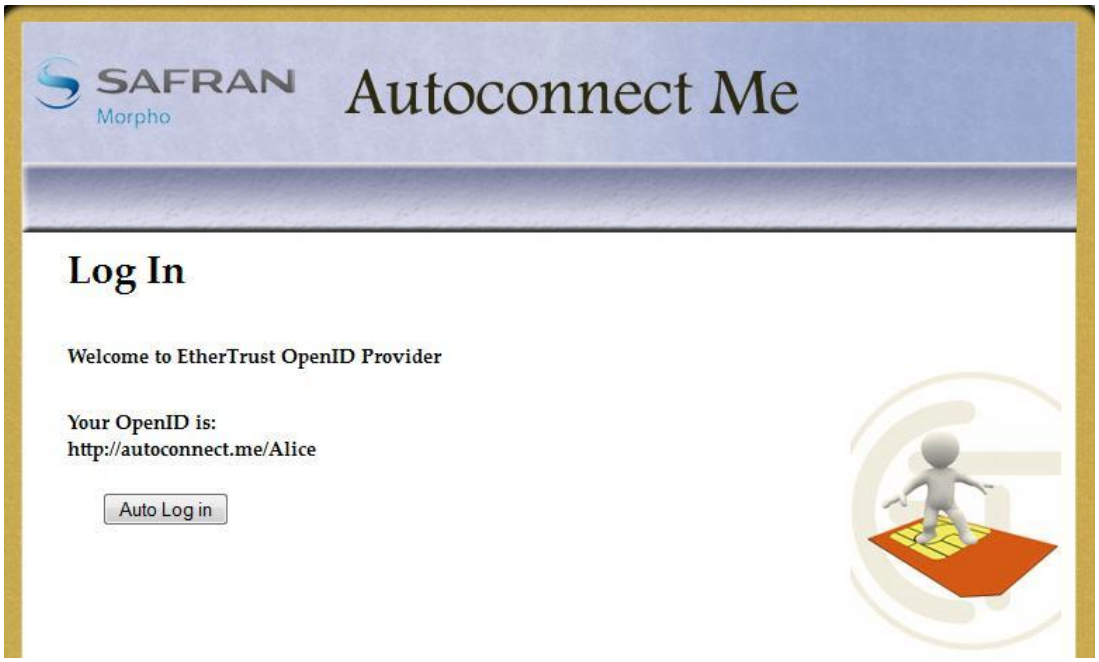

Figure 23 : Interface de Login sur notre IdP OpenID

De ce point de vue, la plateforme que nous avons montée peut être vue comme un procédé de sécurisation de l'architecture OpenID, ayant à cet effet fait l'objet d'un dépôt de brevet. L'implémentation a été réalisée à partir de la librairie Janrain pour PHP. La principale modification apportée à cette source concerne la méthode d'authentification proposée. Par défaut, en effet, il s'agit d'une simple demande de couple identifiant / mot de passe. Nous l'avons transformée en un bouton

réalisant une authentification mutuelle TLS utilisant obligatoirement la carte de l'utilisateur, comme le montre la capture d'écran représentée figure 23. Par ailleurs, bien que l'IdP puisse théoriquement être entièrement séparé du serveur d'identités, nous avons fait le choix de le rattacher au même domaine, afin d'accentuer le lien existant entre ces deux entités.

Dans ce même esprit, un onglet dédié à OpenID a été ajouté sur la page de chaque identité générée par l'utilisateur sur le serveur d'identités, lui indiquant notamment l'URL lui servant d'identifiant OpenID, du type : *http://autoconnect.me/identity*.

5.2.2 Scénario d'authentification

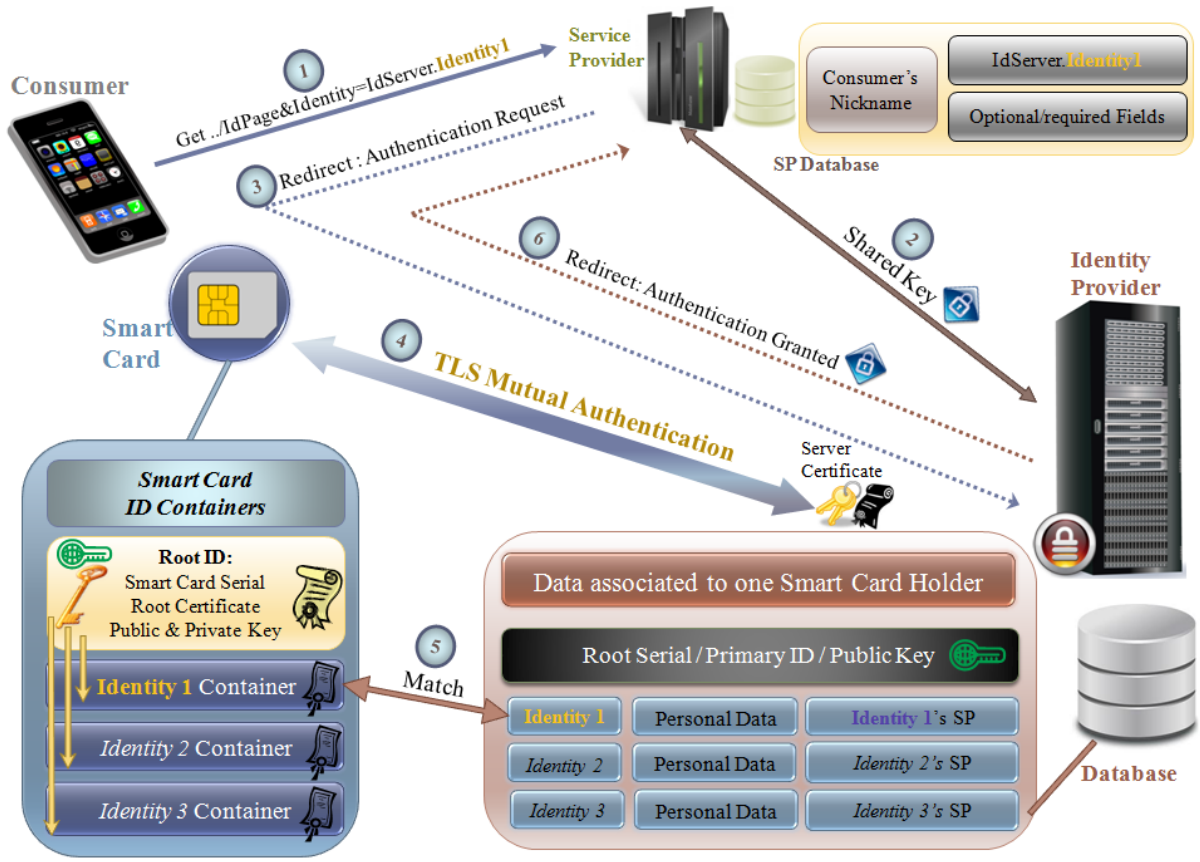

Figure 24 : Plateforme de démonstration

La plateforme de démonstration ainsi constituée peut être visualisée selon la figure 24. Pour des raisons de simplicité, le serveur d'identités et l'IdP OpenID ont ici été rassemblés en une seule entité. L'enregistrement de l'utilisateur auprès du serveur d'identités à l'aide de sa carte étant considéré acquis, la génération et le chargement de conteneurs d'identité dans la carte devient possible. Pour chaque identité générée et chargée, l'utilisateur peut à tout moment se servir de son identifiant OpenID pour s'authentifier auprès d'un SP. Bien entendu, si le container d'identité n'est pas présent dans la carte de l'utilisateur, l'authentification sera impossible. Conformément au standard OpenID, Le SP redirige alors l'utilisateur vers l'IdP du domaine *autoconnect.me* où l'utilisateur peut

s'authentifier à l'aide de sa carte. La seule contrainte qui existe pour l'utilisateur, à cet instant, est d'avoir positionné l'identité courante vue par l'application EAP-TLS embarquée sur l'identité de son identifiant OpenID, sans quoi le certificat client envoyé lors de l'authentification ne correspondra pas aux attentes de l'IdP. Ce positionnement peut être très simplement effectué, soit par l'interface Web du serveur d'identités, soit directement par le logiciel proxy. Une fois l'authentification terminée, l'IdP communique au SP le succès ou l'échec de cette dernière par l'intermédiaire d'une redirection HTTP dont l'en-tête contient les données précédentes d'authentification, chiffrées par une procédure HMAC à l'aide du secret partagé entre IdP et SP (cf. chapitre 2 pour plus de détails sur le fonctionnement de OpenID). Cette phase est correctement traitée par OpenID et n'est pas sujette à des failles importantes de sécurité.

Nous avons pu tester avec succès cette plateforme sur de nombreux SP compatibles OpenID, notamment *www.plaxo.com* qui occupe une place importante dans le paysage du Web, et qui à ce titre constitue une démonstration probante.

5.2.3 Limitations

Malgré le caractère fonctionnel de notre preuve de concept, les possibilités de déploiement de notre modèle d'identité numérique offertes par une telle plateforme restent limitées. Avant tout, les cibles potentielles restent les services s'inscrivant dans la mouvance de la fédération d'identités, ou du SSO. Mais il est difficile de prétendre établir un standard que suivront les serveurs gérant eux-mêmes les comptes clients et leur authentification, bien souvent ancrés dans des carcans dont ils ne souhaiteront pas bouger. Au-delà de cette simple constatation, et en nous limitant au cadre des SP compatibles avec la technologie OpenID, deux remarques peuvent venir limiter la portée de notre travail :

- Bien qu'il soit théoriquement possible d'atteindre tous les serveurs autorisant une authentification de type OpenID, les acteurs majeurs tels que Google, Yahoo, ou Microsoft imposent la redirection vers un IdP qui leur appartient. En l'état actuel des choses, il n'est donc pas possible de dire que notre plateforme fonctionne avec ces acteurs du Web. Seul un partenariat serait susceptible de changer cet état de fait.

- Aussi longtemps que notre modèle d'identité ne connaît pas de déploiement à grande échelle, il n'est pas possible pour un SP d'imposer à l'IdP notre procédure d'authentification. Autrement dit, un SP acceptant de déléguer l'authentification via OpenID continuera d'accepter de la part de l'IdP aussi bien des authentifications faibles de type identifiant / mot de passe que des authentifications fortes telles que celle que nous proposons. Par conséquent, en l'absence de déploiement massif de notre solution, notre procédé de sécurisation de l'architecture OpenID n'a pas d'impact majeur sur la sécurité de l'authentification par OpenID vue par un SP. En revanche, si ce déploiement massif avait lieu, la valeur ajoutée deviendrait nettement plus conséquente.

5.3 Adaptation aux terminaux mobiles

Notre plateforme de démonstration a pour vocation d'être accessible depuis le plus grand nombre de terminaux, aussi bien fixes que mobiles. Le cas des plateformes fixes est déjà traité, le logiciel proxy étant écrit en langage C et communiquant avec le CAD via l'API PS/SC. Bien qu'il ait été développé pour le système d'exploitation Windows, une adaptation pour d'autres systèmes n'a pas de raison de poser problème.

En revanche, les terminaux mobiles présentent des particularités dont il est nécessaire de tenir compte. Nous avions déjà évoqué, dans la conclusion du chapitre 4, les limitations des navigateurs embarqués en termes de support des librairies AJAX, sur lesquelles repose largement notre serveur d'identités. Mais il existe également d'autres contraintes, directement liées à l'implémentation du logiciel proxy. Notamment, eu égard à la nature du proxy, il est indispensable d'intégrer ce dernier non comme une simple application, mais comme un *daemon*. Nous nous attardons ainsi sur le cas d'Androïd, qui a fait l'objet, avec RIM, d'une adaptation complète du logiciel proxy. Nous présentons également, à la suite de cela, une architecture alternative à celle que nous avons décrite impliquant un serveur OTA, afin de remédier aux problèmes d'accès aux fonctionnalités du serveur d'identités.

5.3.1 Le système d'exploitation Android

Le système d'exploitation Android[88][104] a été conçu, originellement, par l'entreprise *Android Inc.*, rachetée par Google en 2005. Le premier terminal mobile intégrant ce système d'exploitation (*HTC Dream*) a été commercialisé en 2008. De nombreuses versions ont été publiées depuis, l'un des apports notables ayant consisté à rapidement développer le support de la technologie NFC, lequel a été présent dès la version 2.3, baptisée *Gingerbread*. Le chapitre 8 de ce rapport détaille par ailleurs un cas d'utilisation concret qui en exploite les possibilités ainsi offertes.

Android inclut un noyau Linux, qui fournit une machine virtuelle Dalvik (DVM, *Dalvik Virtual Machine*). Les applications sont écrites avec un environnement JDK classique. Le *bytecode* produit, i.e. les fichiers .class, est alors converti au format Dalvik (.dex), qui emploie un *bytecode* spécifique. Les applications peuvent être agrémentées de plusieurs types de composants, notamment les *Activities* et les *Services*.

- Une *Activity* permet à une application de gérer un écran pour l'interaction avec l'utilisateur. Le système d'exploitation crée une nouvelle instance de la machine DVM au moment de l'activation d'une telle application. La machine à états représentant le cycle de vie d'une *Activity* est illustrée figure 25. L'état d'une *activity* est directement dépendant de son affichage à l'écran. Si celle-ci se trouve à l'avant-plan, elle est *active*. Si elle est visible, mais à l'arrière-plan, elle est en *pause*. Enfin, si elle n'est plus visible du tout, elle est *stoppée*. Dans les deux

derniers cas, l'*activity* continue de tourner, mais si le système a besoin de mémoire, il peut décider de la tuer.

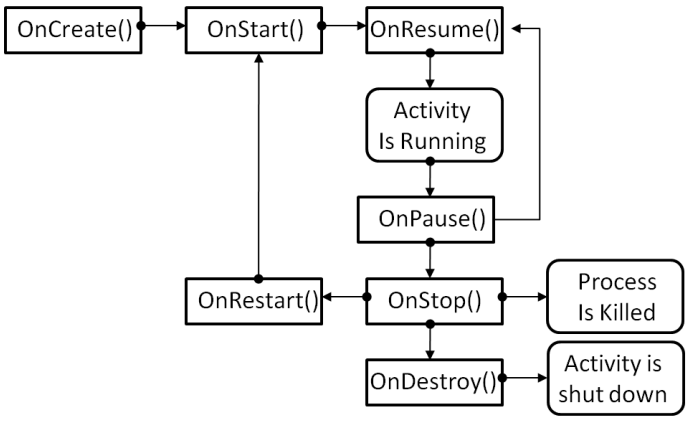

Figure 25 : La machine à états d'une Activity Android

- Un *Service* permet à une application d'exécuter une tâche de fond sans interaction avec l'utilisateur. Il permet également l'exposition de fonctionnalités d'une application à d'autres applications, qui peuvent alors se connecter (*binding*) au *service*. Le cycle de vie d'un *service* est plus simple que celui d'une *activity*, les méthodes de *callback* étant moins nombreuses. La figure 26 illustre le cycle de vie des deux cas d'utilisation d'un *service*.

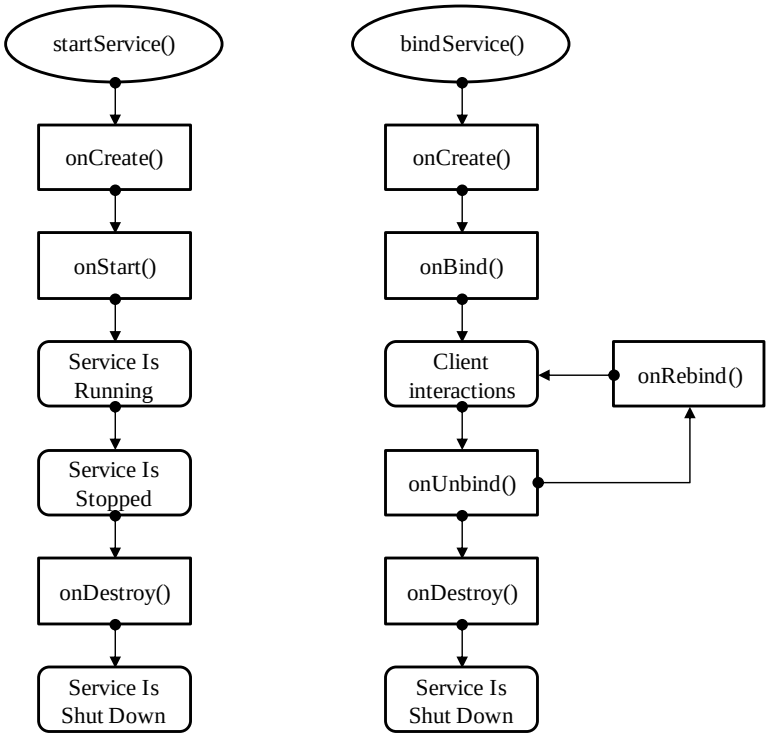

Figure 26 : Machines à états d'un Service Android

La structure d'une application Android est décrite dans le fichier *AndroidManifest.xml*, dans lequel se trouvent toutes les déclarations des composants dont l'application fait usage. Ce fichier inclut également des permissions relatives à l'accès Internet, à l'utilisation de la technologie NFC, ou encore la version minimale du système d'exploitation supportée par l'application. Enfin, chacun des deux composants que nous avons présentés peut être démarré par l'intermédiaire d'un message asynchrone nommé *Intent*, qui est une description abstraite d'une action à réaliser. La liste des *Intents* traités par l'application (*Intent Filter*) se trouve également dans le fichier *AndroidManifest.xml*.

Dans notre cas, l'application étant un logiciel proxy, la forme la plus cohérente à lui donner est celle d'une *activity*, que l'utilisateur doit lancer manuellement, laquelle démarre aussitôt un *service* qui permet l'exécution du logiciel proxy en tâche de fond, et permet donc à l'utilisateur d'utiliser son navigateur pour se rendre et s'authentifier, par exemple, sur le serveur d'identités.

5.3.2 Cas des téléphones RIM Blackberry

Le cas d'Androïd ayant été explicité en détail, nous évoquerons plus succinctement le cas du système d'exploitation produit par l'entreprise canadienne RIM (*Research in Motion*), notamment avec le modèle de Smartphone Blackberry. Le système d'exploitation est constitué d'un environnement Java Micro Edition compatible avec MIDP (Mobile Information Device Profile). Par ailleurs, la plupart des téléphones RIM supportent l'API *Security and Trust Services*, également nommée JSR 177, qui permet de communiquer avec le composant SIM embarqué localement. Par conséquent, l'environnement fourni par ce système d'exploitation est entièrement compatible avec notre application.

Nous nous attardons ici sur une particularité rencontrée lors de l'implémentation. Il s'agit de l'impossibilité d'utiliser une adresse IP fixe locale telle que 127.0.0.1. Cela pose problème relativement à l'implémentation du serveur d'identité, dont les redirections et les liens vers les pages sécurisées étaient prévus pour une configuration standard de ce type, avec des URL suivant le format : *http://127.0.0.1:8080/~url=autoconnect.me/page.php*. L'adresse locale est en effet, dans ce cas, attribuée dynamiquement. Par conséquent, il n'est pas possible, du point de vue du serveur, de prévoir une adresse IP particulière à déployer pour traiter le cas des téléphones RIM.

Le problème soulevé est intéressant, et permet de traiter, plus largement, le problème du numéro de port sur lequel tourne le logiciel proxy. Bien que nos plateformes de démonstration utilisent exclusivement le port 8080, il est en effet nécessaire de prévoir des cas où le numéro de port serait attribué dynamiquement, afin de mettre à jour les liens du serveur d'identité en conséquence. Pour traiter cela, la solution que nous avons mise en place consiste à envoyer, au lancement de l'application, une requête HTTP au serveur d'identités indiquant l'IP locale du téléphone et le numéro de port associé à l'application, de manière à intégrer ces données à la variable PHP $_SESSION. Les liens vers les pages protégées peuvent alors être mis à jour dynamiquement en fonction de chaque utilisateur. De plus, les Smartphones que nous avons utilisés envoient tous, en en-tête HTTP, des

informations relatives à leur identité, ce qui permet au serveur d'analyser facilement le type de plateforme de chaque utilisateur, et de s'adapter en conséquence.

5.3.3 Architecture avec serveur OTA

Nous avons justifié, dans le chapitre précédent, l'intérêt de la technologie AJAX dans la communication entre le serveur d'identité et la carte utilisateur. Néanmoins, comme nous l'avons souligné, elle n'est pas toujours supportée correctement par les navigateurs des terminaux mobiles. Cet état de fait nous a conduits à envisager une architecture alternative intégrant un serveur OTA afin de réaliser la mise à jour de la carte et des identités associées par l'intermédiaire de SMS envoyés par ce serveur. De plus, l'ajout de ce serveur répond à l'orientation de développement vers la téléphonie mobile, les opérateurs préférant souvent ce mode d'administration des cartes (U)SIM à celui d'une interface Web – préférence marquée à plusieurs reprises par des demandes extérieures nous étant parvenues.

Bien que cette architecture n'ait pas encore été implémentée à ce jour, plusieurs scénarios d'intégration du serveur OTA ont été envisagés. Le plus réaliste d'entre eux consiste à simplement déléguer la mise à jour des identités dans le composant de l'utilisateur au serveur OTA, et non à l'interface Web. Dans cette perspective, l'interface Web permet à l'utilisateur de rester maître de l'ensemble des données relatives au serveur d'identités, i.e. création et suppression d'identités et informations personnelles afférentes. En revanche, la maîtrise des conteneurs présents dans la carte à puce passe à l'administrateur du serveur OTA, par exemple un opérateur téléphonique, qui prendrait en compte les requêtes de création et de suppression de conteneurs dans le composant de l'utilisateur avant d'accéder éventuellement à ces requêtes par l'intermédiaire d'envois de SMS permettant de mettre à jour le composant.

Autrement dit, relativement à l'architecture présentée dans le chapitre 4, le serveur OTA et l'administrateur afférent constituent un intermédiaire supplémentaire dans la gestion des conteneurs d'identité, mais n'en modifient pas le principe. Bien qu'une telle architecture n'apporte concrètement aucune nouveauté, il nous semble nécessaire de souligner sa compatibilité naturelle avec la solution que nous proposons – la différence entre les deux se présentant alors simplement en termes de choix d'administration.

5.4 Conclusion

Les développements présentés dans ce chapitre permettent de mettre en valeur le caractère tangible et réaliste de notre solution. L'adjonction d'une interface OpenID permet en effet de valider l'ensemble de notre architecture en situation réelle, et non seulement à partir de tests ou de simulation réalisés localement. L'adaptation aux réseaux mobiles ouvre, quant à elle, des perspectives de déploiement favorables auprès des opérateurs téléphoniques.

Le modèle d'identité numérique que nous avons conçu, la carte à identités TLS, acquiert ainsi de nombreuses possibilités de déclinaisons dépendant des cas d'utilisation imaginables. Envisager ces différents contextes permet en effet de concevoir différents angles de vue relatifs à l'application de notre modèle, qui reste jusqu'ici majoritairement théorique. Bien que notre plateforme de démonstration permette de fournir une preuve de concept sur l'idée la plus fondamentale qu'est l'authentification sécurisée auprès de fournisseurs de services du Web, elle ne met pas en valeur des concepts plus appliqués.

La troisième partie de ce rapport propose de se concentrer pleinement sur cet aspect, en proposant trois contextes très différents dans lesquels l'identité TLS est mise à contribution. Bien que ces cas d'utilisation n'emploient généralement qu'une seule identité TLS, ils s'ancrent naturellement dans le modèle à plusieurs identités que nous avons décrit, dans lequel chaque identité peut être concernée par l'une de ces applications. Par ailleurs, comme cela sera souligné, la signification que possède une identité TLS peut varier considérablement selon le contexte rencontré, notamment dans les cas où les identités sont associées à des machines.

Partie III : Applications concrètes du modèle d'identité numérique fondé sur la carte EAP-TLS

Chapitre 6 : L'identité TLS pour l'authentification dans le Cloud Computing

6.1 Introduction

Concept devenu très en vogue au cours des dernières années, le Cloud Computing[92] est souvent associé à une vision abstraite de services du réseau Internet. Pourtant, ce qu'il désigne à l'origine n'a rien d'une révolution technologique, puisqu'il s'agit, pour un serveur donné, de confier la réalisation de traitements et opérations informatiques à des serveurs distants, potentiellement administrés par des tiers. Autrement dit, le Cloud Computing consiste à louer la gestion d'un service donné. Il peut s'agir d'équipements ou de services de couches basses, ce qui constitue alors l'IaaS (*Infrastructure as a Service*), d'outils de développement sous forme de plateformes, ce qui constitue le PaaS (*Platform as a Service*), ou bien encore simplement des applications, qui constituent alors le SaaS (*Software as a Service*).

Dans une architecture s'inscrivant dans le Cloud Computing, les questions relatives à la sécurité et à la confiance se posent naturellement, car la délégation de la gestion de données à un tiers implique la définition d'une politique claire relative à la confidentialité de ces données, et la mise en place de contre-mesures associées à cette politique. Il est possible, en effet, de ne pas souhaiter laisser la possibilité à la tierce partie d'accéder aux données qu'elle stocke sans que cela nuise à la mission qui lui est confiée, par exemple la gestion des accès à des ressources protégées suite à une authentification.

Dans ce contexte, le cas de l'authentification centralisée auprès d'un serveur RADIUS a attiré notre attention. Largement déployé dans des architectures de type 802.1X (cf. chapitre 3), le serveur RADIUS réalise l'authentification d'un client souhaitant accéder à une ressource protégée d'un réseau. Or, il est tout à fait envisageable de considérer ce type d'authentification dans le contexte du Cloud Computing, où le serveur RADIUS serait administré par un tiers. Néanmoins, il est difficile, alors, d'empêcher ce tiers d'espionner les échanges ou d'accéder aux données sensibles relatives à l'authentification des utilisateurs, surtout dans un modèle symétrique. Mais un modèle asymétrique ne rendrait pas davantage le serveur RADIUS à l'abri d'attitudes malveillantes, car la clé privée du serveur reste soumise, en étant stockée directement sur le terminal, aux mêmes faiblesses que nous évoquions dans le chapitre 1.

Une telle architecture se prête assez naturellement à l'emploi de la carte EAP-TLS, à présent en mode serveur, en guise de moyen d'authentification auprès du serveur RADIUS. Nous décrivons, dans ce chapitre, une solution reposant sur une grille de 32 cartes EAP-TLS en mode serveur et en présentons les performances mesurées.

Nous présentons également une architecture voisine, à la finalité différente mais reposant sur le même principe de grille de cartes. Cette architecture, dépourvue à présent de serveur RADIUS, viserait à déléguer à un tiers la génération de jetons SAML après une authentification mutuelle TLS réussie, montrant ainsi la possibilité de lier identité TLS, Cloud Computing et fédération d'identités, tout en soulignant le caractère convergent de notre modèle d'identité par l'utilisation d'une technologie de fédération différente de OpenID, qui a fait l'objet de la plateforme de démonstration principale décrite au chapitre précédent.

Dans ce chapitre, nous traitons ainsi la problématique consistant à mettre au point une architecture de *Trust as a Service* permettant la délégation d'opérations et d'équipements à un tiers tout en assurant la confidentialité et l'intégrité des données auxquelles le tiers pourrait avoir accès. Les deux cas d'utilisation envisagés sont celui de l'authentification auprès d'un serveur RADIUS administré par un tiers, et la génération et distribution par un tiers de jetons pour la fédération d'identité, avec l'exemple de SAML.

6.2 Une architecture d'authentification auprès d'un serveur RADIUS

6.2.1 Contexte

Nous nous plaçons ici dans le cadre d'une authentification visant à permettre l'accès à une ressource d'un réseau. Cette authentification, réalisée dans le cadre d'une architecture 802.1X[75], emploie, conformément à ce standard, un serveur RADIUS[76], dont nous supposons qu'il a été confié à une tierce partie, en charge de sa seule maintenance. Dans ce contexte, nous ne souhaitons pas que ce tiers puisse s'attaquer aux données sensibles contenues sur ce serveur, ni même les consulter.

A la suite de la présentation de notre modèle d'identité, et notamment de ses apports théoriques en termes de sécurité, l'utilisation d'une carte à puce EAP-TLS du côté serveur nous a semblé être un parti pris assez naturel. De cette manière, toute information ou donnée critique mobilisée dans la procédure d'authentification reste scellée, avec la garantie que le tiers ne pourra ni en avoir connaissance, ni la modifier. Néanmoins, dans le cas où un seul serveur RADIUS centralise les demandes d'authentification d'un grand nombre d'utilisateurs, l'utilisation d'une unique carte EAP-TLS est notoirement insuffisante pour traiter les accès concurrents. Il est ainsi nécessaire de déployer un nombre suffisamment conséquent de cartes.

Dans le cadre d'une plate-forme de démonstration, nous avons opté pour 32 Javacard Gemalto TOP IM GX4[112], placées dans une grille au format SIM. Cette grille de cartes, pouvant en réalité accueillir jusqu'à quatre cents cartes à puce, est hébergée en Allemagne derrière un serveur frontal réalisant le lien avec chacune des cartes de la grille.

Habituellement, ce type de grille est utilisé par les constructeurs de téléphones mobiles afin de vérifier leur compatibilité avec les cartes SIM émises par les différents opérateurs. D'autres architectures impliquant une grille de carte à puces ont également vu le jour, comme dans [89] qui combine la puissance de calcul d'un groupe de Javacards afin de générer un ensemble de Mandelbrot. Toutefois, l'usage que nous faisons ici de la grille, où chaque carte constitue un serveur indépendant, est entièrement original.

6.2.2 Présentation de l'architecture

6.2.2.1 Vue générale de l'architecture

Résultat d'un travail présenté dans [90], la première version d'un serveur RADIUS fonctionnant avec des cartes EAP-TLS en mode serveur était constituée, comme le montre la figure 27, d'un hub USB dans lequel étaient insérés 3 cartes EAP-TLS ainsi que le serveur RADIUS proprement dit, dont le code était contenu dans la mémoire Flash d'un composant USB. Bien que fonctionnel, ce prototype restait incompatible avec le format de la grille de cartes qui nous a été proposée, et pour laquelle nous avons légèrement modifié ce modèle.

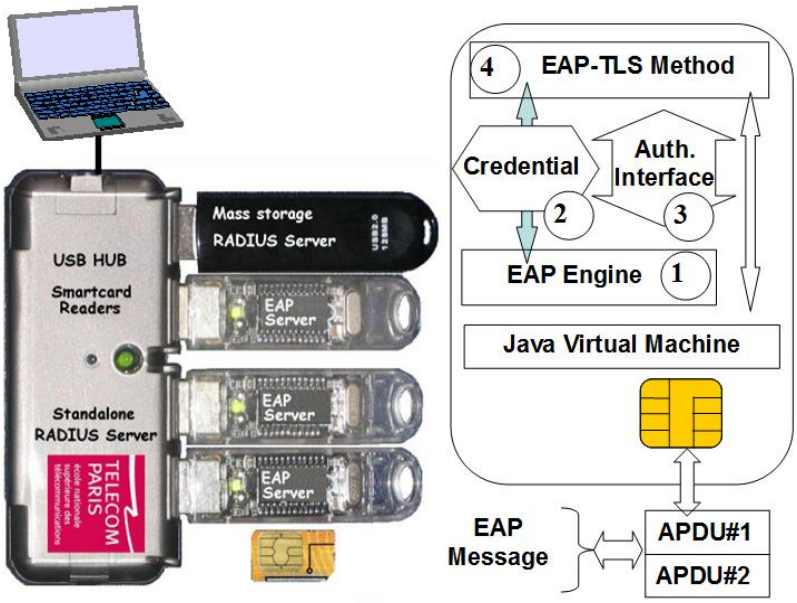

Figure 27 : Serveur RADIUS fonctionnant avec des cartes EAP-TLS serveur

La principale modification a consisté à séparer ce bloc unique constitué par le serveur RADIUS et ses cartes EAP-TLS en deux parties bien distinctes, et physiquement éloignées :

- Une partie purement logicielle traite le protocole RADIUS
- Une partie, constituée par la grille de cartes, traite l'authentification EAP-TLS.

Un lien virtuel est établi entre la partie RADIUS et chacune des cartes de la grille par l'intermédiaire de sockets TCP. Une connexion TCP est ouverte pour chaque carte devant être utilisée pour une authentification. Dans le cas où toutes les cartes sont occupées, la demande d'ouverture de connexion TCP est rejetée, bien qu'il soit également possible de la mettre en attente de la libération d'une ressource. Une vue synthétique de l'architecture distribuée alors obtenue est illustrée sur la figure 28.

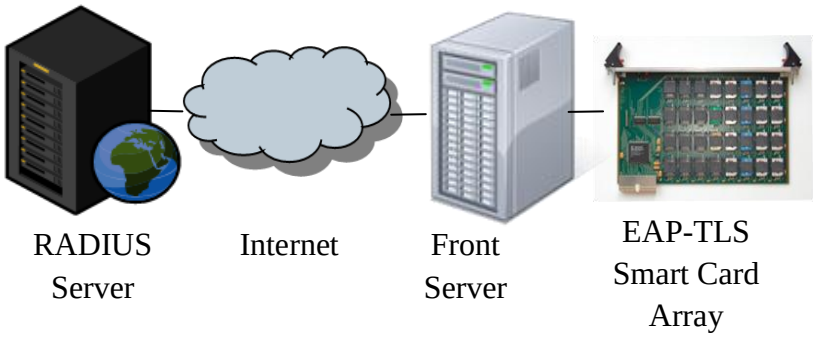

Figure 28 : Architecture distribuée d'un serveur RADIUS

6.2.2.2 Fonctionnement détaillé de la plateforme

Une telle architecture vise à fournir un haut niveau de confiance en réalisant une authentification entre une carte EAP-TLS client, et l'une des cartes EAP-TLS serveur, de manière à mettre à profit les avantages du composant sécurisé des deux côtés de la connexion. Néanmoins, il est difficile de réaliser des tests de montée en charge à partir de plusieurs dizaines de cartes EAP-TLS client, qui pour des raisons de gestion correcte du parallélisme doivent être distribuées sur plusieurs d'ordinateurs – rendant ainsi élevé, voire prohibitif, le coût de déploiement d'une telle plateforme de démonstration. Par conséquent, nous avons monté une plateforme de test à partir de clients simulés par le logiciel *OpenSSL*, dont le TLS est artificiellement traité par plusieurs opérations d'encapsulation avant d'être transmis au serveur RADIUS proprement dit. Ces opérations sont effectuées par un pont logiciel que nous appellerons NAS (*Network Access Server*) par la suite, indispensable pour cette simulation, mais sans objet dans le cas d'une expérience à partir de cartes EAP-TLS client. L'ensemble des éléments mobilisés dans cette plateforme, ainsi que la représentation intégrale du scénario des échanges entre chacun de ces éléments, sont illustrés sur la figure 29.

Il n'est pas nécessaire de revenir sur les opérations réalisées par la partie serveur EAP, le fonctionnement de la carte EAP-TLS ayant déjà été détaillé dans le chapitre 3. Seules deux différences existent par rapport à cette description :

- Le fait que les cartes EAP-TLS sont déployées en mode serveur.
- Le fait que le logiciel proxy réalisant l'interface avec la grille de cartes est de nature différente de celui qui a été décrit. De fait, bien qu'il soit nécessaire de faire tourner en permanence un

daemon en attente de connexions entrantes, ce proxy ne réalise aucune opération relative à une traduction de standards, comme cela était le cas avec TLS et EAP-TLS. Cela est dû au rôle tenu par le serveur RADIUS, explicité ci-dessous. Dans la suite de ce rapport, nous désignerons ce proxy par l'appellation plus générale de « serveur frontal ».

Ainsi, la partie dédiée au traitement du protocole RADIUS doit en réalité accomplir les tâches suivantes, que nous décrivons d'abord de manière synthétique :

- Envoi et réception des paquets RADIUS via des sockets UDP[91] ouverts pour communiquer avec le NAS.
- Vérification de la signature des paquets RADIUS émis par le NAS.
- Gestion de l'encapsulation du protocole RADIUS en vue des échanges avec les cartes EAP-TLS.
- Génération et traitement des APDUs échangées avec les cartes EAP-TLS par l'intermédiaire de sockets TCP.

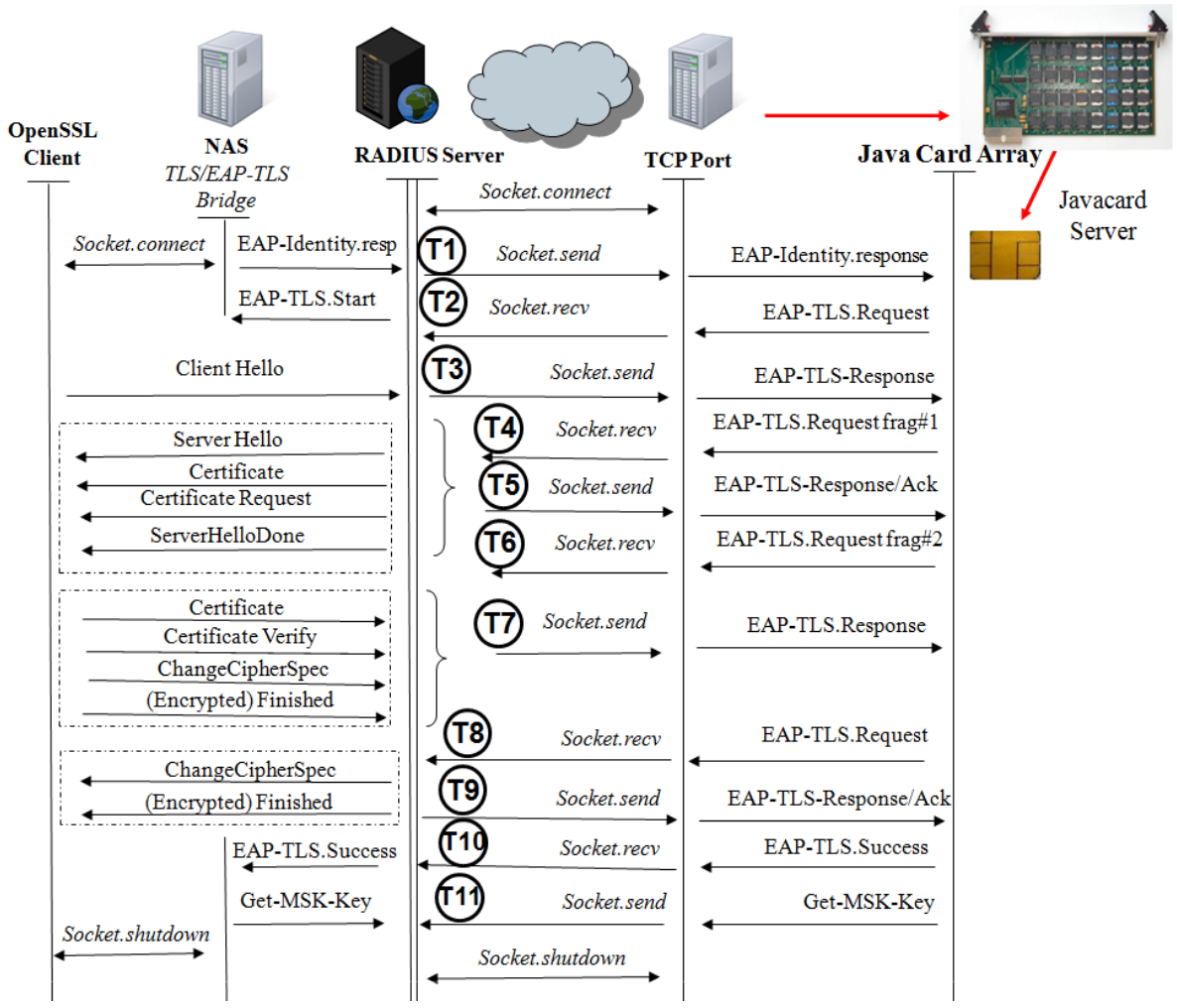

Figure 29 : Scénario d'authentification sur la grille de cartes EAP-TLS serveur

Comme nous l'avons souligné, la connexion simultanée à plusieurs cartes de la grille n'a été possible que par simulation via *OpenSSL*, qui impose la présence du NAS. Un poste client usuel peut traiter convenablement jusqu'à 30 ou 35 ouvertures de connexions TLS simultanées. Au-delà de ce nombre, la puissance de calcul de l'ordinateur fait défaut et nuit à la précision des résultats obtenus. Afin d'alléger la charge d'un poste, il est également possible de répartir les ouvertures de connexions simultanées sur plusieurs ordinateurs, mais à l'échelle de notre expérimentation, cela a un impact sur les performances mesurées, présentées, dans la section 6.4.

Chacune des connexions ouvertes par un client OpenSSL est reçue par le NAS, qui réalise une traduction entre TLS et EAP-TLS, et qui gère également l'encapsulation (et la désencapsulation) RADIUS avant de communiquer avec le serveur RADIUS, localement ou à distance – localement dans le cas de notre expérience. Ce serveur, qui tourne dans un *thread* sur le port 1812, attend des connexions UDP entrantes. A chaque nouvelle connexion reçue, il crée un *thread* initiant une connexion avec l'une des cartes EAP-TLS de la grille, en déroulant le scénario suivant (cf. figure 29) :

- Réception du message RADIUS en provenance du NAS et vérification du format et de la signature (attribut 80).

- Extraction de l'attribut 79 du message RADIUS, qui contient le message EAP encapsulé à retransmettre à l'une des cartes de la grille.

- Fragmentation du message EAP en un nombre approprié d'APDUs qui sont construites à ce moment.

- Conversion des APDUs en un format ASCII spécifique reconnaissable par le serveur frontal (proxy). Ce format comprend, en particulier, un numéro identifiant l'une des cartes de la grille à qui les données doivent être envoyées.

- Association d'un identifiant de session RADIUS avec une carte EAP-TLS spécifique de la grille, de manière à ce que la réception de toute ouverture de connexion soit associée à la même carte tout au long de la procédure d'authentification. Cette carte se trouve alors dans un état « bloqué », au sens où elle ne peut accueillir aucune connexion en provenance d'un autre client. Lorsque l'authentification est achevée, lorsque les clés cryptographiques ont été générées ou bien lorsque la notification d'échec de l'authentification est émise, la carte de la grille ayant été sollicitée est libérée. Elle peut désormais être de nouveau réquisitionnée par une nouvelle demande d'authentification EAP-TLS.

- Création de sockets TCP pour établir la connexion avec le serveur frontal placé devant la grille de cartes. Les APDUs, converties au format attendu par le serveur, sont envoyées à travers ces sockets, qui reçoivent également les réponses émises par les cartes de la grille.

- Lors de la réception d'APDUs de type EAP-Request en provenance de l'une des cartes, réassemblage des différents fragments EAP en un paquet unique, s'il y a lieu.

- Encapsulation du paquet EAP reconstitué dans un message RADIUS avant retransmission vers le NAS, qui appliquera les traitements adéquats avant retransmission vers le client TLS. Fermeture du *thread* qui avait été ouvert.

L'ensemble de cette procédure est renouvelé autant de fois que nécessaire pour traiter l'ensemble des sessions aboutissant à l'authentification (ou à son échec) de chacun des clients. Dans le cas où une erreur se produit au cours de la procédure d'authentification, le traitement est le même que dans le cas d'un échec de cette dernière, ce qui implique notamment la libération immédiate de la carte serveur réquisitionnée.

6.2.2.3 Rôle de l'identité dans les cartes serveur

Dans la plateforme que nous avons montée, les cartes EAP-TLS serveur ont toutes une identité par défaut que nous n'exploitons pas dans notre scénario. Néanmoins, cette identité peut avoir beaucoup plus d'importance si l'on considère le cas où des cartes relatives à des serveurs différents cohabitent au sein de la grille de cartes. Cela peut être le cas si plusieurs entreprises délèguent leur procédure d'authentification au même tiers.

Dans un tel contexte, c'est l'identité chargée dans chaque carte qui permettra de faire la différenciation. Notre modèle de conteneur d'identité conserve donc tout son sens dans ce type d'architecture, même s'il est *a priori* plus difficile d'envisager ici des cas pratiques de déploiement de cartes à plusieurs conteneurs d'identités.

6.3 Une architecture pour la génération sécurisée de jetons SAML

En reprenant la plupart des principes sur lesquels repose l'architecture précédente, il est possible de proposer une architecture alternative qui remplit un objectif différent, en s'inscrivant davantage dans la problématique de la gestion d'identités. La plateforme de démonstration établissant une preuve de concept de notre modèle d'identité, présentée au chapitre 5, mettait à contribution la technologie OpenID. Nous proposons à présent une architecture qui intègre la technologie SAML. Bien qu'elle ait fait l'objet d'une étude théorique à part entière, cette architecture n'a toutefois pas fait l'objet d'une implémentation. La principale raison à cela est la proximité qu'elle entretient à la fois dans la forme, avec la mouture qui a été présentée au paragraphe précédent, et dans le fond, avec la plateforme de démonstration OpenID présentée au chapitre 5. Or, dans les deux cas, les plateformes ont été entièrement montées et ont fait l'objet de mesures de performances, auxquelles les chiffres que nous aurions obtenus avec la présente architecture n'auraient pas apporté d'information pertinente.

Concrètement, la grille de cartes EAP-TLS serveur est conservée à l'identique, toujours placée derrière un serveur frontal qui est en charge de la communication avec chacune des cartes de la grille. En revanche, la partie que nous avions consacrée au traitement du protocole RADIUS est

nettement modifiée, notamment par la suppression du serveur RADIUS. Cependant, l'ensemble des traitements logiciels relatifs à la réception de demandes de connexion et à la gestion de la fragmentation des messages EAP-TLS en APDUs sont conservés.

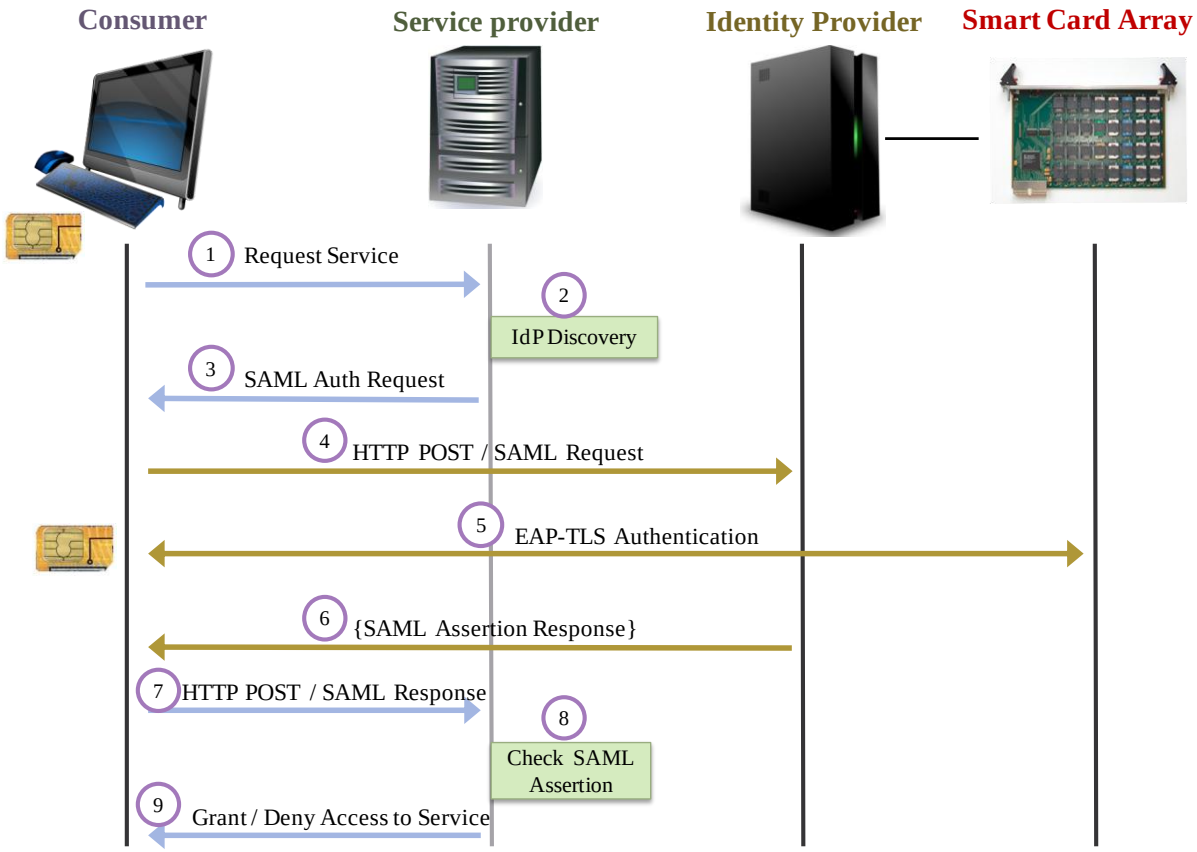

Figure 30 : Scénario d'obtention d'un jeton SAML par authentification sur une grille de cartes

Un scénario pratique correspondant à une telle architecture, illustré sur la figure 30, pourrait être celui du *Web Browser SSO Profile* de la technologie SAML (cf. chapitre 2), dont le principe est très similaire à celui évoqué pour le cas d'OpenID :

- Un utilisateur cherche à s'authentifier auprès d'une ressource réseau (SP) via une authentification mutuelle EAP-TLS. Pour cela, il fait usage d'une carte client en sa possession, dont l'identité active sera celle pour laquelle il cherchera à obtenir l'accès à cette ressource.

- La procédure d'authentification est redirigée par l'intermédiaire d'une requête SAML vers un IdP mettant à profit la grille de serveurs, soit directement, soit par délégation à un tiers. Une carte libre est alors réquisitionnée pour l'authentification. De même que précédemment, si aucune carte de la grille n'est disponible, la demande est soit ignorée, soit mise en attente.

- Dans le cas où l'authentification mutuelle EAP-TLS est réussie, une assertion SAML appropriée à l'identité authentifiée est générée par l'IdP et retransmise à l'utilisateur sous la

forme d'une réponse SAML chiffrée dans le tunnel TLS ainsi établi avec l'IdP. Cette assertion est alors réémise vers le SP sous la forme d'une requête POST pour obtenir l'accès au service souhaité.

Bien que cette architecture soit très semblable à la précédente, ce contexte de déploiement permet de donner une meilleure vision de la compatibilité directe qu'elle possède avec notre modèle d'identité numérique. Nous retrouvons en effet la problématique de la fédération d'identité et de l'authentification sécurisée, pour laquelle nous sécurisons également à présent le côté serveur, dans l'idée d'établir un dialogue *de carte à carte*. De plus, l'architecture présentée au chapitre précédent, intégrant un IdP OpenID, pourrait être aisément adaptée pour correspondre à celle que nous venons de décrire, i.e. pour un IdP générant des jetons SAML.

La question de l'environnement de confiance, déjà résolue pour le côté client, trouve ici un écho direct au niveau du serveur. De ce point de vue, une telle architecture, tant dans le contexte précédent que dans celui-ci, s'inscrit pleinement dans la mouvance du Cloud Computing en proposant le *Trust as a Service*.

6.4 Performances mesurées

La mesure des performances vise spécifiquement à obtenir une appréciation de la gestion d'authentifications concurrentes par la grille de cartes, indépendamment du cas d'utilisation considéré. Par conséquent, seule l'architecture mettant en place un serveur RADIUS, décrite en section 6.2, a fait l'objet d'un véritable déploiement et de mesures afférentes. Les cartes EAP-TLS serveur utilisées sont des Javacards Gemalto TOP IM GX4[112], dont les performances sont ici moins bonnes que les mesures présentées dans le chapitre 3. Cela est dû à la chronologie des mesures, car cette plateforme a été montée lorsque le code Javacard de l'application EAP-TLS n'était pas optimisé – la RAM permettant des lectures et écritures nettement plus rapides que l'EEPROM, mais étant limitée dans ces cartes à seulement 800 octets environ. Par conséquent, lorsqu'elle est directement reliée au poste client par l'intermédiaire d'une interface USB, une carte embarquant cette version de l'application EAP-TLS nécessite entre 5,5s et 6,0s pour réaliser une authentification mutuelle TLS en mode *full*, alors que nos meilleures performances étaient de l'ordre de 2,0s.

Chacun des temps indiqués dans la suite de cette section, de T1 à T11, fait référence à la figure 29. Les mesures se placent au niveau de l'échange des paquets EAP encapsulés dans les APDUs. Qu'il s'agisse de paquets EAP-Requests (Tx) ou EAP-Responses (Rx), la méthodologie employée pour la mesure consiste à prendre en compte le temps d'émission ou de réception de l'ensemble des APDUs contenant le paquet EAP éventuellement fragmenté, jusqu'au renvoi des Status Words correspondant à la dernière APDU. Une vue schématique de cette méthodologie est illustrée sur la figure 31.

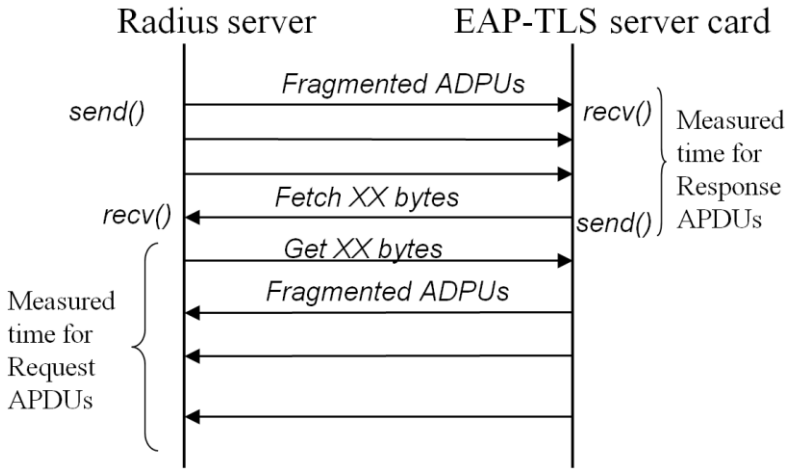

Figure 31 : Méthode de mesure des temps de transmission

La première mesure effectuée sur cette plateforme s'attache aux performances d'une seule authentification sur la grille. En toute logique, les performances ne doivent pas être sensiblement différentes d'une authentification sur une carte EAP-TLS serveur locale, car seule s'ajoute, a priori, la distance entre notre serveur RADIUS, et le serveur frontal qui repose en Allemagne – cela, qui plus est pour une quantité de données échangées relativement faible. Or, ces premières observations se sont avérées être très éloignées de nos prévisions. Le temps pour l'authentification sur une carte de la grille est en effet d'environ 10,0s, soit presque le double du temps nécessaire pour le cas d'une carte serveur locale. Le tableau 6 présente le détail du comparatif des performances entre ces deux cas de figure.

	USB Card	Distant Card
T1- Rx: EAP-Identity.response	30ms	100ms
T2- Tx: Start	5ms	60ms
T3- Rx: Client Hello	430ms	580ms
T4- Tx: Server Hello fragment#1	220ms	1850ms
T5- Rx: EAP-TLS-ACK	40ms	100ms
T6- Tx: Server Hello fragment#2	10ms	200ms
T7- Rx: Client-Finished	270ms	2100ms
T8- Tx: Server-Finished	4320ms	4500ms
T9- Rx: EAP-TLS-ACK	290ms	350ms
T10- Tx: EAP-Success	20ms	60ms
T11- Rx: Get-PMK	20ms	190ms
Total	**5655ms**	**10090ms**

Tableau 6 : Différences de performance entre une session EAP-TLS établie avec une carte locale et une carte distante

Il est impossible d'expliquer une telle différence par le seul temps de transmission des données à travers le réseau. Le temps d'un *ping* sur le serveur frontal est en effet d'environ 30ms. Les paquets TCP échangés pour l'ensemble du transport des APDUs étant de taille très faible, il est possible d'approximer le temps de transmission d'un paquet TCP à celui du *ping*, soit 30ms. Or, au total, 26 paquets TCP sont échangés pour l'ensemble d'une session TLS en mode *full*, ce qui induit un temps de transfert d'environ 780ms. Par ailleurs, la taille des données échangées s'élève à environ 2500 octets, dont le temps de transmission peut être négligé compte tenu du débit en place dans les réseaux à l'heure actuelle. Autrement dit, même en comptant de manière un peu plus large, l'authentification sur une carte distante devrait être *au plus* 1,0s plus longue que sur carte locale, ce qui est loin de correspondre à nos observations.

Le détail des mesures de temps donné dans le tableau 6 nous permet de constater que les retards les plus importants sont associés aux paquets les plus conséquents. Par exemple, T7 est très faible dans le cas d'une authentification avec une carte locale. Cependant, dans le cas de l'authentification distante, ce temps est presque dix fois plus long. Or, le paquet considéré ici est l'envoi du certificat client, i.e. l'un des messages les plus conséquents envoyés au serveur. Après investigation, il s'avère que le serveur frontal communique avec la grille de cartes par un lien à 9600 bauds. La taille des données échangées au cours d'une authentification s'élevant à environ 2500 octets, les contraintes *hardware* que nous rencontrons coûtent ainsi environ 2500ms. Par ailleurs, un délai est induit par la partie logicielle du serveur frontal, en charge de la redirection des messages vers les cartes de la grille. D'après l'ensemble des mesures et des estimations précédentes, ce délai s'élèverait à environ 1000ms.

D'autres mesures confirment les précédentes constatations. Nous avons ici lancé plusieurs authentifications simultanées, respectivement 5 et 20. Si tout se déroulait de manière optimale, nous devrions observer un temps moyen d'authentification comparable à celui d'une authentification unique, puisque l'objectif de l'utilisation de cette grille est de pouvoir paralléliser les opérations. Or, une fois de plus, les résultats divergent par rapport à la théorie. Le tableau 7 montre le détail des mesures temporelles moyennes de l'une des authentifications lancées, respectivement dans le cas de 5 connexions et de 20 connexions.

Il est aisé de constater que les paquets les plus petits, tels que ceux représentés par T9 et T10, sont ceux qui varient le moins lorsque l'échelle de l'expérience augmente. En revanche, l'échange de certificats, représenté par T4 et T7, induit de tels délais qu'il n'est pas envisageable de réaliser un passage à l'échelle à partir de cette plateforme.

Néanmoins, après investigations complémentaires, nous avons pu nous assurer que le serveur frontal ne parallélisait pas les paquets EAP-Requests, mais seulement les paquets EAP-Responses. Si l'on ajoute à cela le faible débit du lien entre le serveur frontal et la grille de cartes, la majorité des problèmes rencontrés au cours de cette expérience peuvent être résolus par une simple mise à jour de la partie serveur afin de corriger ces défauts.

	1 Card out of 5	1 Card out of 20
T1- Rx: EAP-Identity.response	220ms	580ms
T2- Tx: Start	100ms	390ms
T3- Rx: Client Hello	580ms	1300ms
T4- Tx: Server Hello fragment#1	2000ms	6300ms
T5- Rx: EAP-TLS-ACK	550ms	2200ms
T6- Tx: Server Hello fragment#2	400ms	1750ms
T7- Rx: Client-Finished	6500ms	21500ms
T8- Tx: Server-Finished	5000ms	6600ms
T9- Rx: EAP-TLS-ACK	350ms	350ms
T10- Tx: EAP-Success	60ms	60ms
T11- Rx: Get-PMK	190ms	200ms
Total	**15950ms**	**41230ms**

Tableau 7 : Temps moyen d'établissement d'une session EAP-TLS avec une carte distante avec une concurrence de 5 et 20 sessions

6.5 Conclusion

Les deux plateformes présentées dans ce chapitre ne représentent qu'une déclinaison du concept du *Trust as a Service*, représenté par une architecture entièrement adaptée au Cloud Computing mettant en avant les mérites d'une authentification forte réalisée dans un environnement de confiance. Les avantages qu'il est possible d'en tirer sont, pour une part, ceux que nous avons présentés dans le chapitre présentant le concept de la carte EAP-TLS (cf. chapitre 3), mais aussi, de manière plus spécifique à ce paradigme d'architecture, la garantie de protection des données ayant été indirectement confiées à un tiers.

Par ailleurs, notre modèle d'identité trouve sa place à la fois du côté serveur, où chaque carte peut identifier une entreprise donnée, et du côté client, pour lequel l'identité à authentifier est typiquement contenue dans une carte EAP-TLS selon les principes de gestion que nous avons énoncés dans le chapitre 4 de ce rapport. Comme nous l'avons souligné, bien qu'il soit difficile à déployer directement dans le paysage du Web, ce paradigme de grille de cartes serveur est en réalité plus fort, du point de vue de la sécurité, que la plateforme de démonstration évoquée au chapitre 5, car il met en place une authentification mutuelle de carte à carte. Autrement dit, le protocole TLS est exécuté de part et d'autre dans un environnement de confiance, ce qui solidifie d'autant le déroulement de l'authentification.

Nous avons ainsi souhaité reprendre cette idée dans un contexte très différent, à savoir celui du prépaiement, dans des architectures visant à dématérialiser le format papier encore déployé à cet effet à l'heure actuelle. Ce cas d'utilisation, orienté vers le M2M, fait l'objet du chapitre suivant de ce rapport, et permet de mettre en évidence une nouvelle conception de l'identité, relativement au modèle que nous avons conçu.

Chapitre 7 : Une architecture pour le prépaiement sécurisé

7.1 Introduction

L'économie repose sur l'utilisation conjointe de deux types de monnaie complémentaires. D'un côté, la monnaie dite *fiduciaire* (du latin *fides*, confiance), constituée par exemple de billets de banque dont la valeur n'est pas contenue dans le support papier mais par la confiance du public envers ce que celui-ci représente. Cette monnaie est majoritairement utilisée pour les transactions d'un montant relativement peu élevé. De l'autre côté, la monnaie dite *scripturale* renvoie à la certification écrite d'un montant dont dispose un individu au sein d'un dépôt, par exemple un compte bancaire.

Si la monnaie fiduciaire est par définition employée pour des paiements en temps réel, i.e. sans qu'interviennent des opérations de débit ou de crédit *a priori* ou *a posteriori* de la transaction, les moyens de paiement associés à la monnaie scripturale sont en revanche conditionnés à des transactions différées, comme le détaille un bulletin de la Banque de France consacré à la monnaie électronique [93]. De fait, la carte bancaire ou le chèque bancaire sont des instruments de post-paiement, car le débit effectué sur le compte de l'individu intervient postérieurement à la conclusion de la transaction.

Le prépaiement s'inscrit dans la logique inverse, en proposant un mode de paiement où le débit a lieu antérieurement à la transaction. Le prépaiement se matérialise typiquement sous la forme de tickets émis par un organisme à la suite d'un paiement et visant à certifier ce dernier. Un chèque de banque, par exemple, est émis suite au débit du compte de l'individu, et garantit au bénéficiaire l'assurance du provisionnement dudit chèque – sauf en cas de faillite de la banque émettrice. Contrairement aux systèmes de post-paiement, en effet, c'est ici l'organisme émetteur du ticket qui s'engage à remplir ses engagements, et non l'individu qui en fait usage. Par ailleurs, le prépaiement est souvent déployé dans des contextes où le type de transaction réalisable à partir du ticket est restreint à certains types de bien. Une carte de téléphone prépayée constitue une bonne illustration de ce principe.

D'un point de vue administratif, il est possible de diviser les systèmes de prépaiement en deux catégories : centralisés et distribués.

Les systèmes centralisés, tels que les cartes téléphoniques prépayées [98], reposent sur l'utilisation de numéros de série. Les tickets émis comportent alors typiquement un numéro confidentiel devant être découvert par grattage. Le cycle de vie du numéro de série est géré par une base de données dont l'interface peut être par exemple un serveur RADIUS. L'utilisateur est

enregistré dans le système, et son compte est crédité de la valeur associée au numéro de série. Par la suite, la gestion des transactions associées au compte est effectuée *online*.

Dans les infrastructures distribuées, les transactions sont réalisées *offline*. L'individu reçoit généralement un ticket pourvu d'un numéro de série et de preuves d'authenticité typographiques – ticket qui constitue alors le moyen de paiement. Le numéro de série peut contenir un élément cryptographique, par exemple issu du calcul d'une signature électronique. Un exemple de système distribué est celui des chèques émis par des entreprises spécialisées dans les services prépayés pour la restauration.

Dans ce chapitre, après une brève présentation générale des systèmes de paiement, nous proposons une architecture visant à dématérialiser les transactions d'un système distribué, en remplaçant les tickets par des jetons numériques transportés de manière sécurisée au moyen de cartes EAP-TLS. TLS étant, de fait, le standard établi pour la sécurisation du commerce électronique, son utilisation dans le contexte du prépaiement est d'autant plus naturelle. Par ailleurs, l'identité de l'individu souhaitant effectuer un paiement est directement associée à sa carte EAP-TLS par l'intermédiaire du modèle que nous avons présenté précédemment.

7.2 Les systèmes de paiement

Ainsi qu'il est décrit dans [94], « le commerce implique toujours la présence d'un *payeur* et d'un *marchand*, qui échangent de l'argent pour des biens et des services, mais également celle d'une institution financière qui relie les *bits* à l'*argent* ». En pratique toutefois, ce sont généralement deux institutions financières qui sont impliquées dans le déroulement d'une transaction : un *émetteur*, qui agit pour le *payeur*, et un *acquéreur*, qui est lié au *marchand*.

7.2.1 Généralités sur les transactions par carte bancaire

Un paiement par carte bancaire s'effectue en deux temps [95], le *flux de transaction* et le *règlement*. Le *flux de transaction* a pour objectif de mettre en place une authentification. L'émetteur et l'acquéreur sont connectés par l'intermédiaire d'un réseau de cartes de paiement (PCN, *Payment Card Network*) géré par des entreprises de cartes bancaires – lesquelles peuvent également jouer le rôle d'acquéreur. Le marchand rassemble les informations associées à la carte du payeur via un terminal, et les soumet à l'acquéreur. Ces informations comprennent, entre autres, le numéro de la carte, la date d'expiration, et le type de carte. De manière optionnelle, l'acquéreur les envoie à l'institution de l'émetteur à travers le PCN, pour que ce dernier vérifie l'état du compte à débiter. Une réponse est alors renvoyée à l'acquéreur qui, en cas de succès, renvoie un code d'*autorisation* de la transaction au terminal du marchand.

Le *règlement* fait quant à lui référence aux opérations de transfert de fonds qui ont habituellement lieu dans un délai de plusieurs jours à compter de la conclusion de la transaction.

7.2.2 Les questions de sécurité

Le *Payment Card Industry Data Security Standard* [96] spécifie les obligations de sécurité des industries de cartes de paiement en les classifiant en six groupes :

1) Construire et maintenir un réseau sécurisé, par le déploiement d'infrastructures physiques (e.g. firewalls) et logiques (VPN).

2) Protéger les données du détenteur de la carte, par l'utilisation de la cryptographie pour le stockage et la transmission.

3) Maintenir un programme de gestion des vulnérabilités, par la sécurisation des applications Web et la mise à jour des systèmes avec des patchs de sécurité.

4) Mettre en œuvres des mesures fortes pour le contrôle d'accès, en privilégiant l'authentification à deux facteurs à base de composants ou de biométrie, au détriment des mots de passe qui doivent être évités.

5) Tester et contrôler régulièrement les réseaux.

6) Maintenir une politique de sécurité des informations.

7.2.3 Le cas de la technologie EMV

Le standard EMV[97] a été créé en 1994 par les entreprises Europay, Mastercard, et Visa, dont l'acronyme a donné son nom au standard. La plus grande part du commerce électronique est actuellement effectuée via des cartes EMV.

Celles-ci sont pourvues d'un composant sécurisé (un carte à puce) qui accompagne la traditionnelle bande magnétique et stocke des informations relatives au porteur de la carte. La puce contient également le certificat de l'émetteur, ainsi qu'une signature (nommée SSAD, *Signed Static Application Data*) calculée grâce à la clé privée associée au certificat émetteur. De manière optionnelle, la carte peut également contenir un certificat ICC (*IC Card*) créé par l'émetteur, ainsi qu'une clé privée associée (*ICC private key*), utilisée dans une procédure de challenge / réponse visant à authentifier la carte (DDA, *Dynamic Data Authentication*).

La carte EMV génère des cryptogrammes d'application (AC, *Application Cryptograms*) habituellement à partir de l'algorithme 3DES avec une clé secrète d'une longueur de 112 bits.

La transaction est initiée par le terminal du marchand, via une commande GENERATE AC, qui possède trois options :

- Demande de transaction Off-line (TC, *Transaction Certificate*)

- Demande de transaction On-line (ARQC, *Authorization Request Cryptogram*)
- Abandon de transaction (AAC, *Application Authentication Cryptogram*)

Dans le cas de figure que nous considérons, la requête du terminal sera du type ARQC, et transmet à la carte plusieurs informations telles que la date et le montant de la transaction. Si la carte accepte cette demande, elle renvoie un *Cryptogram Information Data* (CID) au terminal.

Le marchand transmet alors à l'acquéreur une demande d'autorisation qui comprend notamment l'ARQC et le CID. Selon la réponse obtenue, le terminal envoie une deuxième commande GENERATE AC visant à indiquer l'acceptation ou le refus. Dans le premier cas, la carte génère un cryptogramme de type TC, et dans le second, de type AAC. Le marchand fait alors parvenir à l'acquéreur un message de confirmation relatif à la conclusion de la transaction.

Du point de vue de la sécurité, une authentification à deux facteurs est mise en place pour protéger les transactions. D'une part, un code PIN est demandé à l'utilisateur avant la génération du premier cryptogramme par la carte, lequel indique si un code correct a été présenté ou non. D'autre part, le cryptogramme contient une signature réalisée grâce à une clé secrète stockée dans le composant sécurisé.

7.3 Proposition d'un modèle pour le prépaiement

Contrairement aux systèmes de paiement classiques dont le principe vient d'être décrit, le prépaiement a la particularité d'être un système de transactions relatives à des paiements déjà effectués. Autrement dit, l'enjeu de l'architecture est entièrement lié à la vérification et au transport sécurisé d'un jeton (équivalent numérique du ticket) matérialisant la preuve du paiement *a priori*. Nous présentons ici une solution pour le prépaiement compatible avec des cas d'utilisation concrets, tels que la restauration. Après avoir donné une vue générale de l'architecture et du déroulement d'un scénario typique, nous détaillons les différents composants mis en œuvre.

Tous les composants sécurisés déployés, notamment la carte utilisateur et le SAM, embarquent notre application EAP-TLS et sont de fait compatibles avec le modèle d'identité présenté dans la deuxième partie de ce rapport. Néanmoins, un tel contexte est peu propice à l'utilisation de multiples identités pour un utilisateur donné, sans toutefois en interdire l'existence. Par conséquent, l'identité de l'utilisateur qui sera employée ici en permanence pourra être son identité *racine*.

7.3.1 Vue générale de l'architecture

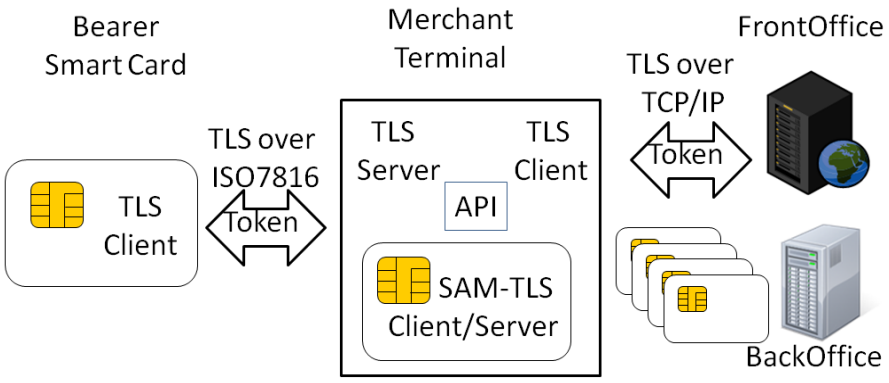

Figure 32 : Architecture pour le prépaiement

Un système de prépaiement fonctionne nécessairement par l'attribution aux individus de preuves de paiement, autrement dit de tickets émis par l'organisme ayant été crédité. Ainsi, habituellement, dans le cas de la restauration, un individu reçoit environ une vingtaine de tickets par mois, d'un montant fixe. Dans l'idée de supprimer l'utilisation de tickets sous forme papier, tout en restant aussi proche que possible des principes de fonctionnement des systèmes existants, nous considérons une architecture telle que représentée sur la figure 32. La partie serveur de cette architecture est constituée de deux entités, le *Front End* (FE) et le *Back End* (BE), respectivement contenus dans le *Front Office* et le *Back Office*. Les jetons représentant les tickets sont émis par le FE sur délégation du BE. Ces jetons sont alors chargés dans la carte client par l'intermédiaire du terminal marchand, après instauration d'un tunnel sécurisé TLS. Cette carte client est alors capable d'utiliser ces jetons en les soumettant au terminal marchand, à nouveau à travers un tunnel sécurisé TLS. Les jetons sont alors stockés dans un composant sécurisé nommé SAM (*Secure Access Module*), et embarqué dans le terminal marchand. Par intervalles de temps réguliers, e.g. à la fin de chaque journée, les différents jetons stockés dans le SAM sont transférés par tunnel sécurisé vers le FE, où ils sont vérifiés puis détruits. Le marchand est alors crédité par l'intermédiaire du BE de la valeur des jetons ainsi collectés.

7.3.2 Les jetons

Les jetons sont émis par le FE, qui obtient du BE une liste de jetons à émettre. Le cycle de vie d'un jeton est donc le suivant :

- Existence au sein du BE
- Création par le FE

- Stockage dans la carte de l'utilisateur ou bien dans le SAM

- Destruction par le FE et conservation de la trace de son existence par le BE.

Un jeton est constitué d'un certificat X509 et d'une entrée contenant la valeur monétaire en clair (précision à ajouter). Ainsi, cette valeur peut être lue par le terminal du marchand et comparée au niveau du FE après transmission du jeton à ce dernier.

De manière générale, les jetons sont systématiquement transmis à travers des tunnels sécurisés TLS, et sont stockés dans des environnements de confiance tels que la carte EAP-TLS utilisateur, le SAM, ou bien le HSM (*Hardware Security Module*) associé au FE (cf. section 7.3.4).

7.3.3 Le terminal marchand

Le terminal marchand a un rôle d'intermédiaire entre l'utilisateur et le serveur de l'organisme. Son interface avec l'utilisateur remplit deux fonctions :

1 - L'acceptation d'un paiement par jetons de la part de l'utilisateur

2 - La recharge d'une carte utilisateur vide

Dans le cadre de la première fonction, le terminal vérifie la date et la valeur des jetons transmis par la carte utilisateur. Le code PIN est optionnel à ce niveau, car selon les cas d'utilisation pratiques, il pourra être utile de pouvoir réaliser un paiement via une interface sans contact afin d'optimiser le temps nécessaire pour la mise en place de la transaction.

En revanche, la deuxième fonction ne peut être remplie que par une interface avec contact et sur présentation d'un code PIN, afin de s'assurer de l'identité du possesseur de la carte. Lorsque le PIN est saisi avec succès, le terminal contacte le FE afin d'établir un tunnel sécurisé directement entre la carte utilisateur et le FE, qui permet le rechargement sécurisé de la carte utilisateur.

L'interface entre le terminal et le serveur de l'organisme est en revanche réduite à une seule fonction, laquelle consiste à soumettre au FE l'ensemble des jetons collectés par intervalles de temps réguliers. Une ligne téléphonique, une ligne GSM, ou bien une connexion Internet relie le terminal marchand au FE. Par conséquent, le terminal marchand doit gérer trois types de flux de transaction, comme l'illustre la figure 33 : de la carte utilisateur au terminal, du terminal vers le FE, et du FE vers la carte utilisateur.

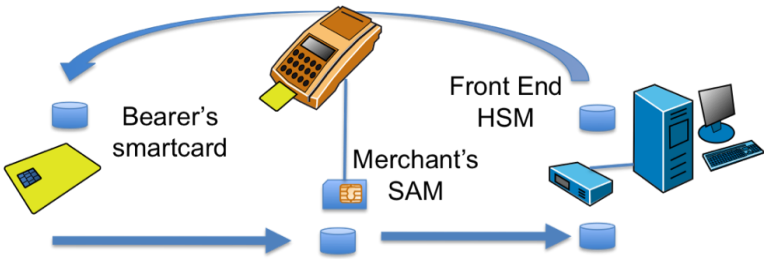

Figure 33 : Les échanges de jetons

Cela permet de mettre en évidence un élément qu'il est nécessaire de souligner : le SAM possède, alternativement, un rôle de serveur et de client, lors de la mise en place des tunnels sécurisés TLS. Ces deux rôles sont assurés par une unique identité TLS, autrement dit par un unique certificat X509. Seule intervient la sélection de l'identité en mode serveur ou en mode client. Néanmoins, un choix d'implémentation alternatif aurait pu consister à séparer ces deux modes en deux conteneurs d'identité distincts. Cet aspect, qui amène à reconsidérer la notion d'identité lorsqu'elle est associée à des machines, fait l'objet d'une discussion plus approfondie dans la conclusion de ce chapitre (section 7.6).

7.3.4 Le Front End

Relativement au vocabulaire en usage dans les systèmes de paiement, le Front End possède à la fois le rôle d'émetteur et d'acquéreur. Plus précisément :

- Il est émetteur de jetons. Comme le montre la figure 33, le FE est connecté à un composant sécurisé appelé HSM (*Hardware Security Module*). Ce composant créé les jetons devant être chargés dans le carte d'un utilisateur. Les jetons créés sont par ailleurs stockés de manière chiffrée dans la base de données du FE.

- Il est acquéreur de données en provenance du terminal marchand. De manière régulière, le terminal contacte le FE afin de transmettre les jetons accumulés par le SAM. Après vérification des jetons, le FE détruit ces derniers et notifie le BE afin que le compte du marchand soit crédité.

7.3.5 Le Back End

Le Back End joue exclusivement un rôle d'émetteur. C'est en effet cette entité qui est en charge du déploiement des cartes utilisateur d'une part, mais aussi des SAM. Il personnalise ces différentes cartes, notamment pour la définition du code PIN.

Il est également en charge de la gestion des flux en jeu dans les transactions, notamment la création de jetons et le crédit ou débit des comptes.

7.4 Scénario détaillé d'une transaction

Une transaction relative au paiement par jetons prépayés peut être découpée en deux phases : le paiement proprement dit, qui fait intervenir la carte utilisateur et le SAM, puis, postérieurement, la transmission des jetons reçus au FE.

7.4.1 Transmission des jetons clients vers le SAM

La figure 34 illustre le déroulement d'une connexion entre une carte utilisateur et le SAM, visant à transmettre des jetons prépayés. De manière similaire au cas rencontré dans le principe de la grille de cartes serveur présentée au chapitre précédent, la connexion est ici établie de carte à carte via le protocole EAP-TLS embarqué. Le SAM fonctionne ici en mode serveur.

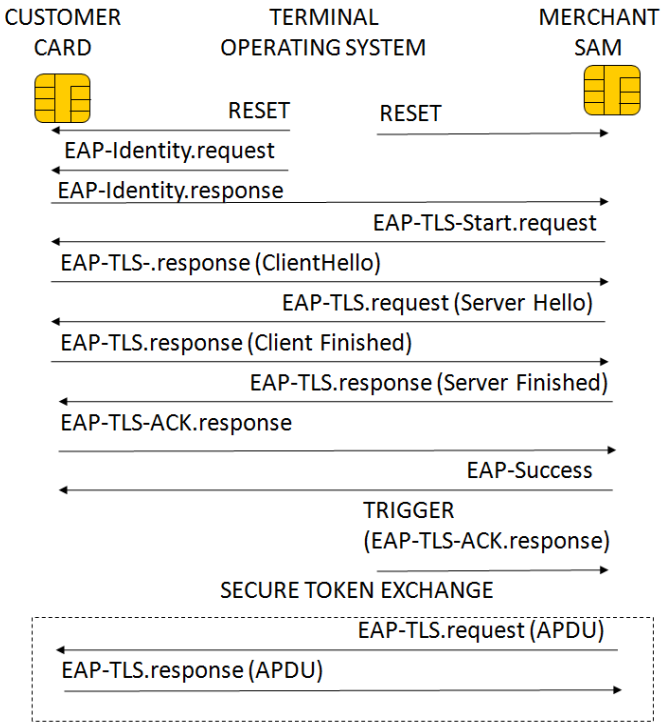

Figure 34 : Transaction entre le client et le SAM

Le terminal marchand maîtrise le déroulement de la connexion, notamment en notifiant le début de celle-ci par l'envoi d'un message RESET aux deux cartes, dont la fonction est de remettre à l'état initial la machine à états EAP. C'est également le terminal qui forge la requête *EAP-Identity.Request* envoyée au client, lequel envoie le message *EAP-Identity.Response* au SAM. Le terminal n'a ici pas d'autre fonction que de relayer les messages EAP entre les deux cartes. A la suite de cela, la carte serveur envoie un paquet *EAP-TLS-Start.Request* qui permet d'initialiser le *Handshake* TLS proprement dit, tous les paquets TLS étant encapsulés par le protocole EAP.

Lors de la notification du succès de la session EAP-TLS par la carte serveur, matérialisée par l'envoi du message *EAP-Success*, le client et le serveur sont prêts à échanger des informations chiffrées.

Le terminal forge alors artificiellement un paquet *EAP-TLS-ACK.Response* dont la véritable fonction est celle d'un *Server Trigger*. En effet, sur réception de ce message, le SAM déclenche la construction d'une APDU visant à demander au client le transfert de jetons. Cette commande est chiffrée en accord avec les paramètres de session négociés et transportée dans un paquet *EAP-TLS.Request*.

Le client peut finalement envoyer un ou plusieurs jetons via des APDUs dédiées échangées avec le SAM, APDUs chiffrées et encapsulées de la même manière que précédemment.

7.4.2 Connexion du SAM au serveur du Front End

Une fois les jetons clients reçus et stockés par le SAM, ces derniers doivent être transmis au FE. Pour cela, une connexion sécurisée est établie selon le scénario représenté sur la figure 35. A présent, le SAM fonctionne en tant que client. Contrairement au cas précédent, le serveur du FE fonctionne a priori avec le protocole TLS, bien qu'il soit également possible d'y adjoindre une carte serveur EAP-TLS afin d'unifier les éléments de l'architecture. Néanmoins, la plateforme qui a été montée n'intégrait pas cette carte. Par conséquent, le terminal marchand a ici un rôle supplémentaire de pont permettant de réaliser la transition entre les protocoles EAP-TLS et TLS, d'une manière similaire au logiciel proxy qui intervenait dans notre architecture d'authentification auprès d'un serveur Web, telle que nous l'avons présentée au chapitre 3.

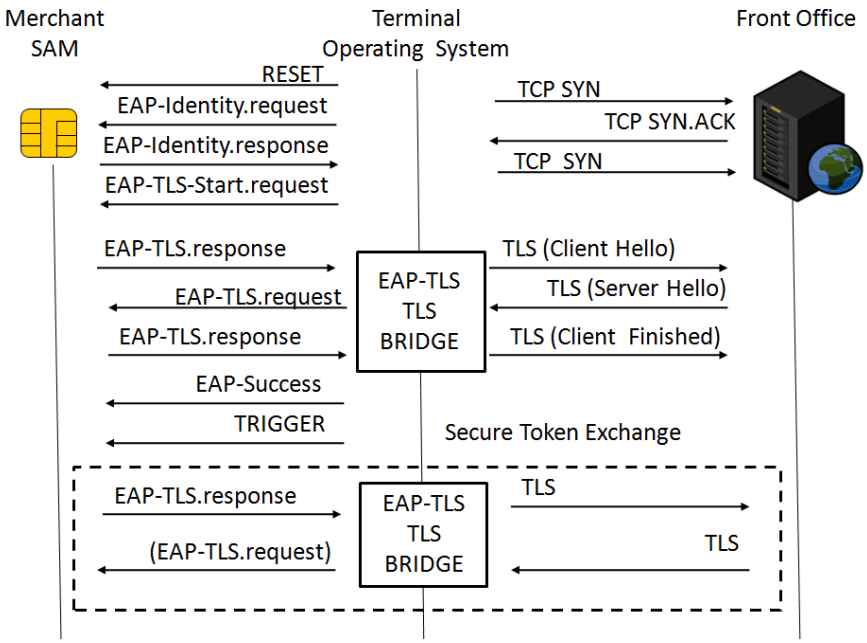

Figure 35 : Transaction entre le SAM et le Front End

Le terminal gère la connectivité TCP/IP par l'intermédiaire de sockets. Il ouvre ainsi une session TCP avec le serveur du FE sur le port 443. Parallèlement, il envoie une commande RESET au SAM, puis forge un message *EAP-TLS-Start.Request* afin d'initier la session EAP-TLS, dont le déroulement du *Handshake* TLS ne varie pas.

Lorsque la session EAP-TLS est établie avec succès, le terminal envoie, de manière similaire au scénario précédent, un message *Client Trigger* matérialisé par un paquet *EAP-TLS-ACK.Request*. Le SAM initie alors un échange d'APDUs avec le serveur visant à transmettre les jetons, l'ensemble des commandes et des jetons transportés étant chiffrés et encapsulés dans des paquets EAP-TLS.

7.5 Performances mesurées

Relativement au contexte considéré, il est nécessaire de pouvoir obtenir des performances intéressantes, notamment lors d'une transaction effectuée entre la carte utilisateur et le SAM. En effet, dans des cas d'applications concrets, par exemple la restauration, il peut se constituer des files d'attente de clients, pour l'optimisation desquelles le temps par transaction est un facteur déterminant. Il semble raisonnable de fixer une limite maximum à cinq secondes.

T_{MD5} /bloc 64 B	T_{SHA1} /bloc 64 B	$T_{RSA\ PUB}$ 128 B	$T_{RSA\ PRIV}$ 128 B	T_{3DES} /bloc 8 B	T_{AES} /bloc 16 B	T_{RC4} /byte	T_{IO} /byte
0,49	0,93	25	560	2,10	2,60	0,50	0,17

Tableau 8 : Performances cryptographiques (ms)

La plateforme déployée a fait usage, à nouveau, de Javacards Gemalto TOP IM GX4[112], dans lesquelles l'application EAP-TLS réalise une utilisation optimisée de la RAM, et dont les performances ont déjà été présentées au chapitre 3. Le tableau 8 donne à nouveau les caractéristiques cryptographiques de cette carte à puce. En reprenant la *ciphersuite* que nous avions utilisée, à savoir RC4-MD5 (avec RSA 1024 bits pour la partie asymétrique), l'ouverture d'une session TLS en mode *full*, nécessite un temps de 2,0 secondes, et pour le mode *resume*, un temps de 0,5 seconde. Cependant, la session étant établie de carte à carte, les temps à considérer doivent être doublés. Autrement dit, le paiement vers le SAM nécessite :

- 4,0 s pour une session *full*, cas majoritairement rencontré dans ce contexte.
- 1,0 s pour une session *resume*.

Ainsi, même dans le cas de sessions *full*, un temps de transaction raisonnable est observé, ce qui valide le caractère réaliste d'une telle plateforme.

7.6 Conclusion

Le modèle d'identité reposant sur la carte EAP-TLS peut être décliné à l'envi selon les applications envisagées. Le contexte du prépaiement s'accorde naturellement au concept de carte personnelle stockant des conteneurs d'identité TLS. Bien que la multiplicité des identités ne soit pas ici mise à profit, notre modèle d'identité reste entièrement compatible avec des applications éloignées du principe de l'authentification Web, ce qui résulte du caractère convergent de ce modèle.

L'exemple du prépaiement a permis de démontrer que la carte EAP-TLS s'inscrit aussi bien dans la mouvance de la dématérialisation des biens, qui implique la mise en place de nouvelles mesures de sécurisation et d'authentification, mais aussi, de manière plus générale, dans toute architecture de type M2M.

Les performances mesurées sur la plateforme décrite dans ce chapitre montrent que l'authentification mutuelle de carte à carte peut être aisément déployée dans tout contexte où un temps de plusieurs secondes est considéré comme acceptable pour une session TLS en mode *full*. Ainsi, tous les cas d'utilisation relatifs à l'authentification dans l'Internet des Objets, notamment, peuvent être couverts par une architecture semblable à celle que nous avons décrite – qui a également permis de montrer qu'une carte peut être pourvue d'une identité jouant à la fois le rôle de client et de serveur. Cet état de fait souligne l'ambiguïté de la notion d'identité y compris pour des entités non vivantes – au sens biologique du terme. Ainsi, plutôt que de considérer, comme nous l'avons fait, qu'un élément possède une seule identité dotée de deux rôles très différents, il eût été possible de considérer la juxtaposition de deux identités, chacune étant dévouée à un seul rôle bien défini.

Par conséquent, le modèle d'identité que nous avons présenté reste entièrement valable dans toute architecture M2M. Si le seul principe de l'identité *racine* assure la compatibilité du modèle, la multiplicité des identités trouve également du sens pour tous les cas où une entité donnée assure plusieurs fonctions. La notion d'identité est alors non plus associée au pseudonyme d'un individu, mais à une fonction, ou un rôle, d'une machine ou d'un objet.

Chapitre 8 : Une architecture pour la distribution sécurisée de clés pour serrures RFID

8.1 Introduction

Le développement de la mobilité et des Smartphones, ainsi que leur succès, ont considérablement modifié le rapport des utilisateurs aux services. Qu'il s'agisse des services originellement accessibles par une interface Web, mais aussi de services nécessitant une présence ou un objet physiques, lesquels tendent à disparaître avec la tendance à la dématérialisation dont le chapitre précédent offre une illustration concrète.

Les spécificités technologiques des Smartphones offrent de nouvelles opportunités dans la construction d'un accès rapide et sécurisé à des services extrêmement variés. Ainsi, le système d'exploitation Android est compatible, dans ses versions les plus récentes, avec le standard NFC (*Near Field Communication*). Il est alors possible de mettre à profit le mode lecteur NFC d'un Smartphone Android afin d'établir une communication avec une carte externe.

Usuellement, les cartes RFID (*Radio Frequency IDentification*) ont pour objet d'autoriser l'accès à un individu à un lieu donné après lecture du contenu de sa carte – autrement dit de sa clé. Des lieux tels que les hôtels utilisent fréquemment un tel système pour le (dé)vérouillage des portes des clients. Actuellement, les architectures de distribution de clés reposent sur des environnements dédiés : la clé, créée dans le système d'information de l'organisme, peut être chargée dans la carte à partir d'équipements de confiance (*Card Encoder*) devant réaliser un lien physique avec la carte, ce qui constitue une contrainte.

Ainsi, en considérant un contexte dans lequel nous disposerions d'une telle carte RFID, il est possible de supprimer cette contrainte en déléguant au possesseur de la carte la capacité à y charger lui-même la clé correspondante. Pour cela, l'établissement d'un tunnel sécurisé après authentification du possesseur de la carte est indispensable. La carte EAP-TLS, et le modèle d'identité afférent, s'inscrivent naturellement dans la conception d'une architecture répondant à cette ambition, mais doivent être adaptés à l'environnement des cartes sans contact.

Dans ce chapitre, nous décrivons une architecture de distribution de clés intégrant d'une part un Smartphone Android, qui permet d'assurer à la fois la connectivité TCP/IP, l'interface utilisateur et la lecture de cartes sans contact par la technologie NFC, et d'autre part une carte RFID à interface duale, personnelle, compatible avec notre modèle d'identité, et établissant une authentification mutuelle et un tunnel sécurisé TLS avec un serveur de clés.

8.2 La technologie NFC

8.2.1 Généralités

La technologie NFC[99] est une interface radio fonctionnant à la fréquence 13,56 MHz. Il s'agit d'une extension de la norme ISO/IEC 14443[100] qui définissait un standard pour les cartes de proximité utilisées pour l'identification (RFID), en visant ainsi à permettre à des équipements mobiles de se comporter comme des composants RFID. Elle reprend notamment de cette norme les deux types de cartes (typeA et typeB) différenciées, entre autres, selon la méthode de modulation et le schéma de codage des données. Selon le type de codage mis en œuvre, le débit obtenu peut être de 106, 212, 424 ou même 848 Kbits/s, bien que ce dernier chiffre soit incompatible avec la norme ISO/IEC 18092[101], décrivant l'interface et la communication entre deux périphériques NFC, et également incluse dans la définition de la technologie NFC.

Trois modes d'utilisation de cette technologie ont été définis :

- Mode *peer to peer* : échange de données entre deux périphériques NFC.
- Mode *card emulation* : Emulation, par un périphérique NFC, du comportement d'une carte sans contact.
- Mode *reader* : lecture de tags NFC, alimentation et communication avec un périphérique NFC fonctionnant en mode *card emulation*.

En pratique, la technologie NFC est embarquée dans de petits composants électroniques à faible consommation de puissance (moins de 10 mW), alimentés par induction électromagnétique. Les distances admissibles pour établir une communication sont très faibles (moins de 10 cm). Les composants RFID sont divisés en deux catégories :

- Les composants de moins de 1mm² conçus avec une logique câblée.
- Les microcontrôleurs sécurisés, d'une superficie d'environ 25 mm², intégrant du CPU, de la RAM, de la mémoire non volatile, et des accélérateurs cryptographiques.

Une illustration de la première catégorie de composants est la carte Mifare 1K [102], comportant 1Ko de mémoire et qui est une implémentation propriétaire de la norme ISO 14443 TypeA. Ces cartes faisant partie de l'architecture que nous avons conçue, nous en donnons une description détaillée à la section 8.3.2.

La seconde catégorie trouve un exemple dans les passeports électroniques, normalisés par les standards ICAO[103], intégrant la RFID. Les composants peuvent être aussi bien du typeA que du typeB. Les informations sensibles telles que les données biométriques des individus sont stockées de manière chiffrée, par des algorithmes classiques tels que 3DES, RSA ou Diffie-Hellman sur courbe elliptique (ECDH).

8.2.2 Support de la technologie NFC par Android

Le système d'exploitation Android[88][104] a déjà été présenté au cours du chapitre 5. Depuis la version 2.3 d'Android, également nommée Gingerbread, le support d'API logicielles NFC a été ajouté, permettant la construction d'un cadre abstrait au-dessus d'un composant matériel qui constitue l'adaptateur NFC. Gingerbread supporte le mode *peer to peer* et le mode *reader*. Bien que le mode *card emulation* puisse être fourni par la partie *hardware*, il n'est pas encore pris en compte par Android.

La classe *NfcManager* énumère l'ensemble des adaptateurs NFC disponibles sur le Smartphone Android. Habituellement, un seul composant de ce type est présent. Par conséquent, la méthode statique *getDefaultAdapter()* renvoie une instance de la classe *NfcAdapter* qui représente cet adaptateur.

```
private void resolveIntent (Intent intent)
throws IllegalArgumentException, IllegalAccessException  {

String action = intent.getAction();
 if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action))
 { Tag tag = intent.getParcelableExtra
  IsoDep TagI = IsoDep.get(tag) ; if (TagI== null) return ;

  try { TagI.connect();}
  catch (IOException e) {return;}

  byte request[]= {(byte)0x00,(byte)0xA4,(byte)0x04,(byte)0x00,
                   (byte)0x07,(byte)0xA0,(byte)0x00,(byte)0x00,
                   (byte)0x00,(byte)0x30, (byte)0x00,(byte)0x01};

// Send ISO7816 Request, Receive Response
try { byte[] response= TagI.transceive(request);
catch (IOException e) {return;}

   try {TagI.close();}
   catch (IOException e) {return;}

  if ( (response[0]==(byte)0x90) && (response[1]== (byte)0x00))
 { tv.setText("OK");setContentView(tv);   }
 return;
}
else  return;  } }
```

Figure 36 : Détection d'un composant NFC et échange de données

Lorsqu'une application est enregistrée auprès d'un événement de découverte RFID, tel que TAG_DISCOVERED, elle obtient un objet *Intent* à partir duquel est extrait un objet *Tag*, lequel est converti en un objet RFID permettant d'exécuter des commandes de lecture ou d'écriture de manière appropriée. Ainsi, des méthodes statiques telles que *MifareClassic.get(tag)* ou *IsoDep.get(tag)* renvoient respectivement une instance de type Mifare ou ISO 14443. La figure 36 donne un exemple de code réalisant la détection d'un tag RFID ISO 14443 et exécutant en cas de succès un ensemble de requêtes / réponses ISO 7816 échangées entre le tag et l'application.

8.3 Les cartes RFID à interface duale

Une carte RFID à interface duale est, de la même manière qu'une carte à puce classique, un microcontrôleur sécurisé doté de contre-mesures physiques et logiques. Cependant, elle a la particularité de supporter à la fois une interface Mifare et une interface ISO 14443. La carte que nous avons utilisée pour notre plateforme, répondant à ces critères, est une JCOP41, dont nous détaillons à présent les caractéristiques techniques.

8.3.1 Le système d'exploitation JCOP

La plupart des systèmes d'exploitation pour microcontrôleurs sécurisés implémentent une machine virtuelle Java. Parmi ceux-ci, le système d'exploitation JCOP (*Java Card Open Platform*) a été créé par le laboratoire de recherche d'IBM de Zurich[105]. D'après [106] et [107], la puce matérielle est composée d'un processeur, de composants de sécurité, de ports d'entrée/sortie, de mémoires volatile et non volatile (4608 octets de RAM, 160 Ko de ROM, 72 Ko d'E²PROM), d'un générateur de nombres aléatoires, de co-processeurs cryptographiques exécutant des algorithmes classiques tels que 3DES, AES, et RSA. Le composant embarque également une interface sans contact ISO 14443.

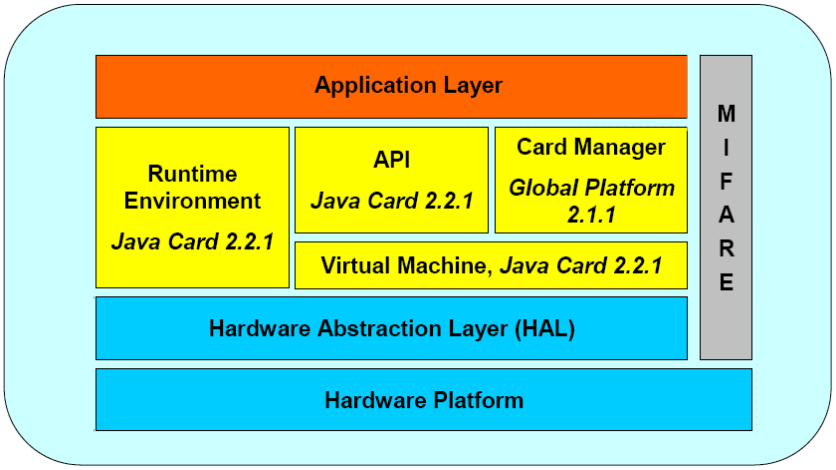

Figure 37 : Le système d'exploitation JCOP41

Le système d'exploitation JCOP41 embarque également une JVM, dont les différents composants, déjà évoqués pour la plupart dans la section 3.2.2, sont représentés sur la figure 37. Cette dernière figure fait également apparaître l'existence d'une API Mifare, bien que réduite à une seule méthode. JCOP41 inclut en effet l'émulation d'un composant Mifare 1K, qui fournit les mêmes fonctionnalités matérielles et logicielles qu'un composant natif.

8.3.2 La carte Mifare Classic 1K

Un composant Mifare Classic 1K [102] implémente une version propriétaire de l'interface sans contact ISO14443 TypeA. Il comprend une mémoire EEPROM de 1 Ko, organisée en 16 secteurs de 4 blocs, chacun étant composé de 16 octets. La figure 38 illustre la structure de la mémoire d'un tel composant.

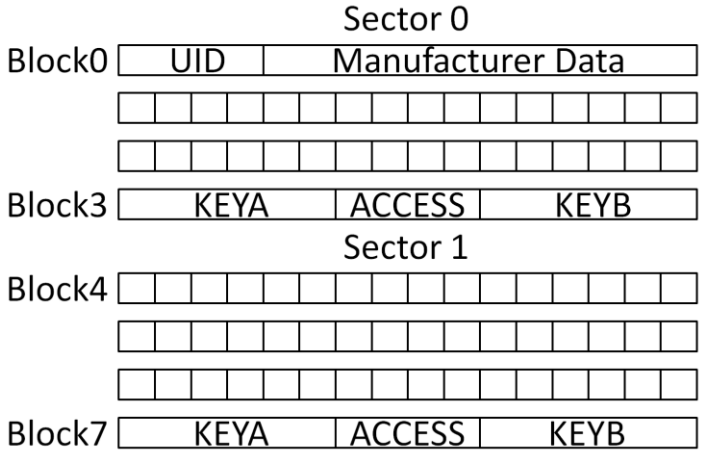

Figure 38 : Structure de la mémoire Mifare Classic

Chaque bloc est identifié par un index numéroté de 0 à 63. Le quatrième bloc de chaque secteur stocke deux clés de 48 bits (*KeyA* et *KeyB*). Les quatre octets restants définissent les conditions d'accès à la mémoire du secteur, à l'exception de ce quatrième bloc dont les conditions d'accès sont spécifiques. Les opérations de lecture et d'écriture peuvent ainsi être soit libres, soit contrôlées par des procédures d'authentification faisant intervenir l'une des clés A ou B. En revanche, la clé A n'est jamais accessible directement par un lecteur. La clé B peut l'être dans certaines configurations, mais ne pourra pas dans ce cas faire l'objet d'une procédure d'authentification. Cet espace est alors simplement utilisé pour le stockage de données. Enfin, le premier bloc de la mémoire, nommé *Manufacturer Block*, contient quatre octets d'identifiant unique (UID, *Unique Identifier*), un octet de vérification de l'UID nommé BCC (*Bit Count Check*) et obtenu par un XOR des octets de l'UID, et enfin onze octets de données du constructeur.

L'authentification repose sur un protocole basé sur le standard ISO 9798-2 (*Mutual Three-pass Authentication*)[108] utilisant le chiffrement à flot Crypto-1, et deux nombre aléatoires respectivement générés par la carte Mifare et par le lecteur. Aucun échange de clé n'a lieu, cette dernière étant préalablement connue de chacune des deux entités. La procédure d'authentification permet à la carte et au lecteur de se prouver mutuellement cette connaissance de la clé, sans l'envoyer. Le scénario d'authentification, illustré sur la figure 39, se déroule de la manière suivante :

- Le lecteur choisit un secteur devant faire l'objet d'un accès, puis choisit la clé A ou la clé B qui sera utilisée comme clé de chiffrement.

- La carte lit la clé correspondante ainsi que les conditions d'accès du secteur concerné. Elle génère et envoie un nombre aléatoire *r1*.

- Le lecteur calcule une réponse en chiffrant avec un algorithme E le nombre *r1* avec un autre nombre aléatoire *r2* qu'il a lui-même généré. A partir de cet instant, tous les échanges successifs sont chiffrés.

- La carte vérifie la réponse et notamment la présence de *r1*, qui certifie la connaissance de la clé par le lecteur. Afin de certifier au lecteur qu'elle partage cette même clé, elle chiffre et renvoie *r2*.

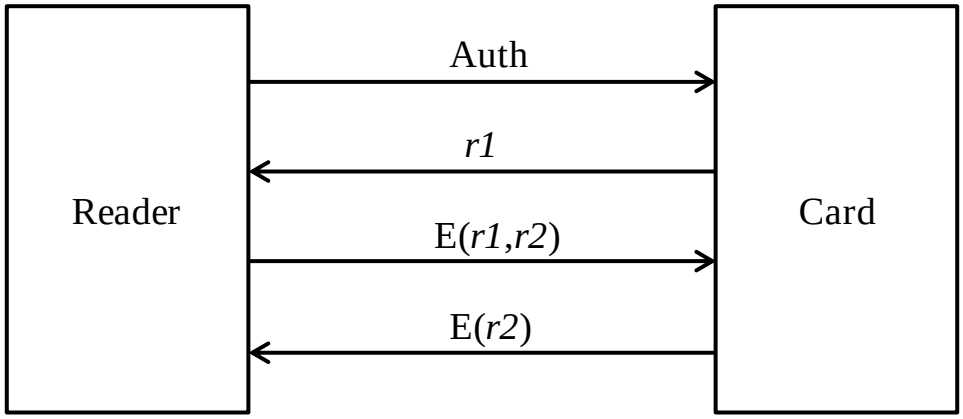

Figure 39 : Authentification Mifare Classic

L'algorithme de chiffrement symétrique utilisé, Crypto-1, noté E dans la figure précédente, est un algorithme propriétaire créé par NXP Semiconductors. Le cœur de cet algorithme est constitué d'un LFSR (Linear Feedback Shift Register) de 48 bits et d'une fonction filtre. La cryptanalyse de cet algorithme, et plus généralement de son utilisation pour l'authentification dans la carte Mifare Classic [109] [110], a montré qu'il était possible de retrouver une clé secrète à partir d'une ou deux tentatives d'authentification en moins d'une seconde, le tout sans calculs préalables, mais a aussi montré qu'un espion écoutant l'ensemble d'une communication entre une carte et un lecteur était capable d'en déduire toutes les clés secrètes utilisées au cours de l'échange, ce qui rend aisé le clonage de cartes.

Néanmoins, les cartes Mifare Classic restent très utilisées dans des services tels que les transports en commun, le paiement dans les cantines ou encore l'autorisation d'accès à des lieux tels que des chambres d'hôtel à partir de clés personnalisées.

8.3.3 L'émulation Mifare

Il est souvent utile pour les systèmes d'exploitation de microcontrôleurs sécurisés de pouvoir écrire et lire des données dans des blocs de mémoire Mifare. L'émulation Mifare implémentée par le système JCOP41 permet ainsi à des applications Javacard de pouvoir interagir, via une API dédiée, avec la mémoire Mifare. Pour des raisons de sécurité, la connaissance, par l'application, des clés

secrètes A ou B n'est pas requise. Afin de remplacer la connaissance de ces valeurs, un paramètre appelé *Mifare Password* (MP, [111]), d'une longueur de 8 octets, est calculé pour chaque secteur selon la formule suivante :

$$MP = DES_{DKEY1} \circ DES^{-1}_{DKEY2} \circ DES_{DKEY1}(IV)$$

Le paramètre IV est une valeur nulle de 8 octets, DKEY1 et DKEY2 sont deux clés DES de 56 bits respectivement construites à partir des clés A et B, de longueur 48 bits. La connaissance de la valeur MP d'un secteur donné permet à une application de réaliser des accès à la mémoire Mifare correspondante.

Le système d'exploitation JCOP possède une API Mifare réduite à une seule méthode, dont le prototype est :

short JZSystem.readWriteMifare(short mode, byte[] data, short offset, short mifareBlock)

Cette méthode renvoie une valeur nulle en cas de succès, et ses paramètres sont définis ainsi :

- *mode* : accès en lecture ou en écriture à un bloc particulier
- *data* : buffer contenant le MP relatif au secteur contenant le bloc faisant l'objet d'un accès, ainsi qu'un espace additionnel de 16 octets pour la lecture ou l'écriture de données.
- *offset* : numéro de l'octet à partir duquel se situe le premier octet du MP dans le buffer data précédent
- *mifareBlock* : numéro du bloc devant faire l'objet d'un accès en lecture ou en écriture.

8.4 Proposition d'une architecture pour la distribution de clés

8.4.1 Les éléments constitutifs de l'architecture

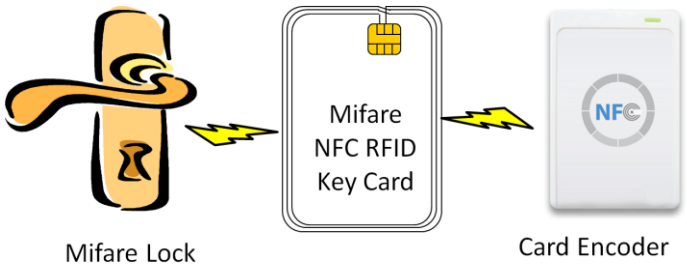

Figure 40 : Systèmes classiques pour l'ouverture de serrures via le NFC

Comme nous l'avons souligné dans l'introduction, les systèmes classiques pour l'ouverture de serrures de porte reposent sur la technologie NFC, avec l'utilisation de cartes Mifare venues remplacer les cartes à bande magnétique. La figure 40 représente une vue simplifiée de l'architecture relative à ces systèmes, où seuls trois éléments entrent en jeu : la carte Mifare, mise à jour par le *Card Encoder*, peut ensuite ouvrir la serrure qui lui est dédiée. Afin de prendre appui sur des éléments concrets, nous considérerons par la suite le cas de l'ouverture de chambres d'hôtel, sans que cela constitue la limite de notre solution.

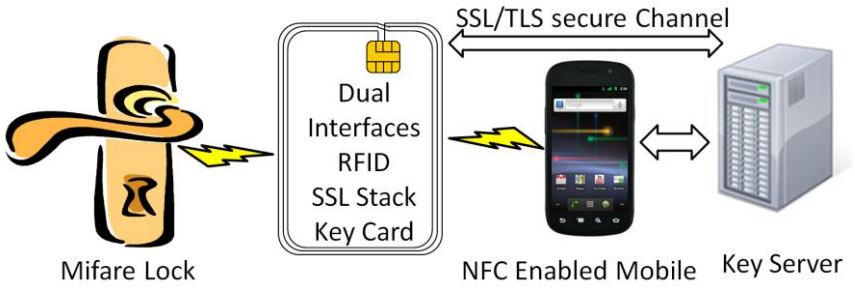

Figure 41 : Proposition d'architecture pour la distribution de clés comme service mobile

Dans notre approche, l'utilisateur, possesseur d'une carte à interface duale, éventuellement indépendamment de son lien avec l'hôtel, est au centre de l'architecture. Il peut ainsi charger lui-même sa clé depuis un serveur dédié par l'intermédiaire d'un tunnel sécurisé TLS. La figure 41 représente, sous un angle comparable à la figure précédente, une telle architecture. Celle-ci remplace la carte Mifare par une carte à interface duale, le *Card Encoder* par un serveur de clés, et ajoute un quatrième élément, un Smartphone Android, qui permet d'adapter le principe du service de distribution de clés à la mobilité. Le choix du système d'exploitation Android a été motivé par la possibilité laissée aux utilisateurs de télécharger des applications non nécessairement disponibles depuis l'*Android Market*, via des sources de confiance – les applications devant toutefois être signées.

La carte à interface duale que possède l'utilisateur embarque une application notée RA (*Radio Application*) contenant, d'une part, la technologie EAP-TLS que nous utilisons habituellement, et d'autre part, l'interaction avec la partie Mifare afin d'y écrire les clés téléchargées depuis le serveur dédié. Le terminal mobile, quant à lui, embarque une application notée MA (*Mobile Application*) dont le rôle est essentiellement celui de logiciel proxy tel que décrit dans le chapitre 3, bien qu'adapté dans le cas présent pour la technologie NFC.

La MA est construite autour d'une classe nommée tls_tandem, dont le constructeur possède le prototype suivant :

public tls_tandem(int mode, ReaderSC reader, String aid, String pin, String identity)

Ce constructeur met en place l'environnement nécessaire à la réalisation des opérations TLS avec une carte RFID externe, les paramètres ayant la signification suivante :

- *mode* : rôle de la carte RFID (client ou serveur TLS)

- *reader* : représentation abstraite du lecteur RFID, reposant sur le modèle NFC Android. Cet objet fournit par ailleurs le support pour les opérations d'entrées / sorties.
- *aid* : *Application Identifier* permettant de sélectionner l'application RA stockée dans la RFID.
- *pin* : code PIN nécessaire pour l'activation de la carte RFID.
- *identity* : chaîne de caractères identifiant un ensemble de paramètres relatifs à un conteneur d'identité TLS tel que décrit au chapitre 3. S'il existe, ce conteneur est alors sélectionné comme identité active pour l'établissement de la session TLS mutuelle.

8.4.2 Scénario d'obtention d'une clé

Figure 42 : Scénario d'obtention d'une clé

Le scénario d'obtention d'une clé par un utilisateur, dont une vison synthétique est fournie dans la figure 42, est amorcé par la détection d'une carte à interface duale par le Smartphone. L'utilisateur est alors amené à sélectionner la MA. La pile TLS embarquée dans la carte RFID est activée par l'intermédiaire de la RA. La MA ouvre une *socket* TCP avec le serveur de clés distant et supervise la phase 1 (*Handshake*) de la session TLS mise en place entre la carte de l'utilisateur, dont l'identité a été sélectionnée, et le serveur. En cas de succès de l'authentification mutuelle, les key-blocks calculés par la carte sont transmis au terminal mobile qui gère entièrement la phase 2 de la session TLS, i.e. le chiffrement et déchiffrement des données applicatives transmises.

De manière plus précise, la MA construit une requête HTTP transmise au serveur de clés au-dessus de la session TLS. La figure 43 montre un extrait du code résumant le déroulement de la MA. Le fichier de clés est ici identifié par l'URL : https://www.server.com/getkey.php. Il est impossible d'accéder à ce fichier sans authentification mutuelle préalable, ce qui permet au serveur de conserver l'identité de la carte client par l'intermédiaire de la variable PHP $_SERVER['SSL_CLIENT_CERT'].

```
private void resolveIntent (Intent intent) throws IllegalArgumentException, IllegalAccessException {
String action = intent.getAction();

if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action))
{ Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
TagI = IsoDep.get(tag)   ;
if (TagI==null) { Box("Error","Non ISO Tag"); return;}

ReaderNFC  myReader= new ReaderNFC(TagI);
tls-tandem myTlsTandem=
new tls-tandem(tls_tandem.CLIENT,myReader,
"A0000000300002FFFFFFFF8931323800","0000","client");

https reg = new https(myTlsTandem);
if (!reg.HTTPS(="www.server.com","/tandem/secure/getkey.php"))
Box("Error","HTTPS Error!!!");

String apdu1 = "00D0 8001 80 " + reg.body.substring(0,128);
String apdu2 = "00D0 8002 80 " + reg.body.substring(128,256);
String apdu3 = "00D0 8002 80 " + reg.body.substring(256,384);
String apdu4 = "00D0 8003 80 " + reg.body.substring(384,512);

myTlsTandem.SC_reader_socket.Echo_TAPDU(apdu1);
myTlsTandem.SC_reader_socket.Echo_TAPDU(apdu2);
myTlsTandem.SC_reader_socket.Echo_TAPDU(apdu3);
myTlsTandem.SC_reader_socket.Echo_TAPDU(apdu4);
myTlsTandem.Close(null); }

else { Box("Error","No card"); return; }
}

public class https {
    tls-tandem tandem = null ;
    record-layer RecordLayer=null;
     public String body=null;

    public https(tls-tandem mytandem) {
    tandem =mytandem;}

    public boolean HTTPS(String host, String myfile){
        boolean done;
        if (tandem == null) return false;
        RecordLayer = tandem.OpenSession(host,(short)443);
        if (RecordLayer == null) return false;
        done= http(myfile);
        RecordLayer.Close();
        return done ;
    }
}
```

Figure 43 : Structure du code de l'Application Mobile

De manière similaire à la procédure mise en place pour le chargement sécurisé des conteneurs d'identité TLS dans une carte, décrite à la section 4.3.4, le fichier getkey.php récupère la clé publique de la carte de l'utilisateur (une clé RSA, dans notre cas) via des appels aux fonctions de *OpenSSL*, afin de créer un conteneur de clé, constitué de deux parties :

- D'une part, la clé devant être stockée dans la mémoire Mifare, chiffrée avec la clé publique de la carte, selon le standard PKCS #1 dans notre cas.

- Une signature PKCS#1 de la partie précédente faisant usage d'une clé privée associée à un certificat reconnu comme étant de confiance par la RA, qui en possède alors la clé publique associée.

Le conteneur ainsi formé est renvoyé dans une représentation ASCII dans le corps de la réponse HTTP, laquelle est récupérée par le terminal mobile. Ce dernier envoie le conteneur vers la carte de l'utilisateur via le lien NFC. La figure 43 fait apparaître clairement la construction des 4 APDUs à partir du corps de la réponse HTTP et l'envoi de ces dernières à la carte RFID (appels à la fonction *Echo_TAPDU()*). Cette dernière vérifie alors la signature de conteneur puis déchiffre la valeur de la clé Mifare, avant de la stocker dans la mémoire dédiée en utilisant l'API et le *Mifare Password* décrits dans la section 8.3.3. L'interface Mifare de la carte peut alors être mise à profit pour l'ouverture de la serrure attribuée à l'utilisateur. La figure 44 illustre la synthèse des différentes étapes mises en œuvre pour l'obtention d'une clé.

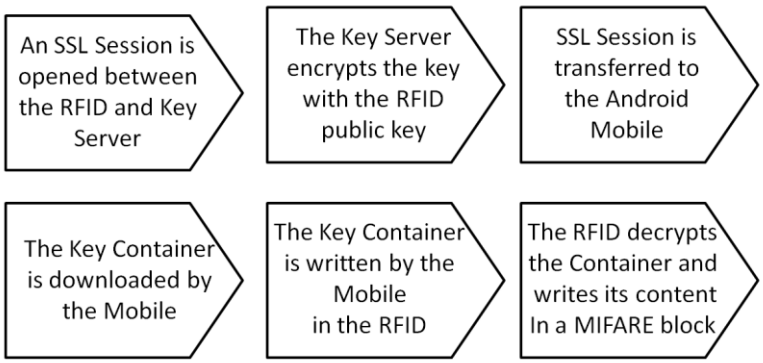

Figure 44 : Procédure de chargement d'une clé jusqu'à son écriture dans un bloc Mifare

Dans le cadre présenté ici, l'identité de la carte client est unique. Néanmoins, il est possible de concevoir un contexte plus étendu pour cette architecture, dans lequel il serait intéressant de posséder plusieurs identités. Par exemple, si plusieurs organisations ou entreprises s'accordent à distribuer leur clé sur une même carte, chaque identité pourra permettre à l'utilisateur d'obtenir la clé correspondant à l'un de ces organismes. Les identités stockées sur la carte de l'utilisateur ne sont donc pas des pseudonymes, mais différents représentants du même utilisateur auprès de chaque organisation. La multiplicité des identités préconisée par notre modèle trouve donc naturellement sa place au cœur de cette application.

8.4.3 Performances

Les performances mesurées avec la carte JCOP41 sont très éloignées de celles obtenues jusqu'à présent avec d'autres composants. D'ordinaire, une session TLS en mode *full* établie avec un serveur se monte à, au plus, cinq ou six secondes. Ici, le temps mesuré est environ le double. Pourtant, les performances cryptographiques de cette carte sont très raisonnables, comme le montre le tableau 9, notamment si l'on rapporte ces temps à ceux du tableau 2.

IO /byte	MD5 /block 64 B	SHA1 /block 64 B	RSA 1024 bits PUB	RSA 1024 bits PRIV
0,125	2,0	4,0	26,0	120,0

Tableau 9 : Performances cryptographiques d'une carte JCOP41

Une session TLS en mode *full* utilisant la *ciphersuite* habituelle (cf. chapitre 3) demande 11,7s. D'après le tableau 9, l'ensemble des calculs cryptographiques réalisés au cours de cette session consomme environ 1,6s, et l'échange de 2500 octets coûte 0,3s. Autrement dit, l'essentiel du temps nécessaire à l'établissement de la session TLS est consommé par l'exécution du code Javacard, qui prend à elle seule 9,8s.

Dans le mode *resume*, l'établissement d'une session demande 2,6s, dont 0,5s dédiées à la cryptographie et l'échange de données, et 2,1s d'exécution du code.

Dans tous les cas, c'est donc la machine virtuelle qui consomme la majorité du temps de calcul, ce qui n'est pas représentatif du comportement général des Javacards, puisque ce sont parfois les calculs cryptographiques qui représentent la majorité de ce temps d'exécution (cf. section 3.6.1, Javacard en session *full*).

Quoiqu'il en soit, il est certain que ces performances sont, en soi, trop peu concluantes pour être propices à un déploiement effectif de la solution. Néanmoins, les mesures réalisées avec d'autres types de cartes offraient des performances beaucoup plus intéressantes, comme l'attestent les résultats présentés dans les chapitres 3, 6 et 7 de ce rapport. Par conséquent, la viabilité de la solution repose ici sur le choix d'une carte à interface duale beaucoup plus performante que la JCOP41 dans l'exécution du code Javacard.

8.5 Conclusion

L'architecture que nous avons présentée dans ce chapitre, qui a fait l'objet d'un dépôt de brevet, permet d'adapter un système classique de gestion de clés aux possibilités offertes par les avancées menées dans les terminaux mobiles, tout en assurant la compatibilité avec les anciens systèmes. Cette solution, dont le contexte a été volontairement limité à l'exemple concret de serrures de chambres d'hôtel, peut s'appliquer en réalité à toute situation exigeant une autorisation pour l'accès à un lieu pour un temps limité. Par ailleurs, l'application mobile propose un service générique. En ce sens, il est possible d'en prévoir le déploiement d'une version unique, compatible avec toutes

les cartes à interface duale fournies par les différents organismes associés à un même cas d'usage – par exemple les hôtels.

La faible sécurité offerte par les cartes Mifare, à l'origine entièrement impersonnelles, est en partie compensée par l'adjonction d'une interface embarquant le protocole TLS, qui permet d'affecter une identité à la carte, selon le modèle que nous avons défini. Néanmoins, il subsiste avec ces cartes une menace réelle résidant dans l'authentification entre l'interface Mifare et la serrure électronique. Des cartes Mifare plus élaborées que la version Classic mettent en place un algorithme de chiffrement nettement meilleur que Crypto-1, par exemple 3DES ou AES, et pourraient à cette fin être efficacement déployées, dans une version à interface duale, au cœur d'une telle architecture.

Une autre approche consiste à tirer profit des possibilités des Javacard. Cette technologie permet en effet de partager des objets entre plusieurs applications. Il serait alors possible de remplacer le principe de l'interface duale par la cohabitation de deux applications Javacard, l'une pour le chargement de clés par l'intermédiaire d'un tunnel sécurisé TLS, et l'autre pour l'interaction avec la serrure.

Dans tous les cas, les opportunités d'amélioration des systèmes existant sont rendues possibles et aisées à mettre en place du fait de l'existence d'un modèle d'identité dont le caractère convergent permet de l'insérer au cœur de technologies à l'origine dépourvues de tout lien avec la notion d'identité. C'est pourtant ce lien qui permet d'assurer une flexibilité à la technologie, par la délégation de l'accès à certains services aux utilisateurs finaux, tout en permettant de consolider la sécurité des échanges via une authentification forte et fondamentalement associée à l'identité de l'utilisateur.

Conclusion générale

L'objectif de cette thèse était d'élaborer un modèle d'identité tirant parti des avantages fournis par un environnement de confiance afin d'obtenir une authentification forte et personnalisée, tout en s'adaptant aisément aux technologies émergentes et convergentes.

Après avoir, dans une première partie, exploré les composantes de l'identité numérique dans les réseaux à travers, d'une part, l'association entre la représentation d'une identité et la preuve de cette dernière – c'est-à-dire l'authentification – et d'autre part, les différents moyens mis en œuvre pour lutter contre la prolifération des identifiants des utilisateurs – c'est-à-dire la fédération d'identité – nous en avons tiré les caractéristiques qui nous semblent être indispensables pour la conception d'un modèle d'identité numérique, à savoir : l'utilisation de standards reconnus et déployés, la recherche d'une authentification forte obtenue par l'alliance de protocoles à la sécurité éprouvée et d'un environnement de confiance dédié, la mise en place d'un format d'identité simple mais fortement associé avec la procédure d'authentification choisie, et enfin, l'assurance de l'adaptation – voire, d'une participation active – du modèle à la convergence.

Nous avons alors décrit, dans le chapitre 3 qui clôt cette première partie, les choix technologiques qui ont fondé notre modèle d'identité, notamment le principe d'une authentification mutuelle via le protocole TLS, lequel se trouve entièrement embarqué dans un microcontrôleur sécurisé, dont les contre-mesures permettent de fournir un environnement de confiance pour le stockage d'éléments cryptographiques sensibles. Nous y présentons également le format de l'identité afférent à cette technologie, ainsi que l'architecture basique nécessaire au fonctionnement de cette technologie.

La seconde partie de cette thèse est ainsi consacrée à la construction du modèle d'identité proprement dit, dans sa version la plus générale, et entièrement adaptée aux usages du Web, qui constituait la cible première de ce modèle. Le principe d'une carte personnelle, et personnalisable, la liberté laissée à l'utilisateur d'y charger des identités associées à des pseudonymes, sont les éléments qui nous ont semblé les plus importants à mettre en place. Nous décrivons l'implémentation d'un serveur d'identité réalisant l'ensemble de ces fonctionnalités, la sécurisation de la génération et du chargement des identités, ainsi que le cycle de vie global du composant, depuis sa création jusqu'à sa disparition.

Sur les bases ainsi posées, nous décrivons alors une architecture entièrement fonctionnelle associant notre modèle d'identité à la fédération d'identité, laquelle se pose ici en moteur pour le déploiement de notre solution sur le Web. La preuve de concept ainsi établie permet à tout utilisateur de s'authentifier auprès d'un serveur Web quelconque sur la base des informations qu'il aura lui-même choisies de stocker dans son composant. Nous illustrons aussi la possibilité de réaliser cette opération à partir d'un terminal mobile – les spécificités techniques liées à ces derniers ayant été prises en compte dans l'implémentation du serveur d'identité et du serveur de fédération d'identité.

Enfin, dans une dernière partie, nous évoquons différentes perspectives de déploiement de notre modèle, dont les contextes peuvent mettre à profit tout ou partie de ce dernier. Nous nous sommes efforcés de rechercher des cas extrêmement éloignés les uns des autres afin d'illustrer au mieux la capacité de notre solution à s'intégrer, voire à remplacer, les technologies préexistantes dans chacun de ces domaines.

Le premier cas exposé est celui d'une architecture pour l'authentification sécurisée dans le Cloud Computing, aussi bien pour l'accès à des ressources protégées d'un réseau interne que pour la génération de jetons dans le cadre de la fédération d'identité. Cette architecture met à contribution l'ensemble des fonctionnalités de notre modèle pour le côté client, et permet de faire cohabiter des identités d'organismes différents pour le côté serveur. Elle met également en avant la possibilité de réaliser une authentification sécurisée de composant à composant, qui peut être avantageusement incluse dans la mouvance du M2M.

Le second cas s'oriente sur la dématérialisation d'un service de prépaiement géré, à l'heure actuelle, principalement par le format papier. L'architecture que nous proposons, qui permet de sécuriser la gestion de jetons de prépaiement tout en restant extrêmement fidèle aux principes des architectures en place, a la particularité d'intégrer un composant pour lequel deux identités coïncident – client et serveur, en l'occurrence. Cela permet d'illustrer un principe intéressant, qui stipule que l'identité numérique d'une machine peut, à l'image de l'identité numérique d'un individu, ne pas être unique. Si dans le second cas, les multiples identités permettent de conserver des pseudonymes dans l'interaction avec les différents services, dans le premier cas, les identités peuvent être réparties selon les fonctions, souvent non uniques, attribuées aux machines. Suivant ce raisonnement, notre proposition de modèle d'identité s'intègre alors parfaitement au cas de l'authentification dans le M2M.

Enfin, le dernier cas met en valeur les possibilités offertes par l'ajout de la notion d'identité pour la personnalisation de services tels que la distribution de clés. A partir d'un cadre où le seul outil d'obtention d'une clé permettant l'accès à un lieu pour une durée limitée est une carte impersonnelle du type Mifare, nous réorientons l'ensemble de l'architecture vers l'exploitation des possibilités offertes par les terminaux mobiles tels que les Smartphones, notamment leur compatibilité avec la technologie NFC. Nous mettons alors à contribution les cartes à interface duale afin de maintenir la compatibilité avec le standard Mifare, mais également afin d'introduire notre modèle d'identité au cœur de l'architecture de distribution de clés, qui permet la délégation du service à l'utilisateur tout en assurant la sécurisation du chargement des clés dans la carte. L'orientation vers des services centrés utilisateur est en effet en phase avec la mouvance actuelle perceptible via, notamment, la prépondérance des réseaux sociaux, et ne peut être réalisée que par la prise en compte d'un modèle d'identité assurant un lien personnalisé avec l'utilisateur.

Le modèle d'identité que nous avons présenté dans ce rapport n'est pas exempt de limitations, notamment dans la perspective d'un déploiement à grande échelle. La promotion d'un standard de sécurité tel que l'authentification mutuelle TLS ne peut avoir lieu sans une adaptation globale des relations des serveurs Web avec les utilisateurs. Bien que la fédération d'identité nous permette de

repousser le problème, elle ne le résout pas entièrement pour autant. Néanmoins, le problème du déploiement d'une telle solution relève davantage de problématiques liées au monde industriel que d'un travail de thèse. Par conséquent, nous ne nous étendons pas sur ces questionnements dans ce rapport. Néanmoins, les possibilités d'application de notre solution, en dehors de la relation d'utilisateur humain à serveur Web, alliées à son adaptation intrinsèque au principe de la convergence, sont suffisamment nombreuses pour illustrer les nombreuses perspectives dont elle peut faire l'objet, quitte à devenir un standard dans un contexte plus restreint que celui de la généralisation de l'authentification sécurisée sur le Web.

Annexe A : Les concepts de la sécurité

Face aux nombreuses faiblesses dont sont victimes les réseaux informatiques et les systèmes d'information, et dont l'exploitation peut être la cause de préjudices conséquents tels que l'usurpation d'identité ou l'espionnage industriel, le domaine de la sécurité informatique a forgé plusieurs concepts définissant notamment les différents services de sécurité ainsi que les typologies d'attaques portées sur les communications réseau. Ces définitions, qui sont l'objet de cette annexe, sont largement répandues dans le monde de la sécurité des réseaux, et permettent d'asseoir la spécificité des différents protocoles ou architectures élaborés, afin de les classifier selon les services de sécurité qu'ils proposent ou leur capacité de résistance face aux attaques.

A.1 Les services de la sécurité informatique

Les services de la sécurité informatique s'intéressent à la classification des garanties offertes par une application, un protocole, ou une architecture. Sont ainsi distingués, en restreignant les définitions à la sécurité des réseaux :

- **La confidentialité** : Il s'agit de la capacité à rendre un message incompréhensible, et donc non accessible, à tout attaquant ou espion potentiel. Seul l'auteur du message et ses destinataires sont capables d'avoir accès au contenu du message. La confidentialité est assurée par la cryptographie.

- **L'intégrité** : Il s'agit de la garantie qu'un message n'a pas été altéré au cours d'une communication, entre le moment où il a été émis et le moment où il a été reçu. L'utilisation des fonctions de hachage ou d'autres techniques (par exemple le bit de parité ou le contrôle de redondance cyclique, moins fiables que les fonctions de hachage) permet de fournir l'intégrité.

- **L'identification** : Il s'agit de la capacité à recueillir des informations sur l'identité de l'auteur d'un message, par exemple un identifiant, représenté par une chaîne de caractères. Contrairement à l'authentification, il n'est pas nécessaire de fournir une quelconque preuve de cette identité.

- **L'authentification** : Il s'agit de la capacité à vérifier l'identité de l'auteur d'un message. Contrairement au cas de l'identification, l'auteur doit ici non seulement fournir une information relative à son identité, mais également une preuve de cette identité. Cette preuve peut être de nature extrêmement variée, et est en général classifiée selon ce que l'utilisateur est (biométrie, etc.), ce que l'utilisateur sait (mot de passe, etc.), ou ce que l'utilisateur possède (certificat X509, OTP hardware, etc.). Le chapitre 1 de ce rapport de thèse explore en détail la question de l'authentification.

- **La non répudiation** : Il s'agit de la capacité à certifier que l'auteur d'un message ne peut plus prétendre ne pas l'avoir envoyé ou ne pas en être le véritable auteur. Cela implique la garantie de l'intégrité du message, sans laquelle il est possible à un tiers d'ajouter des modifications au cours de la communication. La non répudiation est fournie par la signature électronique, qui est une opération cryptographique asymétrique portée sur un condensât du message, opération pour laquelle l'auteur du message réalise un chiffrement à partir de sa clé privée, dont il est par définition l'unique possesseur.

- **L'horodatage** : Il s'agit de la capacité à dater précisément l'envoi d'un message, généralement à l'aide de l'heure Unix qui possède une précision à la seconde. L'horodatage peut par exemple être utilisé comme moyen de contrer les attaques par rejeu (cf. section A.2).

Ainsi, une communication fortement sécurisée doit mettre en place l'ensemble de ces services pour chacun des messages échangés. Néanmoins, cela n'est généralement pas le cas. Par exemple, un échange TLS avec authentification non mutuelle garantit bien la confidentialité, l'intégrité et l'horodatage de chaque message échangé, mais garantit uniquement les services d'identification, d'authentification et de non répudiation pour le serveur, et non pour le client.

A.2 Typologie des attaques réseau

Les attaques menées contre les réseaux peuvent être classées et différenciées selon leur nature afin de juger plus finement de la solidité des protocoles et architectures à l'aune de cette typologie. Notamment, ces attaques peuvent être répertoriées selon deux groupes complémentaires, à savoir les attaques actives, qui impliquent une participation de l'attaquant aux messages échangés, et les attaques passives, davantage relatives à l'espionnage des conversations.

Les **attaques passives** sont assez peu nombreuses, du point de vue de la classification des attaques, mais elles peuvent être extrêmement faciles à réaliser, notamment si le trafic n'est pas sécurisé. Deux types d'attaques peuvent être dégagés :

- **L'écoute passive du trafic** : Il s'agit simplement pour l'attaquant d'écouter l'ensemble des messages échangés entre deux entités. Pour que cette attaque ait un intérêt, il faut que le trafic ne soit pas chiffré. Toute information échangée en clair, par exemple un mot de passe envoyé lors d'une requête HTTP, est susceptible d'être récupérée par l'attaquant.

- **L'analyse de trafic** : Il s'agit pour l'attaquant de s'intéresser aux meta-données transmises dans le trafic afin de déduire des informations relatives à l'échange et aux entités, par exemple les paramètres de sécurité négociés. L'analyse de trafic peut également conduire, sur du trafic chiffré, à des attaques de type cryptanalyse, qui peuvent amener l'attaquant à obtenir des informations ou à décrypter entièrement le trafic.

Les **attaques actives** comportent un éventail nettement plus large d'attaques, dont la liste peut s'allonger selon l'imagination des attaquants. Ces attaques ont toutes en commun le fait

d'impliquer l'attaquant dans la communication par l'émission ou la modification de messages. Les principales attaques existantes demeurent les suivantes :

- **Le rejeu** : Le principe de cette attaque est d'enregistrer une série de messages échangés entre deux entités, typiquement un client, ici la victime, et un serveur, afin de les rejouer tels quels en vue d'obtenir par exemple un accès à des ressources protégées. Cette attaque fonctionne notamment sur une conversation cryptée si des contre-mesures n'ont pas été mises en place, lesquelles consistent généralement en l'échange de nombres aléatoires uniques ou l'horodatage.

- **Le Man in the Middle** : Cette attaque, décrite dans le chapitre 1 de ce rapport, consiste pour l'attaquant à se placer entre deux entités communicantes, en se faisant passer auprès de chacune des entités pour son interlocuteur légitime. L'attaquant intercepte alors tous les messages échangés et peut les modifier à loisir, avant de les réémettre vers leur véritable destinataire. Il s'agit d'une attaque très difficile à éliminer d'un point de vue théorique, lors de la conception d'un protocole par exemple, car elle repose sur la question de l'identité de son interlocuteur lors d'une procédure d'authentification. Ainsi, tous les protocoles à mots de passe sont par définition susceptibles d'être victimes d'une attaque Man in the Middle.

- **Le déni de service** : Il s'agit d'une attaque visant à épuiser les ressources réseau ou système d'une machine. Ces attaques peuvent cibler un défaut de conception des protocoles (attaques Land, Ping of Death, etc.) ou bien fonctionner par montée en charge du trafic (attaque Smurf par exemple). Cette version du déni de service trouve son prolongement dans les attaques de **déni de service distribué**, qui consistent à mobiliser une grande quantité de machines zombies (i.e. infectées par un malware discret) pour générer un volume très important de trafic contre une cible donnée.

- **La force brute** : Cette attaque vise à tester un ensemble de clés ou de mots de passe pour tenter d'accéder à une ressource protégée de manière illégitime. Il peut s'agir par exemple du test exhaustif de tous les mots de passe possibles pour un utilisateur donné. Cette attaque, mentionnée dans le chapitre 1 de ce rapport, est de complexité exponentielle en temps, et n'est donc efficace que sur des ensembles de données très restreints.

A ces attaques s'ajoutent toutes les méthodes de cryptanalyse visant à casser les systèmes cryptographiques conçus, notamment en trouvant des failles dans les algorithmes de chiffrement ou dans leur implémentation. Ces attaques, ainsi que l'ensemble des éléments fondamentaux relatifs à la cryptographie (chiffrement symétrique, asymétrique, fonctions de hachage, etc.) ne sont pas détaillés dans ce rapport.

Annexe B : Le standard ISO/IEC 7816

La définition des caractéristiques physiques, des interfaces et des protocoles de communication pour les cartes à puce à contacts est décrite dans le standard ISO/IEC 7816. Il est composé de 15 parties, dont les quatre premières constituent les bases fondamentales des formats des messages et du fonctionnement des cartes à puce. Cette annexe vise à décrire les grandes lignes de ce standard, et notamment le format des APDUs, qui permettront de donner quelques exemples de traces d'échanges de message obtenues au cours de ce travail de thèse. En aucun cas cette annexe ne prétend décrire exhaustivement le contenu des différentes parties du standard.

B.1 ISO 7816-1/2 : Caractéristiques physiques et contacts

Il y a peu à dire des deux premières parties du standard. L'ISO 7816-1 décrit les caractéristiques physiques relatives aux cartes à puce : limites d'exposition aux champs électromagnétiques, température ambiante, comportement en cas de pliage de la carte, etc. La deuxième partie, ISO 7816-2, donne quant à elle les dimensions, la position, et la fonction des huit contacts électriques des cartes à puce, répartis en deux colonnes de quatre contacts, nommés de C1 à C8 de haut en bas et de gauche à droite. Ces contacts possèdent les fonctions suivantes :

- C1 : **Vcc**. Alimentation de 3V ou 5V, qui permet la mise en fonctionnement du microprocesseur de la carte.
- C2 : **RST**. Remise à zéro de la carte.
- C3 : **CLK**. Signal d'horloge fourni au microprocesseur de la carte.
- C4 : **RFU**. Non utilisé.
- C5 : **GND**. La masse.
- C6 : **Vpp**. Tension pour la programmation de l'EEPROM des premières cartes à puce. Ce contact est actuellement obsolète.
- C7 : **I/O**. Contact d'entrées / sorties permettant une communication half-duplex entre la carte à puce et le lecteur.
- C8 : **RFU**. Non utilisé.

B.2 ISO 7816-3 : ATR et protocoles de transport

La troisième partie du standard détaille les signaux électriques relatifs aux différents contacts présentés ci-dessus, ainsi que le format de l'ATR (Answer To Reset), et les différents protocoles de transmission disponibles pour le transport des données applicatives. Cependant, nous ne détaillerons pas les spécifications des signaux électriques dans cette annexe.

Lorsque la carte reçoit un signal de remise à zéro par l'intermédiaire du contact RST, elle doit renvoyer une réponse, nommée **ATR**, comportant jusqu'à 33 octets, dont deux au moins, nommés **TS** et **T0**, sont obligatoires.

L'octet **TS** est le caractère initial. Il peut prendre deux valeurs : 0x3B pour indiquer une logique directe, et 0x3E pour une logique inverse des signaux électriques.

L'octet **T0**, le caractère de format, indique via ses 4 bits de poids fort la présence de 4 octets d'interface optionnels nommés **TA1**, **TB1**, **TC1**, **TD1**. Il code aussi la présence d'octets d'historique dont le nombre, représenté sur les 4 bits de poids faible, est limité à 15.

Les octets d'interface contiennent des valeurs de paramètres relatifs au protocole de transport utilisé, tels que la durée nominale d'un bit ou la valeur de *Vpp*. L'octet **TD1** code la présence ou l'absence de quatre octets d'interface additionnels optionnels (TA2, TB2, TC2, TD2), ainsi que le protocole de transport (T=0 ou T=1) mis en œuvre pour la communication, protocole dont la description est donnée ci-dessous. De manière plus générale, chaque octet d'interface TDi code la présence ou l'absence de quatre octets d'interface d'indice i+1. Si l'octet TD1 est absent de l'ATR, c'est le protocole de transport T=0 qui sera sélectionné par défaut.

Enfin, un octet **TCK**, résultat du XOR de tous les octets de l'ATR à partir de T0, figure à la fin de l'ATR dans le cas où le protocole de transport figurant dans TD1 n'est pas T=0.

Nous illustrons cette description par deux exemples d'ATR :

- Carte Sésame Vitale : 3F 65 25 00 2C 09 69 90 00 (contient les octets optionnels TB1 = 0x25 et TC1 = 0x00, ainsi que 5 octets d'historique)
- Carte EAP-TLS GX4 : 3B 7D 96 00 00 80 31 80 65 B0 83 11 C0 C8 83 00 90 00 (contient les octets optionnels TA1 = 0x96, TB1 = 0x00, et TC1 = 0x00, ainsi que 13 octets d'historique)

Il existe deux protocoles de transport disponibles pour assurer la communication entre la carte et le lecteur. Le choix de ce protocole est codé dans l'ATR sur les quatre bits de poids faible de l'octet optionnel TD1. La valeur de ces quatre bits peut déterminer théoriquement jusqu'à 15 protocoles, mais seuls les protocoles T=0 et T=1 sont actuellement définis.

Le protocole T=0 fonctionne en mode caractère, selon lequel chaque octet de donnée utile est précédé d'un bit de start, et suivi d'un bit de parité ainsi que d'au moins deux bits de stop (le nombre

de ces derniers étant fixé par l'octet TC1 de l'ATR). Il s'agit du protocole le plus utilisé pour le mode contact.

Le protocole T=1 est quant à lui orienté bloc, mais est rarement utilisé en mode contact. Un bloc est constitué d'un en-tête de 3 octets, suivi d'un champ optionnel d'information utile (une APDU) contenant jusqu'à 254 octets. Ce dernier est enfin suivi d'un ou de deux octets de somme de contrôle.

B.3 ISO 7816-4 : Format des APDUs

Les données applicatives sont transmises selon un format défini dans la quatrième partie du standard ISO 7816, laquelle contient la définition de plusieurs commandes de base ainsi que la description du stockage arborescent des fichiers dans une carte (répartition en *Elementary Files*, *Dedicated Files* et *Master File*). L'application EAP-TLS utilisée au cours de ces travaux ne faisant pas usage de cette arborescence, cette partie n'est pas décrite dans cette annexe.

Le format des APDUs diffère selon qu'il s'agit d'une requête à la carte ou d'une réponse de cette dernière.

Les requêtes sont constituées d'un en-tête de 4 octets obligatoires, notés **CLA**, **INS**, **P1**, **P2**. Ces 4 octets sont éventuellement suivis d'octets optionnels constituant le corps de la requête et répartis selon trois champs, figurant dans cet ordre : **Lc**, **Data**, **Le**. Les champs *Lc*, et *Le* peuvent contenir, s'ils sont présents, de 1 à 3 octets. Une requête APDU est donc traditionnellement représentée ainsi :

CLA INS P1 P2 [Lc] [Data] [Le]

L'octet **CLA** désigne la classe d'instructions à laquelle fait référence l'octet INS ; il peut notamment servir à préciser si les instructions sont celles référencées dans le standard ISO 7816-4 ou bien s'il s'agit d'instructions spécifiques au concepteur de l'application. L'octet **INS** désigne l'instruction envoyée à la carte, par exemple une lecture ou une écriture. Les octets **P1** et **P2** sont quant à eux des paramètres dont la fonction dépend de l'instruction ; il peut par exemple s'agir d'un offset pour la lecture ou l'écriture.

Une requête peut être de quatre types : entrante, sortante, entrante et sortante, ou ni entrante ni sortante.

- Une requête ni entrante ni sortante, par exemple l'appel à une fonction n'attendant aucun paramètre tel qu'un *reset* de l'application, ne contient pas de donnée utile, et la réponse n'en contient pas non plus. Dans ce cas aucun des champs optionnels ne figure dans la requête.
- Une requête entrante, par exemple une écriture, contient des données fournies dans le corps de l'APDU via le champ **Data**, dont la longueur est alors précisée par le champ **Lc**. En revanche la réponse ne fournit pas de donnée

- Une requête sortante, par exemple une lecture, ne contient pas de donnée, mais indique à la carte via le champ **Le** la longueur des données qu'elle doit renvoyer, données qui figurent alors dans la réponse.

- Une requête entrante et sortante transmet des données en entrée et attend des données en réponse. Les trois champs **Lc**, **Data** et **Le** figurent dans la requête.

Les réponses de la carte sont quant à elle constituées de deux octets obligatoires **SW1 SW2**, appelés *Status Words*, éventuellement précédés de données dans le cas d'une réponse à une requête sortante. Le statut 90 00 indique le bon déroulement de la commande.

Les APDUs sont transportées par un protocole de transport, typiquement T=0 ou T=1. La structure d'un TPDU (Transport Protocol Data Unit) pour le protocole T=1 a été évoquée dans la section B.2. La structure d'un TPDU pour le protocole T=0 est quant à elle extrêmement proche du format d'une APDU.

Les TPDUs du protocole T=0 sont constitués d'un en-tête de 5 octets obligatoires, contre 4 pour les APDUs : **CLA INS P1 P2 P3**. Cet en-tête est suivi d'un corps optionnel constitué de données. Les quatre premiers octets ont la même signification que dans le cas des APDUs. **P3** est un paramètre qui vaut 0x00 dans le cas d'une commande ni entrante ni sortante, qui vaut **Lc** (compris entre 1 et 255) dans le cas d'une commande entrante (laquelle contient alors *Lc* octets de données), et qui vaut **Le** (entre 1 et 256, 256 octets étant représentés par *Le* = 0x00) dans le cas d'une commande sortante.

Par exemple, une commande de lecture (INS=0xB0) de 16 octets (*Le* = 0x10) à partir de l'offset 0x1234 produit la trace suivante :

>> 00 B0 12 34 10

<< [16 octets] 90 00

Lorsqu'une commande doit être à la fois entrante et sortante, le protocole T=0 ne permet pas d'envoyer et de recevoir des données en un seul échange. Pour cela, une commande spécifique, appelée Get-Response (INS = 0xC0), permet d'aller récupérer les données que souhaite émettre la carte. Cette commande doit être automatiquement envoyée en cas de statut indiquant cet état de fait, tel que 61 XX, où le second octet indique la longueur des données à émettre, qui est immédiatement réémise en paramètre P3 = *Le* de la commande Get-Response.

Par exemple, une commande incluant 8 octets de données (*Lc* = 0x08) et devant en recevoir 64 en retour (*Le* = 0x40) se déroule ainsi :

>> A0 8A 00 00 08 01 23 45 67 89 AB CD EF

>> 61 40

>> A0 C0 00 00 40

<< [64 bytes] 90 00

B.4 Exemples de traces issues de cartes à puce EAP-TLS

L'application EAP-TLS développée et utilisée au cours de ce travail de thèse a nécessité l'implémentation de nombreuses APDUs spécifiques ne figurant pas dans le standard ISO 7816-4. Par ailleurs, l'application ne faisant pas usage de la structure arborescente de fichiers de données proposée par le standard, plusieurs commandes basiques, telles que la lecture et l'écriture de données binaires, ont été réécrites en conséquence. Les principales commandes implémentées dans l'application EAP-TLS sont décrites dans la section 3.3.4.

Cette section se propose de présenter quelques traces d'échanges d'APDUs dans le contexte d'une carte EAP-TLS. Deux phases d'échanges peuvent être distinguées lors d'une utilisation standard de l'application EAP-TLS, typiquement une connexion à un serveur Web en authentification mutuelle.

Une phase d'initialisation est systématiquement lancée lors de la détection de la carte par le logiciel proxy. Cette phase est constituée de l'envoi de 3 APDUs : *Select AID*, *User PIN*, et *Set Identity* :

Tx: 00 A4 04 00 10 A0 00 00 00 30 00 02 FF FF FF FF 89 31 32 38 00 (Select AID)

Rx: 90 00

Tx: 00 20 00 00 04 30 30 30 30 (User PIN code = '0000')

Rx: 90 00

Tx: A0 16 00 80 06 63 6C 69 65 6E 74 (Set Identity : Identity = "client")

Rx: 90 00

Une session EAP-TLS est amorcée par l'envoi à la carte d'une commande *Reset*, qui remet à zéro la machine à états EAP, puis d'un paquet *EAP-TLS Start*. S'ensuit un dialogue EAP-TLS conventionnel transporté par des APDUs, dialogue présenté ici de manière tronquée. Une fois la session établie, une commande *Get Key Block* est envoyée à la carte afin de transmettre au poste client les clés temporaires de session :

Tx: A0 19 10 00 00 (Reset)

Rx: 90 00

Tx: A0 80 00 00 0A 01 14 00 06 0D 20 4D DD 21 F3 (EAP-TLS Start)

Rx: 61 5C

Tx: A0 C0 00 00 5C (Get Response)

Rx: 02 14 00 5C 0D 80 00 00 00 52 16 03 01 00 4D 01

00 00 49 03 01 4D DD 21 F3 F2 71 4A 5F A7 BA 13

57 72 93 57 93 77 E7 AC 80 6F 56 ED 33 CE 74 0A

D1 00 64 0D 06 20 48 60 B9 60 BB A2 61 C4 00 A9

FC 17 E9 E8 5C 1E D8 14 61 07 3E DC 91 41 D2 54

AC 21 C1 A1 7C 12 00 02 00 04 01 00 90 00 (Client Hello)

[...]

Tx: A0 80 00 00 31 01 03 00 31 0D 00 14 03 01 00 01

01 16 03 01 00 20 E5 9C ED AD DF CE 78 DE D8 DD

E9 6D 0F F5 2D 3F 0D 03 5A 9A 3B 1C EB D6 54 39

C3 22 C1 B4 F2 4B (Change Cipher Spec / Server Finished)

Rx: 61 06

Tx: A0 C0 00 00 06

Rx: 02 03 00 06 0D 00 90 00

Tx: A0 82 CA 00 40 (Get Key Block)

Rx: 55 3C D0 0B 7C 78 79 D9 03 9A ED 96 F7 01 1E 58

C0 3D 52 E1 7F 6A EC EB D1 A1 26 5F A0 2F 7A E8

24 0C D3 85 A4 77 95 01 C3 BB 9B 27 11 19 A9 DB

DE 61 BD 1A CA 52 97 CB 37 A7 72 65 CE A1 37 C9

90 00

Publications

Conférences

- *An Innovative Solution for Cloud Computing Authentication: Grids of EAP-TLS Smart Cards,* Pascal Urien, Christophe Kiennert, Estelle Marie ; ICDT 2010, Fifth International Conference on Digital Telecommunications, 13-19 Juin 2010, Athènes. **Best Paper Award**

- *A New Convergent Identity System Based on EAP-TLS Smart Cards,* Pascal Urien, Christophe Kiennert, Estelle Marie ; IEEE SAR-SSI 2011, 6è Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information, 18-21 Mai 2011, La Rochelle

- *A breakthrough for prepaid payment: end to end token exchange and management using secure SSL channels created by EAP-TLS smart cards,* Pascal Urien, Marc Pasquet, Christophe Kiennert ; IEEE CTS 2011, International Conference on Collaboration Technologies and Systems, 23-27 Mai 2011, Philadelphie

- *A New Mobile NFC Key for Delivering Services*, Pascal Urien, Christophe Kiennert ; IEEE SAR-SSI 2012, 7è Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information 22-25 Mai 2012, Cabourg (accepté)

- *A New Cooperative Architecture For Sharing Services Managed by Secure Elements Control By Android Phones With IP Objects*, Pascal Urien, Christophe Kiennert ; IEEE CTS 2012, International Conference on Collaboration Technologies and Systems, 21-25 Mai 2012, Denver (accepté)

Démonstration

- *A New Keying System for RFID Lock Based on SSL Dual Interface NFC Chips and Android Mobiles,* Pascal Urien, Christophe Kiennert ; IEEE CCNC 2012 Demo, 15-16 Janvier, Las Vegas

Brevets

- Ethertrust, *Procédé de sécurisation d'une architecture d'authentification, dispositifs matériels et logiciels correspondants*, Pascal Urien, Christophe Kiennert, Estelle Marie, déposé à l'INPI

- Ethertrust, *Téléchargement sécurisé de clés Mifare via des Smartphones NFC*, Pascal Urien, Christophe Kiennert, Estelle Marie, déposé à l'INPI

Acronymes

AES	Advanced Encryption Standard
AID	Application IDentifier
AJAX	Asynchronous Javascript and XML
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASCII	American Standard Code for Information Exchange
CA	Certificate Authority
CAD	Card Acceptance Device
CBEFF	Common Biometric Exchange Formats Framework
CGI	Common Gateway Interface
CN	Common Name
CPU	Central Processing Unit
DES	Data Encryption Standard
DNS	Domain Name System
EAP	Extensible Authentication Protocol
ECC	Elliptic Curve Cryptography
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMV	Europay MasterCard and VISA
GSM	Global System for Mobile Communications
HMAC	Hash-based Message Authentication Code
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
IBE	Identity-Based Encryption
IdP	Identity Provider
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
JCRE	JavaCard Runtime Environment

JDK	Java Development Kit
JSR	Javacard Specification Request
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
OTA	Over The Air
OTP	One Time Password
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
RFID	Radio Frequency IDentification
ROM	Read-Only Memory
SAML	Security Assertion Markup Language
SIM	Subscriber Identity Module
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SP	Service Provider
SSL	Secure Socket Layer
SSO	Single Sign On
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator

USB	Universal Serial Bus
USIM	Universal Subscriber Identity Module
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
WS	Web Service
XML	Extensible Markup Language
XRI	Extensible Resource Identifier

Références

[1] : L. Jean Camp, *Digital Identity*, IEEE Technology and Society Magazine, Fall 2004

[2] : *Internet Protocol*, RFC 791, September 1981

[3] : T. Berners-Lee, R. Fielding, H. Frystyk, *Hypertext Transfer Protocol – HTTP 1.0*, RFC 1945, May 1996

[4] : *Transmission Control Protocol*, RFC 793, September 1981

[5] : P. Mockapetris, *Domain Names – Concepts and Facilities*, RFC 882, November 1983

[6] : J.B. Postel, *Simple Mail Transfer Protocol*, RFC 821, August 1982

[7] : S. Kent, R. Atkinson, *Security Architecture for the Internet Protocol*, RFC 2401, November 1998

[8] : A. Freier, P. Karlton, P. Kocher, *The SSL protocol version 3.0*, Internet Draft, March 1996

[9] : T. Dierks, C. Allen, *The TLS protocol version 1.0*, RFC 2246, January 1999

[10] : R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, *DNS Security Introduction and Requirements*, RFC 4033, March 2005

[11] : C. Adams, S. Farrell, *Internet X.509 Public Key Infrastructure: Certificate Management Protocols*, RFC 2510, March 1999

[12] : P. Urien, S. Elrharbi, *Tandem smart cards: enforcing trust for TLS-based network services*, 8th International Workshop on Applications and Services in Wireless Networks (ASWN 2008), October 9th – 10th, Kassel, Germany

[13] : B. Kaliski., J. Staddon., *PKCS #1: RSA Cryptography Specifications version 2.0*, RFC 2437, October 1998

[14] : V. Miller, *Use of elliptic curves in cryptography*, *CRYPTO* 85, 417–426

[15] : D. Boneh, M. K. Franklin, *Identity-Based Encryption from the Weil Pairing*, Advances in Cryptology - Proceedings of CRYPTO 2001

[16] : M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, B. de Weger, *Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate*. Cryptology ePrint Archive, Report 2009/111. http://eprint.iacr.org/

[17] : M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, *SIP: Session Initiation Protocol*, RFC 2543, March 1999

[18] : OpenID community, *OpenID Authentication 2.0*, online: http://openid.net/specs/openid-authentication-2_0.html

[19] : S. Cantor et al., *Assertion and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS Standard, 15 March 2005, online: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[20] : B. Aboba, D. Simon, *PPP EAP TLS Authentication Protocol*, RFC 2716, October 1999

[21] : *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* (2007 revision), IEEE-SA, 12 June 2007

[22] : W. A. Arbaugh, *An inductive chosen plaintext attack against WEP/WEP2*, http://www.cs.umd.edu/~waa/attack/ v3dcmnt.htm, May 2001

[23] : C. E. Shannon, *A mathematical theory of communication* Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948

[24] : *Electronic Authentication Guideline, Recommendations of the National Institute of Standards and Technology*, NIST Special Publication 800-63, V.1.0.2, online : http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf

[25] : Microsoft Safety & Security Center, *Check your password – is it strong ?*, online : https://www.microsoft.com/security/pc-security/password-checker.aspx

[26] : *John the Ripper password cracker*, Openwall, online : http://www.openwall.com/john/

[27] : *The Onion Router*, The Tor Project, September 2002, online : https://www.torproject.org/

[28] : M. E. Eldefrawi, K. Alghathbar, M. K. Khan, *OTP-Based Two-Factor Authentication Using Mobile Phones*, 8th International Conference on Information Technology : New Generations, ITNG 2011

[29] : N. Haller, *The S/KEY One–Time Password System*. In: Proceedings of the ISOC Symposium on Network and Distributed System Security, 1994, pp. 151-157

[30] : D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, O. Ranen, *HOTP : An HMAC-Based One-Time Password Algorithm*, RFC 4226, December 2005

[31] : D. M'Raihi, S. Machani, M. Pei, J. Rydell, *TOTP : Time-Based One-Time Password Algorithm*, RFC 6238, May 2011

[32] : Initiative for Open Authentication, *OATH Reference Architecture Version 2.0*, online : http://www.openauthentication.org/webfm_send/1

[33] : B. Elad, E. Biham, N. Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*, Crypto 2003, 600–16

[34] : D.C. Plummer, *An Internet Address Resolution Protocol*, RFC 826, November 1982

[35] : E., Rescorla, "HTTP Over TLS", RFC 2818, May 2000

[36] : M. Marlinspike, New Tricks for Defeating SSL in Practice, Black Hat DC, July 2009, online : https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf

[37] : *SSLStrip, a HTTPS stripping attack tool*, online : http://www.thoughtcrime.org/software/sslstrip/

[38] : J. Stapleton, *American National Standard X9.84-2001 : Biometric Information Management and Security*, INCITS, 2001

[39] : CBEFF, Common Biometric Exchange Formats Framework, NISTIR 6529-A, NIST, April 2004

[40] : P. Funk, S. Blake-Wilson, *Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)*, RFC 5281, August 2008

[41] : P. Urien, C. Kiennert, *EAP-BIO*, draft-urien-kiennert-emu-bio-00.txt, October 2009

[42] : T. Matsumoto, H. Matsumoto, K. Yamada, S. Hoshino, *Impact of Artificial Gummy Fingers on Fingerprint Systems*, Proceedings of SPIE, vol. 4677, January 2002

[43] : Y.N. Singh, S.K. Singh, *Vitality detection from biometrics: State-of-the-art*, World Congress on Information and Communication Technologies, WICT, December 2011

[44] : J. Postel, *File Transfer Protocol*, RFC 765, June 1980

[45] : H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, February 1997

[46] : R. Rivest, *The MD5 message-digest algorithm*, RFC 1321, April 1992

[47] : D. Eastlake, P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, RFC 3174, September 2001

[48] : X. Wang, Y. Yu, *How to Break MD5 and Other Hash Functions*, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT, 2005

[49] : S. Manuel, *Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1*, IACR ePrint Archive, Report 2008/469

[50] : M. Stevens, A.K. Lenstra, B. de Weger, *Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, EUROCRYPT, 2007

[51] : O. Acucmez, J.P. Seifert, C.K. Koc, *Micro-Architectural Cryptanalysis*, Security & Privacy Magazine, IEEE Volume 5, Issue 4, July-August 2007, Pages 62 – 64

[52] : D.A. Osvik, A. Shamir, E. Tromer, *Cache attacks and countermeasures: the case of AES*, proceedings of RSA Conference Cryptographers Track (CT-RSA) 2006, LNCS 3860, 1 – 20, Springer, 2006

[53] : P. Eronen, H. Tschofenig, *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*, RFC 4279, December 2005

[54] : L.C. Paulson, *Inductive analysis of the Internet protocol TLS*, ACM Transactions on Computer and System Security, Vol. 2, Number 3 : 332–351, 1999

[55] : C. He, M. Sundararajan, A. Datta, A. Derek, J. Mitchell, *A Modular Correctness Proof of IEEE 802.11i and TLS*, CCS'05, November 7-11, 2005

[56] : Microsoft, *Introducing Windows CardSpace*, online : http://msdn.microsoft.com/en-us/library/aa480189.aspx

[57] : T. Wason et al., *Liberty ID-FF Architecture Overview*, Liberty Alliance Project, online : http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications

[58] : S. Cantor et al., *Shibboleth Architecture, Protocols and Profiles*, 10 September 2005, online : http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf

[59] : C. Neuman, T. Yu, S. Hartman, K. Raeburn, *The Kerberos Network Authentication Service (V5)*, RFC 4120, July 2005

[60] : E. Maler, D. Reed, *The Venn of Identity: Options and Issues in Federated Identity Management*, Security & Privacy, IEEE Volume 6 , Issue 2, 2008

[61] : A. Nanda, *Identity Selector Interoperability Profile V1.0*, Microsoft, April 2007

[62] : W.A. Alrodhan, C.J. Mitchell, *Improving the security of CardSpace*, In : EURASIP Journal on Information Security, Vol. 2009

[63] : *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, May 2000, online : http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[64] : *Web Services Architecture*, W3C Working Group Note, February 2004, online : http://www.w3.org/TR/ws-arch/

[65] : *WS-Trust 1.3*, OASIS Standard, March 2007, online : http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html

[66] : *Web Service Federation Language (WS-Federation)*, Version 1.1, December 2006, online : http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP

[67] : *OpenID Provider Authentication Policy Extension 1.0*, December 2008, online : http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html

[68] : T.M. Jurgensen et al., *Smart Cards: The Developer's Toolkit*, Prentice Hall PTR, ISBN 0130937304, 2002

[69] : *ISO/IEC 7816 Smart Card Standard*, Parts 1-4 available online : http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx

[70] : *TCG Specification, Architecture Overview*, Specification, Revision 1.3, March 2007, Trusted Computing Group

[71] : *A method of supporting SSL/TLS protocols in a resource constrained device*, Patent# WO 2006/018680

[72] : G. Yang, *Introduction to TCP/IP Network Attacks*, online : http://seclab.cs.sunysb.edu/sekar/papers/netattacks.pdf

[73] : L. Blunk, and J. Vollbrecht, *PPP Extensible Authentication Protocol (EAP)*, RFC 2284, March 1998

[74] : W. Simpson, *The Point-to-Point Protocol (PPP)*, RFC 1661, July 1994

[75] : IEEE Draft P802.1X/D11, *Standards for local and metropolitan area networks: Standard for port based network access control*, Mars 2001

[76] : C. Rigney, A. Rubens, W. Simpson, S. Willens, *Remote Authentication Dial In User Service (RADIUS)*, RFC 2865, June 2000

[77] : *PC/SC Workgroup Specifications Overview*, PC/SC Workgroup, online : http://www.pcscworkgroup.com/specifications/overview.php

[78] : PKCS #15 v1.1 : *Cryptographic Token Information Syntax Standard*, RSA Laboratories, June 2000

[79] : D. Naccach, D. M'Raihi, *Cryptographic smart cards*, Micro, IEEE, Volume 16, Issue 3, June 1996 pp 14, 16 – 24

[80] : M. Badra, P. Urien, *Adding Identity Protection to EAP-TLS Smartcards*, in Proceedings of the IEEE Wireless Communications and Networking Conference, WCNC 2007, pp 2951 – 2956

[81] : B. Kaliski, *PKCS #1 : RSA Encryption Version 1.5*, RFC 2313, March 1998

[82] : J. Jonsson, B. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, RFC 3447, February 2003

[83] : D. Bleichenbacher, *Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1*, in Advances in Cryptology – CRYPTO'98, LNCS vol. 1462, pages 1-12, 1998

[84] : T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, August 2008

[85] : 3GPP TS 27.07 v6-3-0, *3rd Generation Partnership Project; Technical Specification Group Terminals; AT command set for User Equipment (UE)*, June 2003

[86] : P. Urien, S. Elrharbi, *Tandem smart cards: enforcing trust for TLS-based network services*, 8th International Workshop on Applications and Services in Wireless Networks (ASWN), October 2008

[87] : J.J. Garrett, Ajax: *A New Approach to Web Applications*, February 2005, online : http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[88] : *What is Android ?*, online : http://developer.android.com/guide/basics/what-is-android.html

[89] : S. Chaumette et. al., *Secure distributed computing on a Java Card grid*, 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 2005

[90] : P. Urien, M. Dandjinou, *Introducing Smartcard Enabled RADIUS Server*, The 2006 International Symposium on Collaborative Technologies and Systems (CTS), 2006

[91] : J. Postel, *User Datagram Protocol*, RFC 768, August 1980

[92] : P. Mell, T. Grance, *The NIST Definition of Cloud Computing*, Recommendations of the National Institute of Standards and Technology, Special Publication 800-145

[93] : Bulletin de la Banque de France N°25 – Janvier 1996

[94] : N. Asokan, P.A. Janson, M. Steiner, M. Waidner, *The state of the art in electronic payment systems*, Computer, Volume 30, Issue 9, 1997

[95] : J. Liu, Y. Xiao, H. Chen, S. Ozdemir, S. Dodle, V. Singh, *A Survey of Payment Card Industry Data Security Standard*, Communications Surveys & Tutorials, IEEE Volume 12, Issue 3, 2010

[96] : *PCI SSC Data Security Standards Overview*, PCI Security Standards Council, online : https://www.pcisecuritystandards.org/security_standards/

[97] : *EMV Books, 1 - Application Independent ICC to Terminal Interface Requirement and Application Selection, Book 2 - Security and Key Management, Book 3 – Application Specification, Book 4 - Cardholder, Attendant, and Acquirer Interface Require*. Online : http://www.emvco.org

[98] : S. Cakaj, M. Shefkiu, S. Haxha, *Implementation of prepaid services in Kosovo's fixed network*, ELMAR '09, International Symposium

[99] : *What is NFC ?*, NFC Forum, online : http://www.nfc-forum.org/aboutnfc/

[100] : ISO/IEC 14443 : *Identification cards -- Contactless integrated circuit cards -- Proximity cards*, Parts 1 – 4, 2008

[101] : ISO/IEC 18092 : *Information technology -- Telecommunications and information exchange between systems -- Near Field Communication -- Interface and Protocol*, 2004

[102] : *Mifare Standard Card IC MIF1 IC S50*, Functional Specification, Revision 5.1, Philips semiconductors, May 2001

[103] : International Civil Aviation Organization, *Machine Readable Travel Documents*, ICAO Document 9303, Parts 1 – 3

[104] : W. Enck, M. Ongtang, P. McDaniel, *Understanding Android Security*, IEEE Security & Privacy, Volume 7 , Issue 1, 2009

[105] : P. Baentsch, P. Buhler, T. Eirich, F. Horing, M. Oestreicher, *JavaCard-from hype to reality*, Concurrency, IEEE Volume 7, Issue 4, 1999

[106] : Certification Report, BSI-DSZ-CC-0348-2006 for Philips Secure Smart Card Controller P5CT072V0P, P5CC072V0P, P5CD072V0P and P5CD036V0P each with specific IC Dedicated Software from Philips Semiconductors GmbH Business Line Identification, online : https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte03/0348a_pdf.pdf?__blob=publicationFile

[107] : Certification Report, BSI-DSZ-CC-0426-2007 for NXP P541G072V0P (JCOP 41 v2.3.1) from IBM Deutschland Entwicklung GmbH, online : http://www.commoncriteriaportal.org/files/epfiles/0426a.pdf

[108] : ISO/IEC 9798-2 : *Information technology -- Security techniques -- Entity authentication -- Part 2: Mechanisms using symmetric encipherment algorithms*, 1999

[109] : G. de Koning Gans, J.H. Hoepman, F.D. Garcia, *A Practical Attack on the MIFARE Classic*, in Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications, Volume 5189 of Lecture Notes in Computer Science, Pages 267 – 282, 2008

[110] : F.D. Garcia et al., *Dismantling MIFARE Classic*, in Computer Security – ESORICS '08, Volume 5283 of Lecture Notes in Computer Science, Pages 97 – 114, 2008

[111] : AN02105, *Secure Access to Mifare Memory on Dual Interface Smart Card ICs*, Application Note, Philips semiconductors, January 2002

[112] : *TOP IM GX4*, Product Information, Gemalto, November 2008, online : http://www.gemalto.com/dwnld/5946_TOP_GX4_MSA030_Nov08.pdf