Le transport et la sécurisation des échanges sur les réseaux sans fil

Mohamad Badra

TELECOM PARIS
école nationale supérieure des télécommunications

Travaux de recherche pour l'obtention de la

Thèse de doctorat

De l'Ecole Nationale Supérieure des Télécommunications

Spécialité : Informatique et Réseaux

Par

Mohamad Badra

Le transport et la sécurisation des échanges sur les réseaux sans fil

Jury

Bernard Cousin Pierre Paradinas		Rapporteurs
Mhamed Abdallah Hossam Afifi Omar Cherkaoui Bertrand Du Castel Guy Pujolle Ahmed Serhrouchni		Examinateurs
Philippe Godlewski Pascal Urien		Directeurs de thèse
Ouahiba Fouial		Invitée

A tous ceux qui me sont les plus chers : mon père, ma mère, mes sœurs et mes frères.
A mes nombreux et merveilleux neveux et nièces.

A mon oiseau.

"The secret to creativity is knowing how to hide your sources"

Albert Einstein

Remerciements

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

Mes premiers remerciements sont adressés tout d'abord au professeur Philippe Godlewski, directeur de cette thèse pour avoir accepté de m'encadrer, en tant que doctorant, au cours des deux premières années de labeur. J'apprécie surtout leur regard critique sur ces travaux.

Mes sincères remerciements sont également adressés au professeur Pascal Urien. Je tiens à lui exprimer mes sincères respects pour m'avoir encadré au cours des derniers quatorze mois de ma thèse. Je n'oublierai jamais ses qualités scientifiques, humaines et mentales qui ont contribué énormément à la progression de mes travaux de recherche.

Je tiens à exprimer ma profonde reconnaissance à Monsieur Ahmed Serhrouchni pour m'avoir fait confiance et m'avoir aidé à aller jusqu'au bout et ainsi pour avoir suivi mes recherches qui sans lui, n'auraient jamais eu lieu. Qu'il soit remercié d'être à la fois une référence pour mes recherches et un vrai frère.

Je tiens également à remercier les membres de mon jury de thèse. Je suis reconnaissant envers Monsieur Bernard Cousin, professeur à l'université de Rennes-1, et Monsieur Pierre Paradinas, professeur au CNAM, pour avoir bien voulu rapporter cette thèse. Mes remerciements vont également aux examinateurs de ma thèse qui ont eu l'amabilité d'examiner ma thèse.

Je remercie Monsieur Bertrand Du Castel, directeur de recherche, Axalto (USA) d'avoir accepté la présidence du jury.

Je tiens à remercier également Monsieur Mhamed Abdallah, maître de conférence à l'INT, Monsieur Hossam Afifi, professeur à l'INT, Monsieur Guy Pujolle, professeur à l'université Paris VI, Monsieur Omar Cherkauoi, professeur à QNAM (Canada), et à Mademoiselle Ouahiba Fouial, docteur à l'université de Nancy, d'avoir examiner mon travail.

Mes remerciements chaleureux vont à toutes les personnes avec qui j'ai été amené à discuter et à valider mes travaux de recherche et plus particulièrement, aux membres du groupe de travail TLS au sein de l'IETF.

Une attention toute spéciale s'adresse aux membres du département InfRes, les professeurs, les collègues et les amis. Je remercie particulièrement Céline, Rola, Dany, Haje Ahmed, Jacques, Mazen, les frères Mohamed et Walid qui ont réussi à me supporter pendant ces trois ans. Merci à Salim et à Peter Weyer-Brown, professeur d'Anglais à l'ENST pour leur patience à toute épreuve. Quant à Ouahiba, je te remercie pour ta lecture pointilleuse, tes remarques pertinentes et surtout pour ta gentillesse. Mes remerciements sucrés vont à Rana, Mejdi et Rabah pour les pauses café sans sucre.

Un pense particulière est adressée aux gens avec qui j'ai partagé mon bureau durant ces trois années. Je pense particulièrement à Hazar et Nicolas et je les remercie énormément pour leur gentillesse.

Je garde une place toute particulière pour ma famille, ma mère Halimé, mon père Kamel, mes deux sœur Ramzia et Fatima, mes trois frères Mahmoud, Omar et Ali. Je voudrais leur exprimer toute ma profonde reconnaissance et je leur remercie du fond de mon cœur parce qu'ils m'ont

constamment aidé, malgré la distance, par leur soutien moral et leurs encouragements pour achever cette thèse.

Mon quatrième frère est un cadeau du ciel. Ibrahim, je grade une place particulière dans mon cœur pour toi. Je te promets Barhoum qu'aucune des mes joies n'aura lieu sans qu'elle sera partagée avec toi ou sans se souvenir de toi. Tu m'as supporté durant cette longue thèse. Merci pour ton écoute, ta patience lors de mes moments de doute et pour ton aide scientifique.

Je pense à mes amis d'enfance, à Walid, Hassan, Abdel Rahman, et à tous les habitants de mon village MechMech (la république!).

Résumé

La convergence des réseaux fixes et des réseaux mobiles est une réalité. Les couvertures de ces réseaux sont de plus en plus confondues. Leur intégration dans une architecture commune est une priorité pour les opérateurs et fournisseurs de services. Cela afin de mieux répondre aux problématiques introduites par cette convergence en termes d'interopérabilité, de performance, de qualité de service, de sécurité, d'exploitation et également de réactivité liée au déploiement de nouveaux services.

Concernant la sécurité, beaucoup de travaux et d'efforts ont été consentis ces dernières années afin d'aboutir à des solutions immédiates pour sécuriser les échanges dans les réseaux fixes. Ces solutions, telles que TLS (Transport Layer Security) et IPSec ont été ainsi conçues dans un contexte où les équipements et les entités sont fixes, elles sont opérationnelles à grande échelle.

Malgré leur diversité, ces solutions sont encore limitées, génériques et répondent insuffisamment aux besoins spécifiques des applications de communication dans les environnements mobiles. Nous avons donc opté pour des solutions d'adaptation qui permettent d'adapter les mécanismes de sécurité conçus au départ pour les réseaux fixes aux réseaux mobiles. Ce choix est appuyé par deux raisons principales. La première est que les réseaux sans fil sont opérationnels et reliés de plus en plus aux réseaux fixes et la seconde réside dans le fait que la réutilisation de ces solutions nous permet de réduire leurs coûts d'exploitation. Notre contribution dans cette thèse est donc de faire avancer les solutions de la sécurisation des échanges sans fil tout en prenant en compte les contraintes précédemment citées.

Notre travail de recherche est structuré en quatre parties :

La première partie traite de TLS, de ses performances et de sa charge protocolaire. Dans le but d'étudier son adéquation aux réseaux mobiles, nous expérimentons TLS avec les réseaux GSM, en utilisant la pile protocolaire WAP, et avec les réseaux 802.11 sans fil. Les résultats de cette étude nous amènent à proposer des extensions plus performantes et plus appropriées que les mécanismes standard définis dans WAP et les réseaux 802.11 sans fil.

La deuxième partie est une contribution qui consiste à l'extension et l'enrichissement de TLS pour répondre à des besoins de sécurité dans le contexte du sans fil. Nous avons ainsi proposé de nouvelles architectures pour la convergence avec les réseaux fixes.

Dans la troisième partie, nous proposons d'enrichir la sécurité dans les réseaux WLAN en fournissant des services additionnels comme l'anonymat des échanges, la protection d'identité et la protection contre certains types d'attaques (passives, par dictionnaires, etc.). Nous définissons un mécanisme basé sur l'utilisation d'une clé partagée et de TLS. Cette contribution consiste à ajouter une extension sur le premier message du client TLS tout en respectant la norme "TLS extensions". Ce mécanisme ne nécessite pas l'utilisation des certificats et des PKIs pour l'authentification ; il est mieux adapté pour certains réseaux sans fil et à petite échelle où les clients sont pré configurés ou personnalisés. Nous terminons cette partie en présentant une implantation de EAP-TLS couplée avec une carte à puce.

La dernière partie consiste essentiellement à intégrer les différentes contributions. Ceci pour mettre en exergue une méthode d'authentification couplant "architecture" et "secret partagé".

Nous montrons ainsi comment, avec une telle approche, nous dérivons des services de sécurité non supportés jusqu'à présent par TLS tels que le PFS et la protection de l'identité.

Abstract

The convergence of wired and wireless networks is a reality and their coverage is increasingly merged. Their integration in a common architecture becomes a priority for operators and service providers in order to efficiently address new problems related to this convergence in terms of interoperability, performance, quality of services, security and also reactivity related to new services.

Concerning security, a lot of effort has been made to find immediate solutions in order to secure wired network exchanges and transactions. These solutions, such as TLS and IPsec have been specified for contexts where equipments and entities are fixes; they are operational on large-scale environments.

Despite their diversity, these solutions are limited, generic and insufficiently meet wireless communication application requirements and needs. Therefore, we opted for adaptation solutions, which enable security mechanisms originally specified for wired networks to be adapted to wireless networks and environments. This choice is supported by two main reasons. First, wireless networks are already operational and increasingly related to wired networks. The second reason lies in the fact that the reuse of these solutions allows us to reduce their exploitation costs. Our contribution in this thesis is then to promote and to improve wireless security solutions with regards to previously mentioned constraints.

Our research work is organized in four parts:

The first part involves describing and analyzing the TLS. For the purpose for studying its appropriateness to wireless networks, we experiment TLS with GSM networks, using the WAP protocol stack, and with wireless LAN. These study results lead us to propose new extensions which are more efficient and more appropriate than standard mechanisms defined in the WAP stack and the wireless LAN.

The second part is a contribution consisting in the extension and improvement of TLS in order to meet wireless security requirements and needs. Thus, we propose new architectures in the convergence of wired/wireless networks.

In the third part, we propose to enrich WLAN security by providing additional services such as the anonymity of exchanges, identity protection and the protection against most types of attacks (passive, dictionary, etc.). We define a mechanism based on the use of a pre-shared key and of TLS. This contribution consists in adding an extension to TLS ClientHello message according to the TLS extensions standard. This mechanism allows mutual authentication without it being necessary to use certificates and Public Key Infrastructures and may prove more convenient for most wireless and closed environments where the clients are mostly configured and personalized in advance. We conclude this part by presenting the first implementation of EAP-TLS using EAP smart cards.

The last part shows how our different contributions fit together. Our approach to authentication methods links "architecture" and "shared secret". We demonstrate how, from such an approach, we derive security services such as PFS and identity protection which are not currently supported by TLS.

Table de matières

Liste des figures

Liste des tableaux

Introduction

Avec l'évolution rapide des réseaux sans fil, les exigences en terme de sécurité deviennent de plus en plus indispensables. Ce qui nécessite l'introduction de méthodes avancées d'authentification, de gestion et de distribution des clés entre les différentes entités sur le réseau, tout en respectant les contraintes imposées par les réseaux sans fil, telles que la capacité de l'interface radio et les ressources des terminaux mobiles qui représentent le goulot d'étranglement de ce type de réseaux. Dans cette optique, un protocole de sécurité doit pouvoir établir des sessions sécurisées avec peu d'influence sur la performance globale du réseau, tout en fournissant les différents services de sécurité pour chaque type d'application.

Les réseaux mobiles et sans fil, par leur nature, partagent deux problématiques majeures dans leur conception.

La première problématique due au fait que le support de transmission de ces réseaux est partagée. Ce qui a pour effet d'avoir la possibilité d'intercepter les données envoyées/reçues par le support et par la suite de pouvoir modifier et rejouer les données. L'intrus peut également injecter, saturer ou endommager les équipements du réseau.

La seconde problématique provient du fait que la mobilité fournie par ces réseaux engendre de nouvelles menaces sur la protection d'informations liées aux architectures sans fil. Les mécanismes de sécurité destinés aux réseaux mobiles doivent prendre en compte la mobilité des clients sans fil et réaliser une re-authentification suivant leurs déplacements au cours desquels ils changent leurs domaines de couverture. La re-authentification doit être efficace et performante. Elle ne doit ni augmenter la fluctuation des ressources dans le réseau, ni influencer le taux de transfert et de réception de plusieurs types d'applications comme les médias ou les vidéoconférences.

La plupart des réseaux mobiles n'appliquent pas l'authentification mutuelle entre leurs équipements dans leurs mécanismes de sécurité. Cela permet de réaliser plusieurs types d'attaques ; notamment l'attaque *Man-In-The-Middle* [Rescorla]. Une autre défaillance consiste à pouvoir casser les clés de chiffrement utilisées par des algorithmes de chiffrement faible. Ce qui a pour conséquence de retrouver les données en clair (déchiffrées). En plus, l'absence de l'intégrité des données durant la phase de la signalisation rend possible la réalisation des attaques de type DoS (Denial of Service) [DoS].

Beaucoup de travaux et d'efforts ont été consentis ces dernières années afin d'aboutir à des solutions immédiates pour sécuriser les échanges des réseaux fixes. Ces solutions ont été ainsi conçues dans un contexte où les équipements et les entités sont performants, elles sont opérationnelles à grande échelle (TLS, IPSec, etc.).

Malgré la diversité des ces solutions, elles se veulent encore limitées, génériques et répondent ainsi insuffisamment aux besoins spécifiques des applications de communication dans les environnements mobiles. Nous avons donc opté pour des solutions d'adaptation qui permettent d'adapter les mécanismes de sécurité conçus au départ pour les réseaux fixes aux réseaux mobiles. Ce choix est appuyé par deux raisons principales. La première est que les réseaux sans fil sont opérationnels et reliés de plus en plus aux réseaux fixes et la seconde, réside dans le fait que la réutilisation de ces solutions nous permet de réduire leurs coûts d'exploitation. Notre contribution dans cette thèse est donc de faire avancer la problématique de la sécurisation des réseaux sans fil tout en prenant en compte les contraintes citées précédemment.

Solutions existantes

Dans ces dernières années, plusieurs solutions (TLS [TLS], IPSec [IPSec], JFK [JFK], etc.) ont été proposées afin de répondre aux problématiques de la sécurité des échanges dans les réseaux fixes. Les réseaux sans fil étant une extension des réseaux fixes, ces mêmes solutions sont donc en phase d'adaptation pour la sécurisation des échanges mobiles. Plus particulièrement, les communautés sans fil (EAP, 802.11 sans fil, 3GPP, 3GPP2, etc.) adaptent TLS comme un mécanisme d'authentification, de contrôle d'accès et de distribution des clés. En effet, la simplicité et le déploiement de TLS font de lui le protocole le plus utilisé pour la sécurité des échanges. Il permet d'établir des sessions sécurisées entre deux points communicants. Il offre une architecture ouverte permettant d'introduire plusieurs services de sécurité tels que l'intégrité, la confidentialité et la protection contre différentes attaques.

Malgré tous ces avantages, TLS ne répond pas à toutes les problématiques et les exigences des réseaux filaires et sans fil. Ces problématiques sont liées à la fois, à sa conception, à son coût opérationnel ainsi qu'à l'interaction entre les réseaux fixes et les réseaux mobiles. Parmi ces problématiques, nous pouvons citer :

***1. Incompatibilité* :** la sécurité de bout en bout fournie par TLS est confrontée à des problématiques majeures liées à l'architecture employée. En particulier, dans le cas où deux piles protocolaires incompatibles communiquent entre elles. Cela nécessite l'introduction d'équipements additionnels qui génèrent par conséquent l'impossibilité d'identifier le client auprès du serveur. Parmi les architectures concernées par ce problème, nous étudions les architectures WAP.

1.1. TLS et les architectures WAP : la pile protocolaire WAP [WAPArch] est le résultat du travail du Forum WAP pour favoriser le développement des services sur les réseaux sans fil. Ce forum adapte les protocoles et les langages de l'Internet au monde du mobile et propose par la même occasion différentes architectures. Côté sécurité, le protocole WTLS qui assure la sécurité de la pile WAP est dérivé du protocole TLS pour être appliqué à des réseaux à bande étroite. Côté architecture, la version WAP1.x introduit une passerelle entre la partie fixe et la partie mobile afin d'assurer la conversion entre les différentes couches et plus particulièrement, entre WTLS et TLS. Dans cette version, WTLS assure un lien sécurisé entre le client et la passerelle dans le réseau mobile alors que TLS est utilisé pour assurer le lien sécurisé entre la passerelle et le serveur dans le réseau fixe. Une passerelle WAP traditionnelle ne fournit pas la sécurité de bout en bout entre un terminal WAP et un serveur d'application. En effet, le client chiffre le message à transmettre par une clé secrète partagée avec la passerelle durant la phase du *Handshake* WTLS. La passerelle prend ce message chiffré et le déchiffre avec la même clé secrète. Elle le re-chiffre ensuite avec une autre clé secrète partagée entre elle et le serveur pendant la phase du *Handshake* TLS. Pendant son passage par la passerelle, le message passe en clair et engendre alors un problème de sécurité. Ainsi, cette version n'offre pas la sécurité de bout en bout dont le véritable problème est l'impossibilité d'identifier le client auprès du serveur.

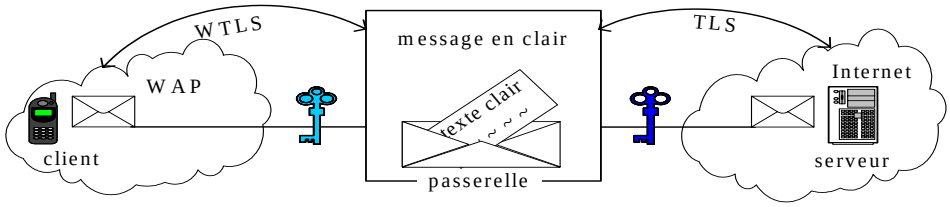

Figure 0.1. Problème de l'architecture WAP1.x

Une deuxième version WAP2.0 répond à ce problème par une implémentation de la pile Internet sur le terminal mobile. Cette solution utilise le protocole TLS de bout en bout entre le client et le serveur. Cependant, un problème majeur de performances se pose, compte tenu du temps pris par les calculs cryptographiques et la charge protocolaire du TLS.

***2. Insuffisance de services offerts*:**

Une autre problématique due à l'application de TLS dans les réseaux mobiles réside dans son insuffisance d'offrir les services nécessaires exigés par les communautés sans fil telles que 802.11 sans fil et EAP-IETF.

2.1. TLS et les architectures 802.11 sans fil : les architectures de 802.11 sans fil ([802.11-Part1], [802.11-Part2]) intègrent par défaut un mécanisme d'authentification de niveau réseau et un chiffrement WEP [WEP]. Cependant, ces mécanismes ne résistent pas suffisamment à plusieurs attaques [802Hack] : de la falsification de l'identité à la récupération de la clé de chiffrement. Afin d'affronter cette problématique, la norme IEEE 802.1X [802.1X] a été introduite dans le but de gérer l'authentification via un serveur centralisé et d'assurer un mécanisme pour une distribution et un renouvellement périodique des clés. Le principe de cette norme est de bloquer le flux de données d'un utilisateur non authentifié.

Un autre problème majeur de 802.1X réside dans son traitement asymétrique de l'authentification entre le terminal et le point d'accès. Par sa conception, cette norme n'offre aucun moyen pour l'intégrité des données. Elle ne résout pas totalement les problèmes liés au WEP.

Cette norme propose l'utilisation du protocole EAP [EAP] comme protocole d'authentification pour les entités de 802.1X. EAP réalise une enveloppe générique pour de multiples méthodes d'authentification. Cependant, 802.11 n'a pas précisé la façon d'implémenter EAP avec 802.1X. Pour cette raison, plusieurs couches sont définies au-dessus de EAP afin de supporter les mécanismes de sécurité. La majorité de ces mécanismes utilise TLS dans l'authentification mutuelle et dans la génération de clés. Cependant, ils ne fournissent pas tous les services de sécurité ; notamment l'échange anonyme de données et la protection d'identité du client. D'autres utilisent TLS pour authentifier le serveur auprès du client et pour établir un canal sécurisé. Ce canal permet d'utiliser d'autres méthodes afin d'authentifier les clients tout en protégeant leurs crédits. L'inconvénient de ces derniers mécanismes est qu'ils ne sont pas protégés contre l'attaque *Man-In-The-Middle* parce que les clés de session générées par le protocole interne ne sont pas reliées avec celles générées par TLS.

Un autre problème lié à l'application de TLS réside dans la logique des réseaux 802.11 sans fil. Durant la phase d'authentification, le serveur envoie son certificat X.509 [X509] au terminal qui doit, à son tour, vérifie si ce certificat est valide et non révoqué. La consultation de la liste de révocation nécessite une connexion Internet. Comme le client n'aura pas cette connexion avant la fin avec succès de la phase d'authentification, il doit supposer que le certificat du serveur est non révoqué et il peut vérifier son hypothèse dès que la connexion est fournie. Bien que cette vérification reste une recommandation, elle est nécessaire dans le cas de *roaming*.

3. Coût cryptographique et charge protocolaire élevés : les spécifications actuelles de TLS permettent l'authentification mutuelle entre les entités via l'utilisation des certificats X.509. L'usage de ces certificats nécessite des infrastructures à clés publiques qui servent à établir une chaîne de certification ou de confiance entre les entités communicantes. Les fonctions requises par les PKIs (opérations cryptographiques, gestion de certificats, charge de données, etc.) rendent

extrêmement coûteux leur usage pour les réseaux mobiles, alors qu'il est déjà complexe pour les réseaux fixes.

4. Limites des mécanismes d'authentification **:** TLS comporte un seul mécanisme asymétrique permettant de partager des clés durant l'établissement de la session sécurisée. Ce mécanisme est basé sur le chiffrement à clé publique qui permet de distribuer et de choisir une clé de session. Deux types de clés publiques sont utilisées: dynamique et statique.

> ***4.1. Pour une clé dynamique*** **:** l'entité génère une paire de clés publique/privée pour chaque session et l'utilise afin de protéger la distribution de la clé de session. Ce type est limité à l'usage de certificats qui prouvent le lien entre la clé publique et l'identité de l'entité. Sans cette preuve, l'attaque de MitM devient facilement réalisable.
>
> ***4.2. Pour une clé statique*** **:** avec ce type, l'entité utilise la clé publique de son certificat durant la phase d'échange des clés de session. Il est bien évidemment obligatoire que les paramètres de l'algorithme appartiennent au certificat de l'entité. Autrement dit, l'identité de l'entité et les paramètres de l'algorithme sont regroupés dans un certificat signé par une autorité de confiance.

Nous nous apercevons qu'un certificat est obligatoire dans les deux types afin de protéger les entités contre plusieurs types d'attaques tels que MitM. Cependant, l'usage de certificats ne fournit ni la protection à long terme des données (PFS) ni la protection de l'identité lorsque nous utilisons les clés statiques. Néanmoins, seulement le service PFS est assuré avec les clés dynamiques.

5. L'anonymat : TLS définit des mécanismes d'authentification basés sur l'échange de certificats. L'introduction du service d'authentification induit de nouveaux besoins tels que l'anonymat. Ce dernier besoin n'est pas fournit par TLS parce que les certificats sont envoyés en clair durant l'établissement de la session. Ce qui permet à un intrus de retrouver facilement l'identité des entités et par la suite d'analyser le trafic des deux parties communicantes.

Contributions

La conception d'un protocole de sécurité doit respecter non seulement les contraintes et les limitations des clients mobiles mais aussi des serveurs d'authentification et d'applications. Elle doit répondre encore aux besoins des réseaux mobiles tels que l'anonymat, la protection à long terme de données et la protection contre les attaques passives. Dans cette optique, nous étendons et élargissons TLS avec d'autres mécanismes d'authentification et de distribution de clés. L'enjeu le plus important de la conception d'une architecture ou d'un protocole de sécurité est de réduire la charge protocolaire, la charge cryptographique et la charge de gestion tout en assurant le niveau de sécurité fourni par les actuelles spécifications de TLS. Dans cette optique, notre thèse traite et propose des architectures et mécanismes d'authentification et de gestion de clés pour les réseaux sans fil et les réseaux fixes à petite échelle.

La convergence de plusieurs facteurs (performance, insuffisance des services offerts, anonymat, etc.) a contribué à définir de nouveaux modes d'authentification avec TLS pour les réseaux fixes à petite échelle et pour les réseaux sans fil. Nous pensons qu'il vaut mieux donc introduire l'authentification à clé partagée pour ces réseaux. En effet, l'utilisation de PKI et de certificats reste applicable quand il s'agit de déployer des certificats du côté serveur ainsi que pour les réseaux à grande échelle. Néanmoins, quand il s'agit de déployer un protocole de sécurité pour une communauté restreinte ou de type sans fil dans laquelle les clients sont pré configurés ou pré

personnalisés, le coût de l'utilisation de PKI devient trop élevé. En plus, il est plus facile et plus simple de gérer un nombre défini de clés partagées qu'une PKI. Ajoutons à cela que l'usage bien défini des clés secrètes pour l'authentification dans les réseaux mobiles ne diminue pas le niveau de la sécurité fourni par les certificats.

Dans ce contexte, nous présentons les quatre principales contributions de cette thèse. Il s'agit de :

1. Une architecture TLS pour la pile WAP.
2. Une extension de TLS afin d'introduire l'authentification basée sur des clés partagées.
3. Une méthode d'authentification EAP pour les réseaux 802.11 sans fil.
4. Une proposition qui intègre les différentes contributions grâce à un nouveau mécanisme d'authentification et de distribution des clés.

Contributions WAP

Un des principaux objectifs de cette thèse est d'enrichir la sécurité d'échanges de la pile WAP en utilisant le protocole TLS. Dans cette optique, nous proposons plusieurs architectures destinées à combler les lacunes de WAP et nous introduisons de nouveaux mécanismes d'authentification à TLS afin d'économiser et de réduire son influence et sa complexité sur les réseaux mobiles. Ces mécanismes rendent la sécurité plus performante et plus efficace en introduisant un tiers de confiance entre le terminal mobile et le serveur d'applications. Cette solution fournit les services de base de la sécurité comme l'authentification de bout en bout et permet ainsi d'éviter d'avoir les messages en clair dans la passerelle WAP. Elle montre les gains par rapport à TLS avec WAP 2.0 en terme d'opérations cryptographiques et de flux de données protocolaires.

Cette partie comporte une autre solution de sécurité basée sur TLS. Bien que TLS utilise des certificats pour l'authentification mutuelle pour des réseaux à grande échelle, notre solution remplace cette méthode en utilisant des clés pré partagées. Pour les raisons citées en amont, l'authentification basée sur une clé pré partagée est plus adéquate, plus fiable et plus performante que celle du certificat. Notre atout pour cette contribution est que CISCO l'utilise dans son protocole EAP-FAST [EAPFast].

Contributions WLAN

Notre troisième contribution constitue une suite logique des travaux effectués dans les deux premières contributions. Elle consiste en l'utilisation des clés partagées et du TLS dans les réseaux 802.11 sans fil. Puisque les contributions précédentes sont des extensions réelles de TLS, nous les adaptons aux réseaux WLAN.

Comme l'anonymat et la protection à long terme de données deviennent de plus en plus indispensables dans les réseaux WLAN, nous proposons une alternative couvrant les exigences de ces réseaux sans l'utilisation des PKIs. Elle se compose de deux phases. La première consiste à ouvrir un tunnel sécurisé entre les deux entités communicantes en établissant une authentification mutuelle basée sur la clé partagée. Les clés dérivées de cette phase permettent aux entités de protéger leurs échanges durant la deuxième phase. La seconde consiste à rafraîchir la clé partagée ainsi que son identificateur en ouvrant une session TLS anonyme. Les informations de cette phase sont encapsulées dans des séquences des attributs TLS. Cette contribution, appelée Double-TLS, répond aux exigences des réseaux WLAN en terme d'anonymat et de protection à long terme de données en réduisant considérablement la charge protocolaire ainsi que le calcul cryptographique en comparaison avec d'autres solutions.

Dans cette partie, nous réalisons une implémentation du protocole EAP-TLS sur une carte à puce. La carte à puce utilisée est dédiée aux réseaux sans fil et répond aux problèmes du stockage des clés sécrètes et privées. Ce type de cartes joue aujourd'hui un rôle important dans l'authentification des clients et des serveurs dans la procédure d'interconnexion entre les réseaux WLAN et 3GPP2.

Une méthode d'authentification intégrant les extensions et les architectures des contributions précédentes

L'utilisation d'une clé partagée pour toutes les sessions ainsi que l'utilisation des clés asymétriques statiques est confrontée à la même problématique. Un intrus peut toujours analyser le trafic et identifier les entités communicantes en interceptant l'identificateur de la clé partagée envoyée en clair pendant la phase de négociation, ou bien en interceptant le certificat dans le cas de l'utilisation de clés asymétriques statiques.

Notre contribution majeure consiste à intégrer les deux types des clés ensemble en utilisant la première pour l'authentification mutuelle et la seconde pour la dérivation des clés de la session. Cette contribution n'utilise pas les PKIs ou les certificats. Elle fournit plusieurs services comme la protection d'identité et la protection à long terme des données. La protection contre les attaques passives et par dictionnaire est donc améliorée. Nous notons ici que la normalisation des futurs standards de l'IETF-TLS et 3GGP2 s'appuie sur notre travail dans l'intégration de la clé partagée et du chiffrement asymétrique avec TLS.

Plan de la thèse

Dans le premier chapitre de ce mémoire, nous étudions le protocole TLS et nous analysons ses avantages et ses inconvénients dans les architectures Internet. Ensuite, nous discutons des attaques possibles sur ce protocole ainsi que sa performance et son coût opérationnel.

La sécurité de la pile WAP apparaît dans le chapitre 2. Dans ce chapitre, nous présentons les différentes spécifications de la pile, les besoins et les problèmes liés à sa sécurité. Nous analysons ensuite les solutions existantes et nous les expérimentons sur des terminaux mobiles.

Dans le chapitre 3, nous étudions les problématiques liées à la sécurité dans les réseaux 802.11 sans fil. Nous examinons les avantages et les inconvénients des solutions existantes. Nous étudions aussi les normes publiées par l'IEEE 802.11 et les standards de l'IETF traitant de la sécurité du réseau WLAN.

Afin de résoudre les problèmes de la pile WAP, nous proposons dans le chapitre 4 deux contributions nous permettant d'assurer l'authentification et le chiffrement de bout en bout. Nous comparons nos solutions à celles définies par le forum WAP et nous montrons leurs avantages en fonction de plusieurs critères.

L'anonymat des échanges, la protection à long terme des données et la performance nous amènent à présenter nos approches pour la sécurisation des échanges pour les réseaux 802.11 sans fil. Dans le chapitre 5, nous traitons deux nouvelles méthodes d'authentification EAP basées sur TLS. Nous poursuivons également les études faites dans le chapitre précédent et nous les adaptons à la sécurisation d'échanges du WLAN. Nous donnons lieu également à une validation à une échelle réelle par une implémentation et un déploiement du protocole EAP-TLS dans un contexte des réseaux sans fil.

L'interconnexion entre les différents types de réseaux nécessite un protocole de sécurité d'échanges plus globale et plus résistant aux différentes attaques connues dans les réseaux filaires et sans fil. Nous présentons dans le chapitre 6 notre dernière contribution proposant l'authentification à clé symétrique, la protection d'identité, la protection à long terme des données, la résistance contre les attaques actives et par dictionnaire. Plusieurs services additionnels tels que le SSO (Single Sign On) peuvent être réalisés en s'appuyant sur notre contribution.

Enfin, le chapitre 7 présente une conclusion générale des travaux de cette thèse ainsi que des perspectives de recherche.

Partie I : Le protocole TLS dans les réseaux sans fil

Chapitre 1

Transport Layer Security

Résumé du contenu

1.1 Introduction

La sécurité des réseaux et la sécurité des échanges sont les deux grands domaines de la sécurité des systèmes d'information. Dans cette thèse, notre recherche porte sur la sécurisation des échanges. Cette sécurisation comporte plusieurs services, à savoir l'authentification mutuelle, la confidentialité et l'intégrité des données.

TLS s'est imposé comme le protocole de la sécurisation des échanges. Deux facteurs ont été à l'origine de cette popularité : le premier est son intégration à tous les navigateurs Web et le deuxième est la conception qui le caractérise. Son adoption par plusieurs communautés de recherche comme IETF-EAP-WG et 3GPP font de lui le protocole phare de l'authentification et de la négociation de clés pour les réseaux sans fil.

Ce chapitre se compose de deux parties principales. La première partie présente un état de l'art du protocole TLS. Elle comprend une étude des mécanismes de base et une présentation de TLS dans différents contextes (fixes et mobiles). La deuxième partie présente une analyse critique du TLS en montrant ses avantages et ses inconvénients. Cette partie se termine par une étude de la charge cryptographique et protocolaire de TLS.

1.2 Une vue générale de TLS

Basé sur SSL qui est développé par Netscape [SSL], TLS est le principal protocole de sécurité au niveau Transport. Diverses applications telles que le commerce électronique sur Internet, les navigateurs WEB et les transactions bancaires supportent la sécurisation des transactions en utilisant le protocole TLS.

TLS fournit un canal transparent. C'est-à-dire qu'il est simple de sécuriser le protocole d'application en insérant TLS entre la couche applicative et celle du réseau. Cependant, TLS nécessite un protocole fiable de transport comme TCP [TCP].

TLS est adaptée à la sécurisation de différents types de trafic et d'échange. Parmi ces types, nous citons le transfert de fichier (FTP [FTP]), la transmission des emails (SMTP [SMTP] et POP [POP]), l'accès à distance (Telnet [Telnet]), l'accès aux répertoires (LDAP [LDAP]), l'accès aux objets (CORBA [CORBA]), etc. Cependant, le nombre d'applications basées sur le transport non fiable comme UDP ne cesse d'augmenter. Les spécifications actuelles de TLS ne peuvent pas être utilisées pour la sécurisation de ce type d'applications. A cet effet, une extension de TLS, appelée Datagram TLS [DTLS], est en cours de construction.

Un point faible commun à toutes ces applications est en l'absence d'un protocole de sécurité comme TLS, les trafics se déroulent en clair sur le réseau et sous les yeux de n'importe quel intermédiaire installé entre la source et la destination. Par exemple, le login et le mot de passe d'un client mail transitent en clair sur le réseau durant la phase d'authentification avec le serveur POP. En plus, le contenu d'un mail est lisible aux hommes du milieu qui peuvent, à chaque instant, changer et intercepter le contenu et modifier les paramètres de sécurité.

1.3 Pourquoi le protocole TLS ?

TLS est un protocole qui peut être appliqué afin de sécuriser les échanges entre deux entités dans le réseau. C'est pour cette raison que le protocole TLS a surpassé son utilisation avec HTTP ([HTTP], [HTTPS]). En effet, plusieurs services assurés par TLS deviennent la base dans les différents types de communication ; notamment :

- TLS fournit la confidentialité des données échangées entre deux applications communicantes. A cet effet, TLS utilise les mécanismes de chiffrement/déchiffrement symétriques comme DES [DES] et RC4 [RC4]. Bien que la cryptographie symétrique nécessite l'utilisation d'une clé secrète partagée entre les deux applications communicantes, le protocole TLS permet, pour chaque session, de générer cette clé de chiffrement/déchiffrement via son sous protocole *Handshake* (Section 4.2).
- TLS assure l'intégrité des données transférées sur le réseau. Ce service est assuré en appliquant la fonction MAC[1] [HMAC] sur chaque portion de donnée échangée entre les deux applications. La fonction MAC utilise les fonctions de hachage telles que MD5 [MD5] et SHA-1 [SHA1] (voir l'Annexe).
- TLS garantit l'authentification des entités communicantes. En effet, les spécifications actuelles de TLS utilisent les certificats numériques et les infrastructures à clé publique afin de créer une chaîne de confiance entre la source et la destination et de les authentifier. Cette

[1] Message Authentication Code

authentification est réalisée en utilisant l'algorithme à chiffrement asymétrique comme RSA [RSA] et DSS [DSS].

- TLS protège les applications contre l'espionnage des données transférées. La protection contre l'interception et le re-jeu est assurée en ajoutant un numéro de séquence à chaque trame de données. Ce numéro de séquence s'intègre aux données avant d'appliquer le chiffrement et le MAC.
- TLS fournit un cadre extensible pour l'intégration de nouvelles méthodes de chiffrement sans introduire de nouveaux protocoles.

1.4 Les sous protocoles de TLS

La connexion TLS est réalisée en deux phases : la phase de la négociation (*Handshake*) et la phase du transfert de données. La première phase permet d'authentifier le serveur et d'établir les clés cryptographiques qui seront utilisées afin de protéger les données à transmettre. Une fois la phase *Handshake* terminée, les deux parties peuvent commencer à échanger les données applicatives.

En plus de *Handshake*, TLS contient trois autres sous protocoles :

- *Record* : met en œuvre les paramètres de sécurité négociés pour protéger les données d'application, les messages de *Handshake*, le message *ChangeCipherSpec* et les messages *Alert*. Il fournit des services de confidentialité et d'intégrité en appliquant le chiffrement et le MAC aux données d'application. Le protocole *Record* prend des messages à transmettre, les fragmente, compresse optionnellement les données, applique un MAC, chiffre et transmet le résultat. A la réception, les données sont déchiffrées, vérifiées, décompressées, et rassemblées puis délivrées à la couche d'application.

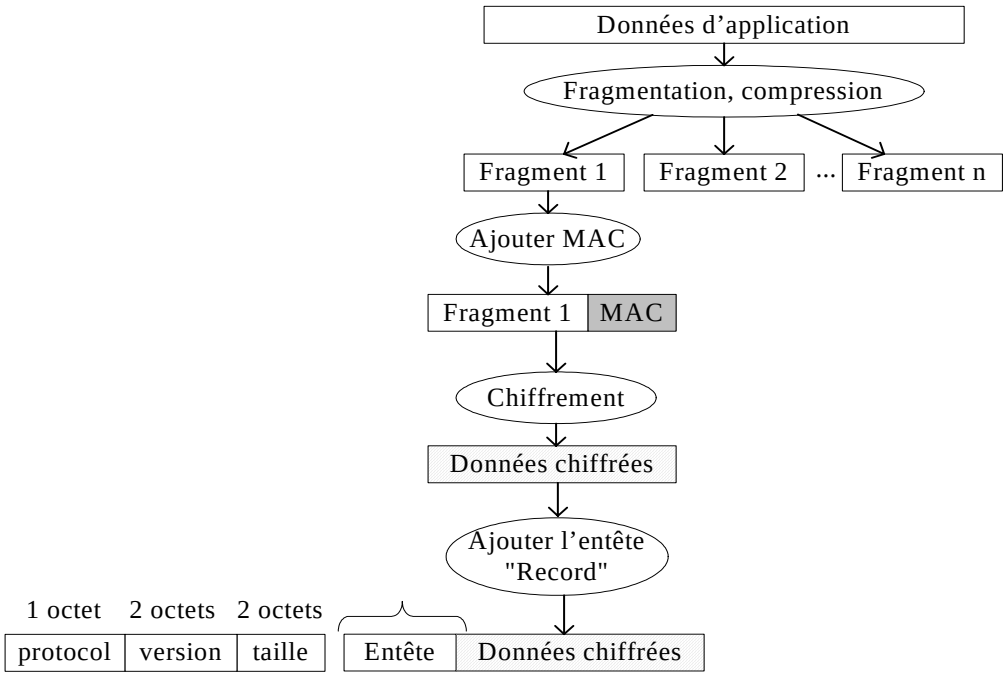

Figure 1.1. Les opérations de la couche Record.

- *ChangeCipherSpec*: message envoyé par le client et le serveur. Il a pour fonction de signaler au *Record* toute modification dans les paramètres de sécurité.
- *Alert* : chargée de signaler les erreurs (fatal ou "warning") durant la vérification des messages ainsi que toute incompatibilité qui pourrait survenir pendant le *Handshake*.

1.4.1 Variables d'états d'une session TLS

Pour chaque session TLS, il y a un ensemble de paramètres à négocier : *session_id*, *certificate*, *compression_method*, *cipher_spec*, *master_secret*, *Is_resumable*[2] [TLS]. Le *cipher_spec* est défini par les éléments suivants: l'algorithme d'échange de clé (RSA, DH, etc.), la catégorie de chiffrement (*stream cipher encryption* ou *block cipher encryption*), l'algorithme de chiffrement (RC2, 3DES, AES, etc.), l'algorithme de hachage (MD5 ou SHA), et *IsExportable* (pour indiquer si le *cipher* est exportable ou non).

Quatre états logiques sont définis par TLS : *current read* et *write*, *pending read* et *write*. Ces états sont configurés et maintenus par la session sécurisée. Tous les *record*s TLS sont traités sous les états *current read* et *write*. Le protocole *Handshake* peut mettre les paramètres de sécurité pour les états en attente (*pending)*. Les états *current* indiquent toujours qu'aucun chiffrement, compression, ou MAC n'est utilisé [TLS 99]. La négociation des paramètres du chiffrement est faite en claire pendant le *Handshake*. Les paramètres de sécurité pour une connexion TLS sont négociés à chaque nouvelle session :

- Deux nombres aléatoires de 32 octets, générés respectivement par le serveur et le client lors de l'établissement d'une session et à chaque nouvelle connexion.
- Serveur et client *write MAC* secrets, employés par les fonctions de hachage pour calculer le MAC. La taille du MAC dépend de l'algorithme de hachage choisi (20 octets pour SHA et 16 octets pour MD5).
- Deux vecteurs d'initialisation pour le chiffrement symétrique en mode CBC[3] [TLS], une pour le serveur et une autre pour le client.
- Deux numéros de séquence, chacun codé sur 8 octets. Ils sont maintenus séparément pour chaque connexion et incrémentés à chaque émission d'un message associé à la connexion. Ce mécanisme sert à détecter les attaques par re-jeu.
- Deux clés pour le chiffrement symétrique des données, une du coté serveur et une autre du coté client.

1.4.2 La phase *Handshake*

Le protocole *Handshake* permet une authentification entre un serveur et un client, de négocier l'algorithme de chiffrement et les clés cryptographiques. Une fois le *Handshake* terminé, les deux pairs partagent une clé secrète, qui est utilisée pour créer un canal sécurisé permettant d'échanger les données d'une manière protégée. Avec *Handshake*, les paramètres cryptographiques d'une session sont réalisés. Cette phase peut se traduire en trois étapes :

---

[2] Indique si de nouvelles connexions peuvent être créées à partir de cette session
[3] Mode de chiffrement par blocs dont le résultat est fonction du chiffrement du bloc précédent.

- Etape 1 : le client et le serveur négocient les paramètres de sécurité (comme les valeurs aléatoires, l'algorithme d'échange de clés, l'algorithme de chiffrement, la fonction de hachage, etc.) afin d'établir une session sécurisée.
- Etape 2 : cette étape consiste à calculer les clés cryptographiques de la session. Ces clés seront utilisées dans les opérations cryptographiques afin de protéger les données durant leurs transfert. Les opérations cryptographiques utilisent les algorithmes cryptographiques négociés durant l'étape 1.
- Etape 3 : durant cette phase, le client et le serveur vérifient l'intégrité de leurs échanges en calculant le condensât de tous les messages transmis et reçus et en chiffrant le résultat avec la clé de chiffrement calculée durant l'étape 2.

Le *Handshake* commence avec le client en envoyant le message *ClientHello* au serveur. Ce message contient une valeur aléatoire utilisée comme une entrée pour le processus de la génération de clés et une liste de *cipher_suite*s supportés par le client. Si le serveur ne supporte aucun des *cipher_suites* envoyées par le client, il arrête le *Handshake*. Autrement, il choisit une *cipher_suite* de la liste envoyée par le client et lui renvoie avec un certificat contient la clé publique du serveur. Le serveur envoie aussi une valeur aléatoire utilisée comme une entrée pour le processus de génération de clés.

Le serveur peut demander le certificat du client. Si le client n'a pas un certificat, il doit répondre par un message contenant "*no certificates*". Le serveur envoie après le message *ServerHelloDone*, indiquant que la phase *hello* du *Handshake* est complète. Le client vérifie le certificat du serveur, ou la chaîne du certificat, par la vérification de l'identité du certificat et la validité de la signature du CA[4]. Le client génère une valeur aléatoire de 48 octets appelée *premaster secret* et le chiffre avec la clé publique du serveur. Cette valeur est envoyée au serveur qui la déchiffre en utilisant sa clé privée pour obtenir le *premaster secret*. Ce secret est utilisé pour générer la même *master_secret* par le client et le serveur. La *master_secret* est utilisée dans la génération des clés symétriques utilisées dans le chiffrement et dans le calcul du MAC.

[4] Certificate Authority

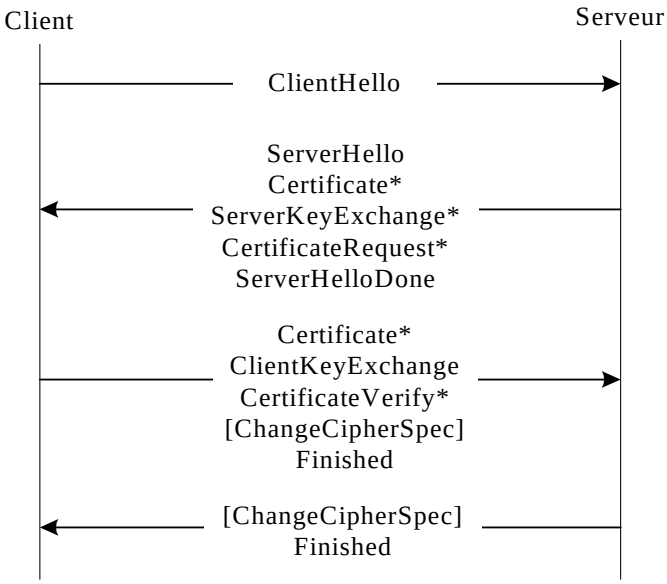

Figure 1.2. La négociation d'une session TLS

Si le client est certifié, il envoie également un message de confirmation explicite : *CertificateVerify*. Ce message inclut un condensât de tous les messages envoyés/reçues depuis le message *ClientHello*. Ce condensât est ensuite chiffré avec la clé privée du client qui envoie le résultat au serveur. Le serveur vérifie le certificat du client ainsi que sa signature appliquée sur le condensât des messages *Handshake*.

À ce moment, le client et le serveur échangent les messages *ChangeCipherSpec* et *Finished*. Le client ou le serveur fait appel au protocole *ChangeCipherSpec* afin de déclencher le chiffrement des échanges selon les choix effectués dans les phases précédentes. Lorsque le client envoie son message *ChangeCipherSpec*, il change de l'état *current* à l'état *pending* en activant le mode de chiffrement. À la réception, le serveur répète la même action mais en activant le mode de déchiffrement. Lorsque le serveur envoie ce message, les opérations précédentes exécutées par le client sont réalisées cette fois-ci par le serveur et vice versa.

Le client envoie immédiatement le message *Finished*. Ce message est un condensât de tous les messages du *Handshake* en employant les attributs cryptographiques qui viennent d'être négociés (sauf les messages *ChangeCipherSpec* et *Alert*). Avec le message *Finished*, on déjoue les attaques de subtilisation en vérifiant l'intégrité de l'ensemble des échanges. Le message *Finished* est calculé comme suit [TLS 99] :

verify_data = PRF(master_secret, finished_label,
MD5(handshake_messages) + SHA-1(handshake_messages)) [0..11];

A la réception du message *Finished*, le serveur reproduit le même condensât de la même manière. Il compare ce condensât avec le message *Finished* précédemment envoyé par le client. Ce message permet de détecter si un intrus a intercepté et modifié les messages *Handshake*. Le serveur envoie ensuite les messages *ChangeCipherSpec* et *Finished*. Le message *Finished* est généré de la même façon utilisée par le client. Après avoir émis/reçu le message *Finished*, le client et le serveur envoient les messages d'application chiffrés avec les paramètres de sécurité négociés.

1.4.2.1 La reprise d'une session établie

Une nouvelle session sécurisée peut être établie sur la base d'une session sécurisée existante. Ce qui nous permet de ne plus ré exécuter la négociation complète ainsi que ses calculs cryptographiques.

TLS identifie la session par le paramètre *session_id*. Lorsque le client veut reprendre une session, il envoie le message *ClientHello* indiquant l'identificateur de la session sécurisée à reprendre. Le serveur de son côté vérifie s'il existe une correspondance dans son cache. Deux cas possibles sont à envisager :

- S'il trouve la correspondance et s'il accepte de rétablir une connexion sécurisée pour la session indiquée, le serveur enverra un message *ServerHello* qui portera l'identificateur de cette session trouvée en cache. A ce moment, le serveur envoie également le message *ChangeCipherSpec* et passer directement au message *Finished* auquel le client devra répondre avec ses propres messages *ChangeCipherSpec* et *Finished*.
- Si l'identificateur de la session n'est pas trouvé en cache, le serveur crée un nouvel identificateur : le client et le serveur exécutent une négociation (*Handshake*) complète.

1.4.2.2 Dérivation de clés

Durant la phase *Handshake*, le client chiffre la clé *premaster secret* par la clé publique du serveur. Cette clé sera utilisée pour calculer et dériver toutes les clés de la session. La dérivation de clés est établie en utilisant la fonction PRF[5] [TLS].

1.4.2.2.1 La fonction PRF

La fonction PRF est une fonction cryptographique appliquée sur trois paramètres : un secret, un label ("string") et une valeur aléatoire ("seed"). Sa représentation peut prendre la forme : PRF(secret, label, seed).

Le string *label* nous permet d'utiliser la fonction PRF pour générer différentes clés en utilisant le même secret. Par exemples, PRF(secret, label 1, seed) et PRF(secret, label 2, seed) sont deux résultats différents.

PRF génère une valeur de longueur arbitraire. Avec TLS, la notation PRF(a, b, c)[0..15] se désigne sur les 16 premiers octets de la sortie du PRF appliquée sur les variables a, b et c.

Le principe du PRF est le suivant : le secret est divisé en deux parties égales, S1 et S2. Si la taille du secret est impaire, le plus faible octet du S1 ($16^{ème}$ octet du secret) et le plus fort octet de S2 sont les mêmes. Chaque partie est utilisée comme un secret pour la fonction P_hash (P_MD5 ou P_SHA-1) qui est basée sur HMAC [HMAC]. La fonction P_hash est calculée comme il est montré dans la figure suivante. Cette fonction est répétée en fonction de la taille de la clé souhaitée. Par exemple, si nous voulons une sortie dont la taille est égale à 80 octets, nous répétons P_MD5 cinq fois et quatre fois pour la fonction P_SHA-1.

[5] Pseudo Random Function

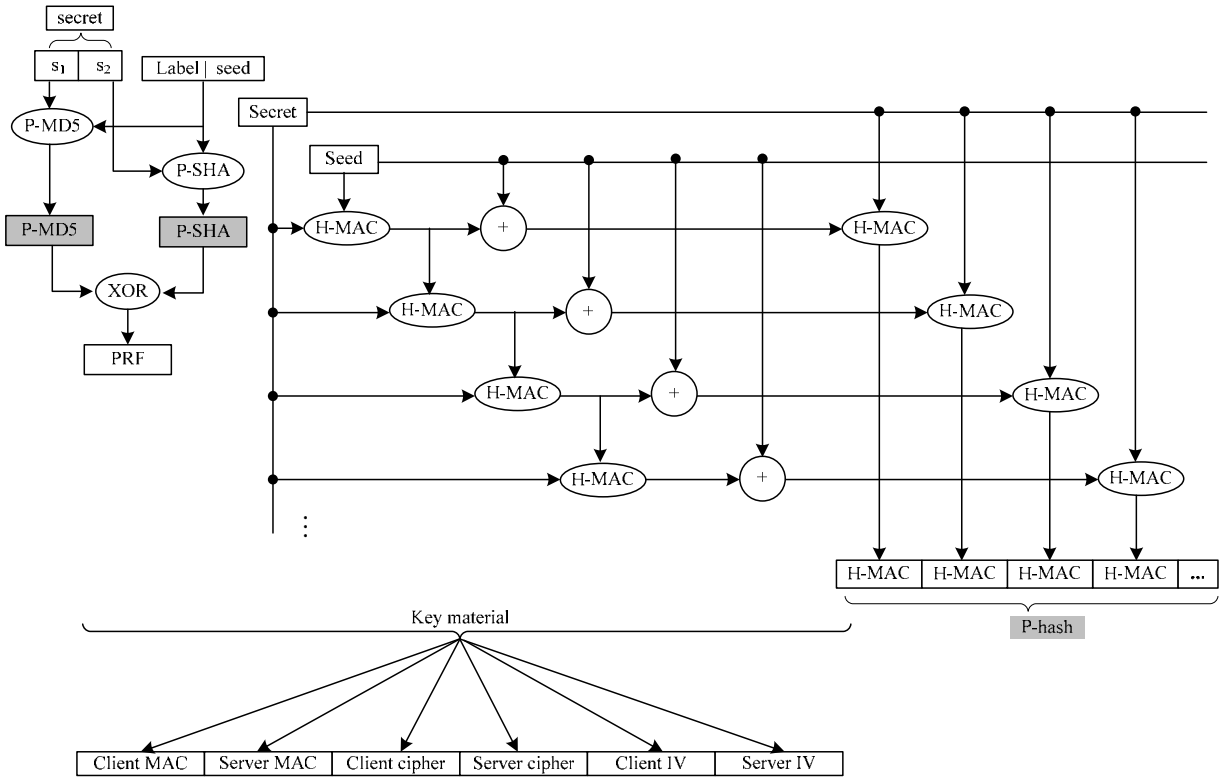

Figure 1.3. La fonction PRF

1.5 Utilisation de TLS dans différents contextes

Le premier avantage de TLS est sa capacité à fournir un canal sécurisé transparent. Cependant, TLS nécessite un protocole de transport fiable tel que TCP. Ainsi, il ne peut pas être utilisé pour sécuriser le trafic sans contrôle de la fiabilité et de type *datagram*. En effet, la non fiabilité de Transport a une influence directe sur la conception de TLS. TLS ne permet pas un déchiffrement indépendant. C'est à dire, nous ne pouvons pas déchiffrer le *record* N si le *record* N-1 est perdu. De l'autre côté, les messages *Handshake* sont supposés être bien délivrés de la source vers la destination et vice versa.

1.5.1 TLS avec UDP

Le nombre des applications conçues en utilisant un protocole sans contrôle de la fiabilité comme UDP [UDP] ne cesse d'augmenter. Avec l'absence d'un protocole de sécurité bien spécifié, ces applications sont confrontées à de vrais problèmes de sécurité. Malgré l'existence d'IPSec, ce dernier agit directement sur la couche IP du modèle OSI et il ne répond pas suffisamment aux besoins de différentes applications. D'autres protocoles de sécurité sont disponibles au niveau de la couche applicative mais malheureusement ce type de solutions nécessite un grand effort en terme de conception et de gestion.

Datagram TLS [DTLS] est une solution alternative basée sur TLS. Cette solution intègre TLS avec UDP. A cet effet, une sémantique est définie afin de garantir la livraison de messages TLS au-dessus de UDP. Bien que cette sémantique assure la fiabilité des messages protocolaires du TLS grâce au numéro de séquence de la couche *Record*, elle ne contrôle pas la perte ou la duplication du trafic (par exemple, les applications de diffusion de média ou celle de la voix sur

IP). Autrement dit, les comportements de telles applications restent inchangés quand DTLS est utilisé.

1.5.2 TLS avec EAP

Un autre contexte dont lequel TLS joue un rôle important dans la sécurité des échanges est le contexte du protocole PPP[6] [PPP] (il sera détaillé dans le chapitre 4). Ce protocole fournit une méthode standard pour transporter des datagrammes multi protocoles sur des liaisons point à point. Il est reconnu pour sa fiabilité. Ce protocole est renforcé par un protocole d'authentification nommé EAP [EAP]. Ce dernier est un protocole d'authentification générique qui supporte plusieurs mécanismes d'authentification (MD5 [MD5], OTP [EAP], etc.). L'extensibilité du EAP permet d'encapsuler tout mécanisme d'authentification à l'intérieur du message EAP. L'intégration de TLS avec EAP est devenue une solution pour l'authentification et la distribution des clés dans les réseaux qui utilisent EAP dans leurs piles protocolaires ; notamment la série de réseaux IEEE 802 ([802.11-Part1], [802.11-Part2]).

1.5.3 TLS et l'interconnexion entre 802.11 sans fil et 3GPP

Les points forts de TLS lui permettent de s'intégrer facilement dans d'autres environnements de communication. Par exemple, les normes de communications mobiles 3GPP [3GPP], 3GPP2 [3GPP2] et 802.11 sans fil s'intéressent énormément à son utilisation dans les authentifications mutuelles et pour l'établissement d'une session sécurisée plus résistante et plus fiable. Cette utilisation n'est pas encore spécifiée pour le 3GPP. Pour cela, plusieurs contributions ont été proposées au sein du groupe de TLS à l'IETF. Nous détaillons nos deux contributions dans les chapitres V et VI.

1.6 Avantages et limitations du protocole TLS

L'avantage principal de TLS est sa capacité à assurer l'intégrité et la confidentialité de données déroulant sur un réseau insécurisé. Il devient le facto standard pour l'authentification mutuelle et pour la distribution de clés.

TLS cependant affronte plusieurs difficultés ; notamment :

- Possibilité d'accepter ou d'utiliser des certificats révoqués. Cette problématique est liée aux navigateurs (Netscape et Internet Explorer). Dans leur utilisation actuelle, les navigateurs ne consultent pas la liste de révocation de certificats.
- Les messages TLS ne s'accommodent pas de l'utilisation de serveurs *proxy* classiques (cache et réplication) parce que TLS a été conçu pour faire face aux attaques MitM, et que le *proxy* va être considéré comme un intrus. Pour qu'un serveur *proxy* puisse gérer du trafic TLS, il doit supporter le protocole SOCKS [SOCKS] ou un protocole spécial de *tunneling* TLS. Netscape Proxy Server par exemple supporte les deux [SFA].
- Durant la phase *Handshake* du TLS, le certificat du client est envoyé en clair.
- Les spécifications actuelles de TLS ne définissent pas les moyens pour négocier d'autres types de certificats (certificat d'attributs par exemple). Ainsi l'authentification et l'autorisation sont limitées à l'utilisation d'un certificat X.509.

[6] Point-to-Point Protocol

1.6.1 Les attaques

Quelques attaques sur le protocole TLS sont théoriquement possibles. Nous pouvons diviser ces attaques en deux parties : les attaques sur les algorithmes cryptographiques utilisés par TLS et les attaques sur les réseaux et les architectures. Cette deuxième partie est détaillée en Annexe A.

1.6.1.1 Les attaques sur les algorithmes cryptographiques utilisés par TLS

TLS utilise une liste de *cipher_suites* contenant un algorithme à clé publique, un autre à clé symétrique et une fonction de hachage. La complexité de tel *cipher_suite* dépend de chacun de ces algorithmes. Par conséquent, si un intrus arrive à réaliser un cryptanalyse contre un algorithme donné, la session TLS devient vulnérable à cette attaque.

1.6.1.1.1 Complexité des algorithmes d'échange de clés

TLS utilise RSA [RSA] et Diffie-Hellman [DH] comme algorithme d'échange de clés. La résistance aux attaques et la complexité de chacun de ces deux algorithmes dépend, entre autres, de la taille de la clés. Un autre facteur de complexité pour ces deux algorithmes est lié à leur conception mathématique.

La complexité de l'algorithme RSA est basée sur le problème de factorisation d'un entier N en un produit de deux entiers premiers. Le mécanisme de factorisation le plus connu est celui de GNFS. Il peut factoriser un entier n avec une complexité d'ordre $\exp[c(\log n)^{1/3}(\log(\log(n)))^{2/3}]$; $c < 2$. En décembre 2003, RSA-576 (la taille en bits de N est 576) a été factorisé [GPNRR].

Diffie-Hellman est basé sur le logarithme discret. De ce fait, il est plus efficace que RSA. Mathématiquement, le problème du logarithme discret est considéré plus complexe que celui de la factorisation d'un entier en produit des entiers premiers.

1.6.1.1.2 Complexité des algorithmes symétriques

TLS utilise plusieurs algorithmes symétriques comme DES [DES], 3DES [3DES] et RC4 [RC4]. Plusieurs attaques sont connues contre ce type d'algorithmes ; notamment l'attaque par force brute.

L'algorithme DES utilise des clés de 56 bits. Nous avons donc 2^56 clés possibles pour chiffrer un message donné. En 1998, EFF[7] a construit un *crack* qui permet de trouver la clé de chiffrement DES en 56 heures sur un machine qui coûte 250.000 $ [EFF]. D'autres attaques sur DES requièrent 2^47 *chosen plaitexts*[8] [DESHack] ou 2^43 *known plaintexts*[9] [Matsui]. Aujourd'hui, le NIST[10] recommande l'utilisation du Triple-DES.

La complexité de l'algorithme RC4 est équivalente à $O(n^{1/2})$ [RC4C] ; n est la taille de la clé.

1.6.1.1.3 Complexité des fonctions de hachage

Récemment à la conférence CRYPTO 2004, une attaque par collision sur la fonction MD5 a été publiée [MD5C]. Les auteurs définissent une méthode générique afin de trouver des collision

[7] Electronic Frontier Foundation
[8] Attaque permettant à un cryptanalyste ayant accès à la fonction de déchiffrement, de choisir les messages afin de découvrir la clé à partir du résultat obtenu du déchiffrement de ces messages.
[9] Attaque permettant à un cryptanalyste ayant à la fois accès au texte clair et au texte chiffré de découvrir la clé de chiffrement.
[10] National Institute of Standards and Technology

avec MD5 et ils ont confirmé qu'ils ont réalisé une attaque contre SHA-0 avec une complexité égale à 2^40.

L'attaque par collision sur MD5 a une influence majeure sur TLS (mais aussi IPSec et les autres protocoles de sécurité qui utilisent MD5) dans différents contextes. Les fonctions de hachage sont utilisées dans les signatures numériques ; notamment dans le contexte de non répudiation. Elles sont encore utilisées dans les primitives d'authentification de TLS (PKIs et certificats).

Supposons qu'Alice veut prendre l'identité de Bob. Alice génère une requête de certificat simplifiée par R[11] = "Alice.com, Key = X" et l'envoie au CA. Ensuite, Alice utilise l'attaque par collision afin de trouver R' tel que R' contient l'identité de Bob (simplifiée par R' = "Bob.com, Key = Y") et que MD5(R) = MD5(R'). Lorsque CA signe R, il signe implicitement R'. Par conséquent, Alice peut utiliser le certificat correspond à R' et donc se présenter comme Bob.

En pratique, le CA spécifie un numéro de séquence pour chaque certificat délivré avant que le certificat ne soit signé. Malgré que ce numéro de séquence soit incrémenté d'un après chaque nouvelle requête d'un certificat et que Alice puisse deviner le prochain numéro délivré par le CA, le numéro de séquence réduit la chance de réaliser la démarche en amont. Cependant, MD5 ne doit plus être utilisée, au moins dans le contexte des PKIs.

1.7 Performances du protocole TLS

La performance de TLS dépend de plusieurs facteurs :

- L'algorithme d'échange de clés, RSA ou DH.
- La taille de la clé de chiffrement/déchiffrement.
- L'algorithme de chiffrement symétrique, RC4, 3DES, AES, etc.
- La taille du *Record*.
- La capacité du microprocesseur client/serveur, le débit du réseau et la vitesse du traitement des données *Handshake*.

Comme nous l'avons déjà cité, TLS utilise une combinaison d'algorithmes à clé publique, d'algorithme de chiffrement symétrique et d'une fonction de hachage.

L'algorithme d'échange de clés le plus utilisé est RSA. Cet algorithme est coûteux s'il est utilisé pour chiffrer des blocs de grande taille. Dans TLS, nous utilisons RSA pour échanger une clé sur 48 octets. Cela nécessite deux opérations RSA ; chiffrement par le client et déchiffrement par le serveur.

La performance de RSA dépend de plusieurs facteurs ; notamment la taille la clé (512 bits, 1024 bits, etc.). Une estimation du temps d'exécution du chiffrement RSA (Te) et de déchiffrement (Td) peut être présentée comme suit [HSRSA] :

Chiffrement RSA :

Te = (w-1)(P+A) + (s^2+s)wA + (ke-1)[(3s^2+s)P/2 + (7s^2+s)A/2] + (he-1)[(2s^2+s)P + (9s^2+3s)A]

Déchiffrement RSA :

Td = (w-1)(P+A) + (s^2+s)wA +(kd-1)[(3s^2+s)P/2 + (7s^2+s)A/2] + (hd-1)[(2s^2+s)P +(9s^2+3s)A/2]

[11] Contient de l'information appartenant à Alice ainsi sa clé publique X

- *he*: Hamming weight[12] du *e*.
- *ke*: la taille de *e* en bits et *kd* la taille de *d* en bits.
- *e* : l'*exponent* publique de l'algorithme RSA.
- *S* : la taille du modulo n en *words* (la taille de la clé RSA est la multiplication par 16 de la valeur de *S*).
- *A* : le nombre de cycles nécessaires pour faire l'addition de deux entiers *single-precision* et *P* le nombre de cycles pour leur multiplication.
- *w* est le *wordsize* du PC utilisé.

Si nous utilisons un microprocesseur avec une vitesse de 10 MHz, w=16, A = P = 3, le nombre de cycles d'exécution des opérations RSA peut dépasser quelques milliards comme il est montré dans la figure suivante.

La courbe Tdc correspond au déchiffrement RSA en utilisant le théorème *Chinese Remainder* [CRT]. Une implémentation de ce théorème est capable de réduire la charge d'exécution du déchiffrement RSA en facteur de 4 [CRT].

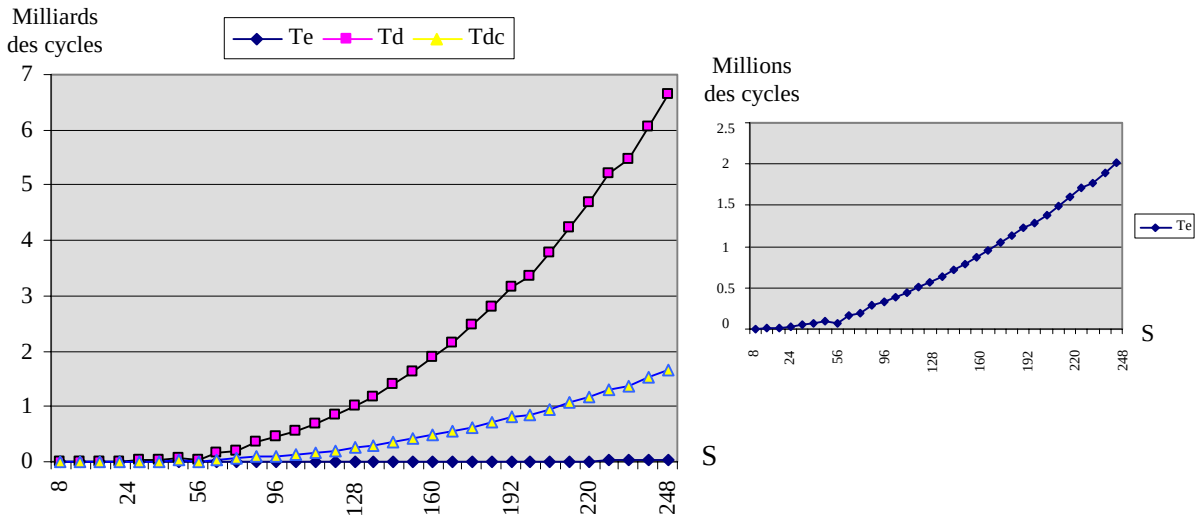

Figure 1.4. Nombre de cycles nécessaires pour les opérations RSA

Pour le chiffrement de données d'applications, TLS utilise des algorithmes de chiffrement symétrique comme DES et Triple DES. Ces algorithmes ont des coûts plus réduits que RSA. Le tableau suivant montre le nombre des différentes opérations nécessaires par le chiffrement DES et 3DES [NESSIE].

Algorithme	Block size (bits)	Word size (bits)	Key size (bits)	Subkey size (bits)	Table loockup/Table size (bitsxbits)	XOR, ADD (bit size)
DES	64	32	56	768	128/8(6x4), 8/8(8x32), 64/11(8x48), 16/16(8x64), 0/8(8x56)	6(32bit), 64(48bit), 14(64bit)

[12] Le nombre de bits à un dans une séquence de bits. Ici, c'est le nombre de bits "1" dans la présentation binaire de *he*.

3DES	64	32	56	-	384/8(6x4), 8/8(8x32), 192/11(8x48), 16/16(8x64), 0/8(8x56)	6(32bit), 192(48bit), 14(64bit)

Tableau 1.1. Les opérations nécessaires pour le chiffrement/déchiffrement DES et 3DES.

TLS utilise les fonctions de hachage dans le calcul de MAC, HMAC et PRF. Ces fonctions sont en générale rapides et nécessitent moins de calcul mathématique. Par exemple, la fonction SHA-1 a besoin de 2984 opérations logiques pour son exécution [NESSIE]. Le tableau suivant montre ces différentes opérations.

Block size	Word size	Rotation (32 bits)	XOR (32 bits)	ADD (32 bits)	AND /OR /NOT (32 bits)
512	32	224	312	325	100

Tableau 1.2. Les opérations basiques de SHA-1.

En résumé, la charge d'une session TLS dépend de la *cipher_suite* choisie par le client et du serveur, de la taille des clés et la chaîne de certification, et du temps de traitement des messages protocolaires TLS. Dans la figure suivante, nous montrons les pourcentages de cette charge dans le cas d'une authentification mutuelle avec l'algorithme RSA à 1024 bits.

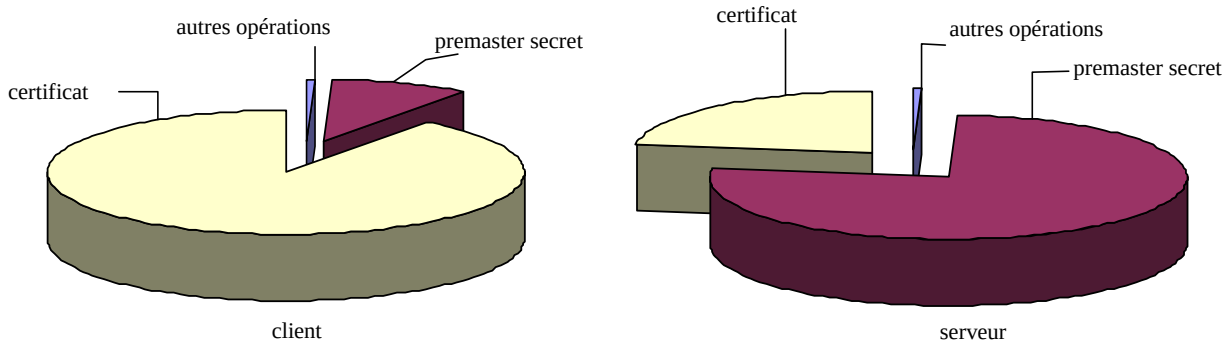

Figure 1.5. Les pourcentages du coût cryptographique d'une session TLS.

1.8 Conclusion

TLS est de plus en plus le protocole phare de la sécurisation des échanges dans le monde Internet et sans fil. En effet, une version future de TLS saura répondre aux différents inconvénients et limitations lorsque TLS sera appliqué comme protocole d'authentification et de confidentialité.

Plusieurs attaques sont ainsi destinées, entre autres, aux mécanismes de chiffrement utilisés par TLS. Ces attaques restent théoriques et elles dépendent des mauvaises implémentations de TLS. Peu d'entre elles s'adressent au noyau de TLS ([BLEI], [CBCATT]). Cependant, des contre-mesures natives à TLS peuvent lutter contre ces attaques [TLS1.1].

La charge cryptographique et protocolaire du TLS peut se réduire en introduisant d'autres mécanismes d'authentification que les certificats et les infrastructures à clé publique et en définissant d'autres architectures. Notons ici que le stockage des clés secrètes, des clés privées et des certificats des tiers de confiance (CA) reste une mission importante pour les systèmes des clients et des serveurs. Cependant, l'utilisation de la carte à puce peut réduire au minimum cette problématique. Nous citons ici que les clients doivent emprunter suffisamment d'attention

lorsqu'ils décident d'accepter ou d'installer un CA particulier dans leur boite de tiers de confiance. En effet, une CA malhonnête peut endommager entièrement la sécurité du système.

Nous avons montré que TLS est largement utilisé pour la sécurisation de différents types de trafic sur les réseaux fixes. Son extensibilité et ses différents intérêts font de lui le standard le plus utilisé pour l'authentification et la distribution de clés.

Etant donné que les réseaux sans fil sont une extension des réseaux fixes, TLS est donc adapté pour la sécurisation des échanges mobiles (WTLS, EAP-TLS, EAP-TTLS, EAP-FAST, PEAP, etc.). Cependant, la sécurité de bout en bout fournie par TLS est confrontée à des problématiques majeures liées aux architectures sans fil. En effet, la plupart de ces réseaux utilise de passerelles, entre le fixe et le sans fil, qui génèrent par conséquent l'impossibilité d'identifier le client auprès du serveur. Parmi les architectures concernées, nous étudions les architectures WAP, détaillées dans le chapitre suivant.

D'autres problèmes sont liés à la conception du réseau sans fil ; notamment la partition du support de transmission, la mobilité et le besoin de protéger les informations liées aux clients et au réseau. L'application du TLS tel qu'il est actuellement spécifié, ne répond pas suffisamment à ces problèmes. Dans le chapitre 4, nous analysons et montrons les inconvénients et l'insuffisance de cette application dans les architectures 802.11 sans fil.

Chapitre 2

TLS dans la pile protocolaire WAP

Résumé du contenu

2.1 Introduction

La mise en place des architectures Internet destinées à des terminaux mobiles et le déploiement d'interfaces radio mieux adaptées aux communications de données que les réseaux GSM (GPRS, UMTS, WLAN) sont les facteurs les plus importants du succès de l'Internet mobile. Le choix initial du "tout ouvert" sur l'Internet laisse une grande place à l'insécurité. De grands projets portent sur l'intégration dans les terminaux et les serveurs de nouvelles méthodes, protocoles et de mécanismes de sécurité dont le but est d'assurer la sécurité indispensable pour des applications telles que les transactions bancaires et les opérations de vote et de réservation.

Dans ce chapitre, nous étudions et nous analysons la sécurité dans les réseaux mobiles en utilisant la pile protocolaire WAP [WAPArch]. Plus particulièrement, nous présentons une étude analytique du protocole WTLS [WTLS], une dérivation du TLS adaptée aux contraintes des réseaux à bande étroite et nous montrons les failles majeures de sécurité liées à cette adaptation.

Ce chapitre montre également les besoins d'introduire d'autres mécanismes plus fiables et mieux adaptés pour les architectures WAP. Il se termine par une étude comparative (performances, charge protocolaire, mise en œuvre, etc.) entre plusieurs solutions de sécurité WAP basées sur TLS.

2.2 La pile WAP

La solution WAP est le fruit du travail du Forum WAP qui étudie la manière de favoriser le développement des services sur les réseaux sans fils. Elle est constituée d'une série de spécifications ayant le but de standardiser un environnement d'applications et de protocoles de communication permettant l'accès des terminaux sans fils à Internet. Basé sur les technologies Web, le WAP a été créé en s'appuyant sur une architecture client/serveur.

L'architecture WAP comporte deux parties : la pile des protocoles et l'environnement de développement des applications. Elle présente un pas vers la convergence entre le monde Internet et le monde des données sur mobiles.

L'utilisation des données sur des terminaux et réseaux mobiles est plus contraignante que celle des données sur des équipements et réseaux fixes. Les contraintes se présentent au niveau des terminaux et réseaux.

La pile WAP permet d'adapter les formats d'Internet aux contraintes des téléphones portables telles que : le débit, la taille de l'écran, la vitesse de connexion qui est relativement lente, la taille de la mémoire à disposition, etc.

Ces limitations font que l'interface utilisateur d'un terminal mobile est radicalement différente de celle d'un terminal fixe.

Il en est de même pour la comparaison entre réseaux fixes et mobiles. Les réseaux mobiles sont plus lents pour l'accès à des données que les réseaux fixes. Les contraintes des réseaux mobiles par rapport aux réseaux fixes sont :

- le débit est plus faible, le temps de réponse est plus long,
- les connexions sont moins stables,
- la ressource radio est moins disponible que la ressource filaire,

Le Forum WAP a créé l'architecture WAP en tenant compte de toutes ces contraintes des terminaux et des réseaux mobiles. Il s'appuie, pour cela, sur les points suivants :

- utiliser autant que possible les standards Internet et mobiles,
- créer une architecture facilement extensible,
- supporter le plus grand nombre possible des réseaux mobiles,
- optimiser l'architecture pour des réseaux à bas débit et à long temps de réponse,
- optimiser l'architecture pour les contraintes des terminaux mobiles,
- fournir un contexte favorable pour développer la sécurité,
- permettre l'utilisation des fonctionnalités locales du terminal (par exemple signaler un appel),
- faciliter l'intégration de solutions de plusieurs acteurs,
- favoriser le fonctionnement entre les produits de différents fournisseurs,
- fournir une solution pour les applications téléphoniques.

2.2.1 Les différents composants de l'architecture WAP

WAP s'appuie sur une architecture client/serveur et une pile de protocoles, il permet le transfert d'informations du monde Internet et Intranet sur le terminal mobile.

Les architectures WAP reposent sur trois principales entités : le serveur d'application, la passerelle et le client.

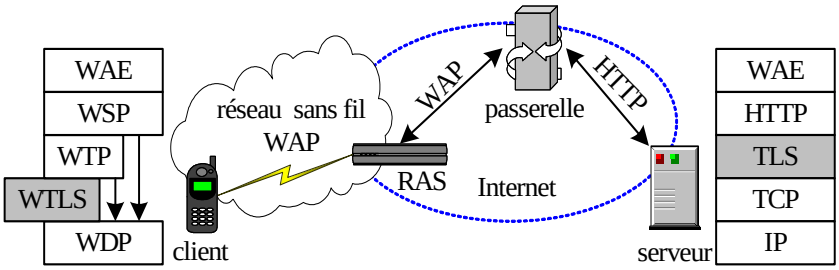

Figure 2.1. L'architecture WAP/Internet

2.2.1.1 Le serveur d'application

Dans les architectures WAP, le serveur permet de traiter les requêtes du client et de générer les réponses correspondantes. Une fois les réponses traduites en WML[13], elles sont envoyées à la passerelle qui se charge de les transmettre au terminal mobile par l'intermédiaire du réseau sans fil.

2.2.1.2 Client WAP

La figure ci-dessous illustre l'architecture WAP d'un dispositif radio conforme. Le Forum WAP spécifie l'architecture fondamentale du terminal tandis que l'interface utilisateur est laissée au constructeur.

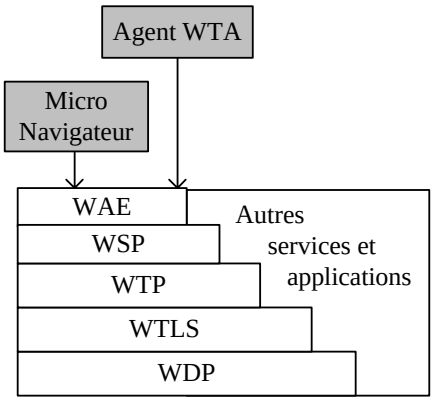

Figure 2.2. Le client WAP

Les terminaux WAP sont équipés d'une pile de protocoles, d'un micro navigateur et d'un environnement de téléphonie.

[13] Wireless Mark-up Language, langage d'encodage des données qui sont affichées par le navigateur WAP. Il est fondé sur le langage XML et est optimisé pour utilisation sur connexion radio et sur terminaux à faible puissance.

- L'implémentation de la pile WAP est nécessaire pour faciliter le transfert bidirectionnel.
- Le micro navigateur est similaire au navigateur Web standard. Il est adapté au terminal sans fil, à petit écran et à puissance de calcul réduite.
- L'environnement de téléphonie WTA[14] [WTA] est un ensemble d'interfaces qui permettent de réaliser des applications téléphoniques. Ces interfaces fournissent un accès aux fonctions de téléphonie telles que la composition d'un numéro et la mise à jour du répertoire.

2.2.1.3 La passerelle WAP

Basée sur la technologie du serveur délégué "proxy", la passerelle WAP contient les fonctionnalités suivantes :

- Conversion de protocoles : la passerelle assure la conversion entre les piles de protocoles WAP et TCP/IP. Elle convertit les requêtes d'une session WSP[15] [WSP] en requêtes HTTP[16] et vice versa. Elle convertit aussi les données du format HTML[17] en format WML et réciproquement.
- Codage et décodage du contenu : elle assure la translation et la traduction des contenus Internet en contenus adaptés utilisables dans le monde des mobiles.

La principale caractéristique de la passerelle WAP est la coexistence des piles de protocoles WAP et Internet (Figure ci-dessous). Les fonctions de la passerelle WAP introduisent des services non supportés par le Forum WAP afin d'améliorer l'installation et l'administration de cette dernière. Citons pour l'exemple, la gestion de la plate-forme de la passerelle et la génération des données pour facturation qui ne sont pas intégrée par le Forum WAP en tant que composant essentiel de la passerelle.

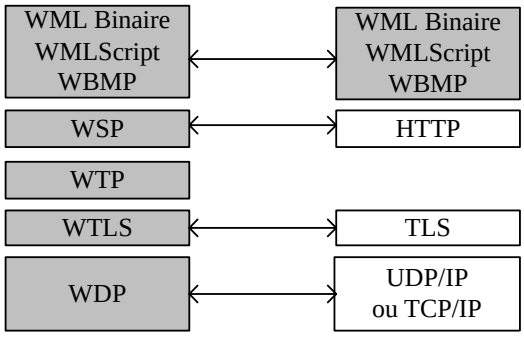

Figure 2.3. La Passerelle WAP

Dans la majorité des cas, la passerelle WAP est installée chez l'opérateur dans le domaine du réseau mobile, cependant, ce n'est pas l'unique cas d'installation (Tableau ci-dessous). Elle peut être intégrée avec le système d'information des entreprises, des fournisseurs d'accès ou des serveurs d'applications. Elle peut être aussi intégrée en un seul serveur avec le serveur d'origine et le filtre HTML.

Cas	Réseau mobile	Réseau fixe

[14] Wireless Telephony Application
[15] Wireless Session Protocol
[16] HyperText Transport Protocol
[17] HyperText Mark-up Protocol

1		Passerelle, filtre et serveur d'origine
2	Passerelle	Filtre et serveur d'origine
3	Passerelle et Filtre	Serveur d'origine

Tableau 2.1. Différents cas d'installation de la passerelle.

L'utilisation de la technologie "proxy" permet aux pages WAP d'exister sur des serveurs d'applications WWW[18]. Accéder à des données WAP sans passer par la passerelle est également possible comme l'illustre la figure ci-dessous. Par exemple, le serveur WTA est accessible sans passer par la passerelle. Cependant, dans les différents cas d'installation de la passerelle, le client WAP ne peut communiquer que du WML binaire.

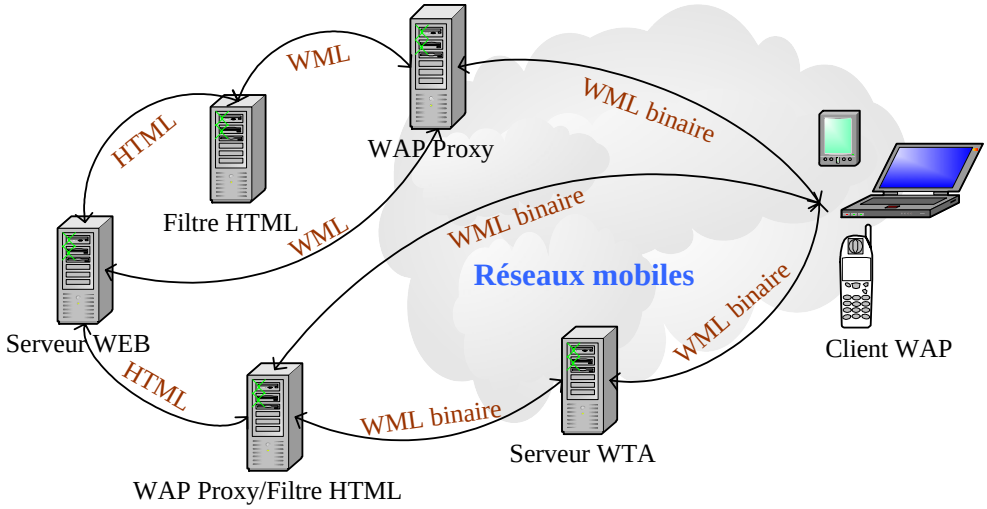

Figure 2.4. Différents exemples d'implémentation

2.2.2 Cycle de vie d'une session WAP

Dans l'architecture WAP, un terminal mobile WAP désirant obtenir des informations via un service WAP doit se connecter à une passerelle WAP à l'aide d'un numéro de téléphone. Les transactions émises par ce terminal sont traduites par la passerelle en requêtes IP envoyées sur le réseau fixe, vers le serveur d'application.

Une fois la connexion établie entre la passerelle et le terminal compatible WAP (mobile ou personnel), ce dernier envoie une requête codée pour accéder à une URL[19]. La passerelle se charge alors de la décoder et de l'envoyer sur le réseau filaire Internet. La passerelle reçoit ensuite la réponse et l'envoie codée au terminal.

En plus de son rôle d'intermédiaire entre réseau mobile et réseau fixe, la passerelle convertit les réponses en provenance du serveur d'application en format binaire plus compact pour les acheminer sur les réseaux mobiles. Nous notons que le modèle de programmation WAP est similaire au modèle Internet, il est conçu sur une approche client/serveur.

[18] World Wide Web
[19] Universal Resource Locator

Le contenu et les protocoles de communication WAP tiennent compte des caractéristiques des réseaux mobiles (débit et temps de réponse) ainsi que des caractéristiques des terminaux (périphériques, capacité d'affichage et capacité de traitement).

2.3 La sécurité dans WAP

Le groupe chargé de la sécurité au sein du Forum WAP travaille sur une solution permettant d'assurer la sécurité de bout en bout (extension au-delà de la passerelle) en se basant sur la solution de base proposée par la pile WAP. Le service de sécurité de la pile de protocoles WAP est assuré par le protocole WTLS[20] [WTLS]. Ce dernier a été défini en adaptant le protocole TLS aux contraintes des réseaux à bande étroite.

La sécurité dans WAP est prise en compte uniquement dans la version 1.1 grâce au protocole WTLS. Opérant au-dessus du protocole de transport, WTLS fournit aux couches de la pile WAP un service de transport sécurisé et offre une interface pour gérer les connexions sécurisées.

Le protocole WTLS offre l'intégrité des données, l'authentification et la confidentialité. Les fonctionnalités de WTLS sont, comme pour presque tous les protocoles de la pile WAP, similaires à celles proposées par le protocole TLS 1.0 du monde Internet.

Le concept de sécurité dans l'architecture WAP consiste à utiliser le protocole WTLS entre la passerelle et le terminal et à utiliser TLS entre la passerelle et le serveur Web (Figure 2.5).

Cependant, WTLS n'est pas compatible directement avec TLS ([Bad10], [Bad14]). La passerelle assure donc la conversion de WTLS en TLS et vice et versa. Comme les données se trouvent à un moment donné en clair sur la passerelle, il est évident que les conditions de sécurité physiques et logicielles de ce serveur sont très particulières.

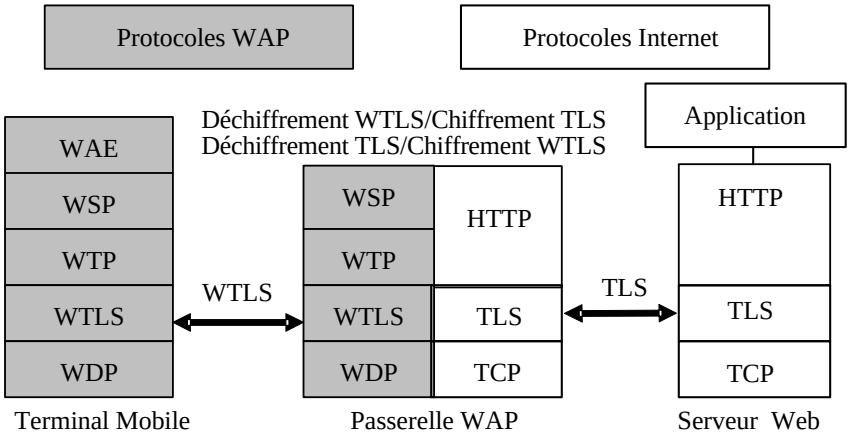

Figure 2.5. Echange sécurisé dans WAP

La solution de sécurité proposée, par la version 1.1 de WAP avec le protocole WTLS, n'offre pas la non répudiation et l'authentification mutuelle entre le client et le serveur. La sécurité de bout en bout n'est donc pas assurée dans cette architecture. Le véritable problème dans cette dernière est l'impossibilité d'identifier le client.

[20] Wireless TLS

La version 1.2 des spécifications WAP traite ce problème par la spécification du module WIM[21] embarqué sur les terminaux. L'interaction avec le WIM est assurée par la bibliothèque Crypto ajoutée à *WMLScript*[22] ([WMLSC], [WMLS]). Dans cette bibliothèque, deux fonctions sont disponibles : *SignText* et *EncryptText*. La première nous permet de signer une portion de données texte. Cette opération peut être faite via l'utilisation du module WIM et permet d'authentifier le client. La deuxième permet de chiffrer les données afin de s'assurer de leur intégrité.

Toutes les données permettant d'identifier le client sont stockées sur la carte à puce SIM. Utilisée par les réseaux GSM, elle permet d'accroître la sécurité dans l'architecture WAP. L'avantage de WIM [WIM] est de standardiser l'accès aux informations stockées. Ce qui permet aux développeurs de fournir des applications sécurisées indépendamment du type des terminaux mobiles utilisés (Figure 2.6).

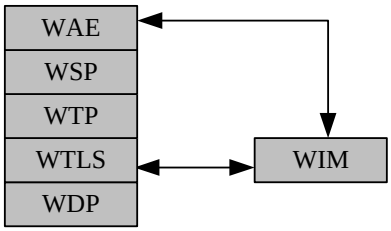

Figure 2.6. WIM dans l'architecture WAP

Dans la version WAP2.0, la sécurité est définie par une implémentation du protocole TLS dans le terminal WAP, comme pour l'I-mode. Cette architecture assure l'authentification de bout en bout. Néanmoins, un problème de performance se pose compte tenu du temps pris par le calcul cryptographique, la charge protocolaire et le transfert des données.

Dans cette architecture, le rôle de la passerelle passe d'un traducteur à un routeur intelligent (proxy).

Cette version oblige le client à implémenter le protocole TCP/IP afin d'assurer la fiabilité de transport qui est indispensable pour TLS. En plus, elle nécessite la définition de nouveaux types de certificats afin d'être supportée par les terminaux légers (exemple, le téléphone portable).

2.3.1 A propos de WIM

Le module WIM peut être utilisé pour effectuer :

- l'identification et l'authentification du client,
- les opérations de signature et les opérations de récupération d'une clé publique.

Dans le contexte WTLS, le module WIM est utilisé pour gérer les connexions sécurisées. Il peut être appliqué, en utilisant *WMLScript*, pour déployer des applications sécurisées non-WAP. Les fonctionnalités fournies par WIM sont, entre autres, la signature afin d'authentifier le client, le chiffrement/déchiffrement en utilisant une paire de clés privée/publique, le stockage des certificats utilisateurs et des CAs. Il assure aussi le service de non répudiation. Son implémentation peut se faire de plusieurs façons :

[21] WAP Identity Module
[22] C'est l'équivalent du Javascript pour le WAP. Il s'agit d'un langage script côté client.

- La première implémentation possible de WIM est la carte SIM des téléphones portables de la norme GSM. Cette dernière possède l'avantage d'être déjà commercialisée. L'existence de l'application WIM sur la carte SIM n'a pas d'impact sur les fonctionnalités GSM. Une entité mobile qui ne supporte pas WIM peut toujours utiliser les fonctionnalités GSM.
- Implémentation du module WIM sur une carte à puce indépendamment de la carte SIM. Cette implémentation a l'avantage de répondre aux besoins de sécurité dans le cas de l'utilisation du terminal mobile comme terminal de paiement.
- Implémentation de WIM sur le terminal.

2.3.2 Gestion des connexions WTLS

La gestion d'une connexion WTLS permet à un client de se connecter à un serveur et de négocier les options de sécurité supportées. L'établissement de la connexion sécurisée se fait en plusieurs étapes. Pendant le processus d'établissement, le client ou le serveur peuvent interrompre à tout moment la négociation. Les sujets négociés sont les paramètres de sécurité (par exemple : la longueur des clés, les méthodes d'échange de clés et l'authentification).

Le déroulement des échanges dans WTLS est divisé, comme dans TLS, en deux phases : la phase d'échange de clés, appelée également phase préliminaire, et la phase d'échange des données.

2.3.3 Services assurés par WTLS

WTLS fournit trois services de sécurité : la confidentialité, l'intégrité et l'authentification du client auprès de la passerelle. Les implémentations de WTLS sont divisées en trois classes (Tableau 2.2) :

- La classe 1 assure la confidentialité et l'intégrité des données, elle est implémentée par défaut.
- La classe 2 assure la confidentialité, l'intégrité et l'authentification de la passerelle.
- La classe 3 assure les services de la classe 2 plus l'authentification du client.

Caractéristique	Classe 1	Classe 2	Classe 3
Echange de clé publique	M	M	M
Certificats serveur	O	M	M
Certificats client	O	O	M
Négociation du secret partagé	O	O	O
Compression	-	O	O
Chiffrement	M	M	M
MAC	M	M	M
Interfaçage avec les cartes embarquées	-	O	O

M : obligatoire, O : Optionnelle

Tableau 2.2. Classes WTLS.

2.3.4 Fiabilité du "*Handshake*" sur les services non connectés

Sur une couche réseau offrant des services non connectés et non fiables, les messages peuvent être perdus, dupliqués et non ordonnés. Pour assurer la fiabilité des messages "*Handshake*" dans ce contexte, WTLS oblige les messages "*Handshake*" envoyés vers la même destination d'être

concaténés dans un seul SDU[23] (unité de données de WDP[24] [WDP]). Le client retransmet le message "*Handshake*" si nécessaire et le serveur doit répondre correctement à ce message.

Durant la phase "*Handshake*", plusieurs messages peuvent être envoyés avant de recevoir une réponse du destinataire. Ces messages doivent être concaténés dans un seul T-SDU[25] pour garantir l'arrivée des messages en ordre après une transmission ou une retransmission. Par exemple, les messages *ServerHello, ChangeCipherSpec* et *Finished* de la procédure "*Handshake*" peuvent être envoyés dans un seul T-SDU. La taille maximale du T-SDU doit pouvoir contenir tous ces messages.

Dans le cas du "*Handshake*" complet, le client doit retransmettre les messages *ClientHello* si les réponses attendues du serveur ne sont pas reçues dans les délais prédéfinis. Si le compteur du nombre de retransmission a atteint un maximum prédéfini, le client arrêtera la procédure de "*Handshake*". Le serveur doit retransmettre, sauf pour le cas d'un nouveau *ClientHello*, le message *ServerHello* tant qu'il y a la réception d'une duplication du message *ClientHello*. Il en est de même pour le message *Finished*.

Dans le cas du "*Handshake*" abrégé, le client doit retransmettre aussi les messages *ClientHello* si les réponses attendues du serveur ne sont pas reçues dans les délais prédéfinis. Durant la fin de cette phase, le client continue à envoyer les messages *ChangeCipherSpec* et *Finished* jusqu'à l'obtention de la part du serveur du message déchiffré avec succès ou bien d'une *Alert*e *duplicated_Finished_received*. Cependant, le serveur doit ignorer les messages *Finished* dupliqués.

2.4 Convergence entre WTLS et TLS

La spécification de WTLS est presque une copie conforme de celle du protocole TLS. Les deux protocoles sont divisés en quatre sous protocoles assurant presque les mêmes services. En ce qui concerne l'établissement des sessions sécurisées, elle se déroule en deux phases :

1) La phase préliminaire durant laquelle a lieu l'identification des parties, la négociation des attributs cryptographiques, la génération et le partage des clés.

2) La phase des échanges de données durant laquelle a lieu la sécurisation à partir des algorithmes et des paramètres négociés dans la phase préliminaire.

La notion de connexion est introduite dans WTLS et TLS afin de permettre à une application de rafraîchir ou de modifier certains attributs de sécurité sans remettre en cause tous les attributs déjà négociés en début de session.

2.5 Divergence entre WTLS et TLS

Le modèle Internet offre deux niveaux de sécurité. Le premier est appliqué au protocole de transport TCP et le deuxième est appliqué au protocole IP. En revanche, le modèle WAP offre un seul protocole WTLS créé pour qu'il soit utilisé sur des protocoles de transport avec ou sans connexion, il se positionne entre WDP et WTP [WTLS].

---

[23] Service Data Unit
[24] Wireless Datagram Protocol
[25] Transport SDU

Le premier point de divergence avec TLS est le fait d'appliquer WTLS sur un protocole de transport non connecté. C'est le résultat de la position de WTLS au-dessus de WDP dans la pile WAP.

Le second point est aussi dû à la position de WTLS au-dessous de WTP dans la pile. Le risque de compresser plusieurs fois les données est présent.

Le troisième point est que le sous protocole "*Record*" dans WTLS n'effectue pas la segmentation[26] des données.

En supposant que le terminal soit connecté à une seule passerelle, tous les points de divergences n'ont plus beaucoup d'impact sur la connexion sécurisée dans la pile WAP. Dans ce contexte, WTLS assure avec succès la gestion de perte, la détection de duplication et le réarrangement des paquets.

2.6 La charge cryptographique de TLS et WTLS

Les protocoles TLS et WTLS supportent des opérations cryptographiques comme la signature numérique, le chiffrement par flux et par bloc, le calcul du condensât et le chiffrement asymétrique ou à clé publique. Les attributs de ces opérations peuvent être négociés pour chaque transmission sécurisée.

Le calcul cryptographique intervient dans la phase "*Handshake*" durant la vérification du certificat, la génération de la clé de session correspondant à une clé secrète appelée *master_secret*, la génération du MAC, la génération des clés de chiffrement et dans le calcul du message *Finished*. TLS et WTLS échangent les paramètres cryptographiques nécessaires afin de permettre à un client et un serveur de se mettre d'accord sur un *master_secret*.

TLS et WTLS utilisent la technique de chiffrement à clé symétrique, comme DES et RC4 afin d'assurer la confidentialité des données. Ils utilisent également le chiffrement à clé publique, comme RSA, afin d'échanger les paramètres permettant de calculer les clés d'une session.

L'authentification dans WTLS et TLS se réalise par l'utilisation d'un certificat de type X509 [X509] (*WTLSCert* et X9.68 sont utilisés avec WTLS comme décrit dans [WPKI]). Ce certificat peut être utilisé pour signer et/ou chiffrer les données.

Il faut noter que WTLS peut utiliser un algorithme d'échange de clé appelé ECC[27] [WTLS]. Cet algorithme assure les fonctions essentielles effectuées par RSA avec moins de ressources de CPU [Palm]. Par ailleurs, il y a une différence entre ECC et RSA : ECC engendre une clé de session symétrique pour chaque message alors que RSA permet un transport sécurisé avec une clé de session symétrique à longue durée.

Afin d'évaluer la charge globale des traitements cryptographiques de WTLS et TLS, nous proposons une implémentation permettant d'évaluer le temps nécessaire (ou débit) pour établir une session, ouvrir une connexion, traiter les données des applications et reprendre une session déjà établie.

Les performances de TLS et de WTLS dépendent de plusieurs facteurs, dont la capacité du processeur utilisé, la mémoire, la suite de chiffrement choisie dans les mécanismes protocolaires et l'arborescence de certification.

---

[26] WTP offre le service de segmentation.
[27] Elliptic Curve Cryptographic

Le coût du traitement cryptographique est évalué en considérant une implémentation avec l'hypothèse suivante :

- DES comme algorithme de chiffrement symétrique
- RSA avec une clé de 1024 bits comme algorithme d'échange de clés.
- MD5 et SHA-1 comme fonctions de hachage
- Le client et le serveur partagent le même CA. Autrement dit, le nombre de certificats dans la chaîne de certification est égal à 1.

Ces algorithmes cryptographiques jouent un rôle important dans l'estimation du calcul cryptographique de TLS et WTLS. Le tableau ci-dessous illustre deux benchmarks réalisés sur une carte à puce SLE66CX16OS à 7.5MHz et sur un Processeur Intel PC à 440 MHz pour les algorithmes choisis.

Algorithme		Intel PC 440 MHz	SLE66CX16OS 7.5 MHz
DES		0.0116 ms	3.7 ms
RSA 1024	signature	32.4000 ms	880.0 ms
	verification	1.8000 ms	5.6 ms
SHA1 512 bits		0.0059 ms	5.6 ms
MD5 512 bits		0.0035 ms	9.0 ms

Tableau 2.3. Temps de traitement des opérations cryptographiques.

D'autres paramètres ont une influence sur la performance des protocoles de sécurité comme la taille des messages de la phase *Handshake* et la taille des attributs de sécurité (les valeurs aléatoires, l'identificateur de la session, etc.).

En comparant TLS et WTLS, nous avons analysé un *Handshake* complet pour chaque protocole et évalué la charge cryptographique nécessaire du côté client pour établir une session, ouvrir une connexion et traiter des messages de 16 Kilo octets utilisant les algorithmes cryptographiques proposés.

Les résultats du tableau 2.4 montrent que la charge cryptographique côté client est légèrement supérieure à celle côté serveur ou passerelle à cause de la capacité du processeur et de la mémoire utilisée.

Nous constatons donc que dans les réseaux sans fil, WTLS est adopté parce qu'il génère moins de calculs cryptographiques et que sa charge d'échanges (durant la phase *Handshake*) est plus réduite que celle du TLS. En effet, le tableau suivant montre qu'une session WTLS consomme moins de ressources et de temps de traitement qu'une session TLS.

Protocole	TLS	WTLS
Session sécurisée (ms)		
Client	1447.00	1128.53
Passerelle	-	1.99
Serveur	2.17	-
Total	1349.17	1130.52
Chiffrement/déchiffrement d'un message de 16 Koctets (ms)		
Client	238.10	211.10
Passerelle	-	0.27
Serveur	0.27	-
Total	238.37	211.37

Connexion sécurisée (ms)		
Client	79.40	76.00
Passerelle	-	00,05
Serveur	0.06	-
Total	79.46	76.05
Session résumée (ms)		
Client	258.70	171.40
Passerelle	-	0.15
Serveur	0.19	-
Total	258.89	171.55

Tableau 2.4. Comparaison de la charge cryptographique entre TLS et WTLS.

2.7 Conclusion

L'évolution des applications sensibles aux attaques (bancaires, commerce électronique, vote, etc.) est liée, entre autres, à la possibilité du Forum WAP de présenter une solution de sécurisation de bout en bout.

Dans ce chapitre, nous avons présenté les failles des mécanismes de sécurité dans l'architecture WAP1.x (WTLS dans WAP 1.1 et WIM dans WAP 1.2) qui offrent une sécurité à deux tronçons. L'établissement d'une connexion sécurisée entre un client et un fournisseur de services passe obligatoirement par le fournisseur d'accès, ce qui est inacceptable dans le cas des instituts de finance qui ne font pas confiance à un tiers.

Le Forum WAP s'est rendu compte de l'intérêt de la sécurisation de bout en bout. Une version 2.0 des spécifications WAP a donc été publiée en décembre 2000 à cet effet. Nous avons montré que cette version apporte des solutions partielles mais aussi coûteuses en terme de calcul cryptographiques et charges protocolaires et ne couvre toujours pas l'ensemble de services de sécurité. Dans cette optique, nous proposons deux contributions afin de mieux adapter TLS aux environnements WAP. Elles seront présentées dans le chapitre 4.

Chapitre 3

TLS dans les réseaux 802.11 sans fil

Résumé du contenu

3.1 Introduction

L'Ethernet est un protocole de réseau informatique à commutation de paquets implémentant la couche Physique et la sous-couche MAC[28] du modèle OSI. Inventé par Xerox au début des années 70 et normalisé par le groupe IEEE[29] vers 1980 sous la norme IEEE 802. L'Ethernet est la technologie la plus utilisée pour les réseaux locaux en bus. Elle désigne une architecture de réseau local sur lequel sont connectées des stations communiquant entre elles par un protocole de communication particulier (ex. CSMA/CD[30]).

L'Ethernet a plusieurs caractéristiques importantes ; notamment un débit nominal allant jusqu'à 10 Mégabits par seconde, une topologie en bus, un faible coût de développement et un coût de mise en œuvre en diminution constante.

Le groupe IEEE a classifié ses standards en plusieurs comités. Par exemple, le comité d'IEEE 802 s'occupe des réseaux MAN[31] qui sont des réseaux locaux à l'échelle d'une ville ou d'une entreprise. Par la suite, nous citons une liste de groupes de travail concernant la série de 802 :

- 802.1 : management et interface.

[28] Medium Access Control
[29] Institute for Electrical and Electronics Engineers
[30] Carrier Sense Multiple Access with Collision Detection
[31] Metropolitan Area Network

- 802.2 : définit le protocole LLC[32], s'interfaçant aisément sur n'importe quel réseau LAN ou MAN de type 802.
- 802.3 : spécifie les couches physiques et basses du logiciel. La plupart des réseaux câblés se conforment à la spécification 802.3 pour les réseaux Ethernet basés sur la méthode d'accès CSMA/CD.
- 802.4 : *Token-Passing Bus Access Method*.
- 802.5 : *Token-Passing Ring Access Method*.
- 802.6 : *Metropolitan Area Network*.
- 802.7 : *Broadband Technical Adversory Group*.
- 802.8 : *Fiber Optic Technical Adversory Group*.
- 802.9 : *Integrated Voice and Data Networks*.
- 802.10 : *Network Security*.
- 802.11 : *Wireless LAN*.
- 802.12 : *100VG-AnyLAN*.

Nous étudions dans ce chapitre la série des standards de 802 sans fil spécifiant les problèmes et la discipline globale de réseaux LAN[33] sans fil. Plus particulièrement, nous nous intéressons aux réseaux 802.11 ([802.11-Part1], [802.11-Part2]) sans fil. Ces derniers ont connu une phase de progression très rapide grâce à leur simplicité de déploiement, leur rapidité et leur facilité de mise en oeuvre. Cependant, ces atouts cachent une réalité beaucoup moins attirante d'un point de vue sécurité. Dans ce chapitre, nous abordons les problématiques de sécurité de ce type de réseaux et les attaques possibles. Ensuite, nous dressons un panorama de la théorie et des solutions pratiques de sécurité.

3.2 La norme 802.11

A la base, les réseaux 802.11 sans fil peuvent être vus comme une extension des réseaux Ethernet câblés. Ces réseaux recouvrent un ensemble de technologies permettant d'établir un réseau local sans l'utilisation du câblage pour les liaisons entre les ordinateurs. En effet, le câblage est remplacé par des liaisons radios.

Les principales technologies permettant de développer des WLAN sont ceux appartenant aux normes IEEE 802.11, parmi lesquelles *Hiperlan* et *Bluetooth*. La norme la plus connue et la plus populaire de WLAN est le 802.11b (WiFi[34]). Cette norme permet d'offrir un vaste panorama d'applications grâce aux caractéristiques avantageuses à la technologie Ethernet.

3.2.1 La topologie du réseau 802.11 sans fil

La norme 802.11 définit deux modes de topologie : le mode infrastructure et le mode ad hoc.

[32] Logical Link Control
[33] Local Area Network
[34] Wireless Fidelity

3.2.2 Le mode Infrastructure

Un réseau 802.11 est un ensemble de cellules de base (BSS[35]). Chaque cellule BSS comporte un point d'accès matérialisé par un dispositif d'émission/réception. Les cellules sont reliées par une infrastructure de communication fixe et interconnectées par un système de distribution afin de former un ESS[36]. Cette infrastructure incorpore un portail permettant d'assurer l'interface avec un réseau local.

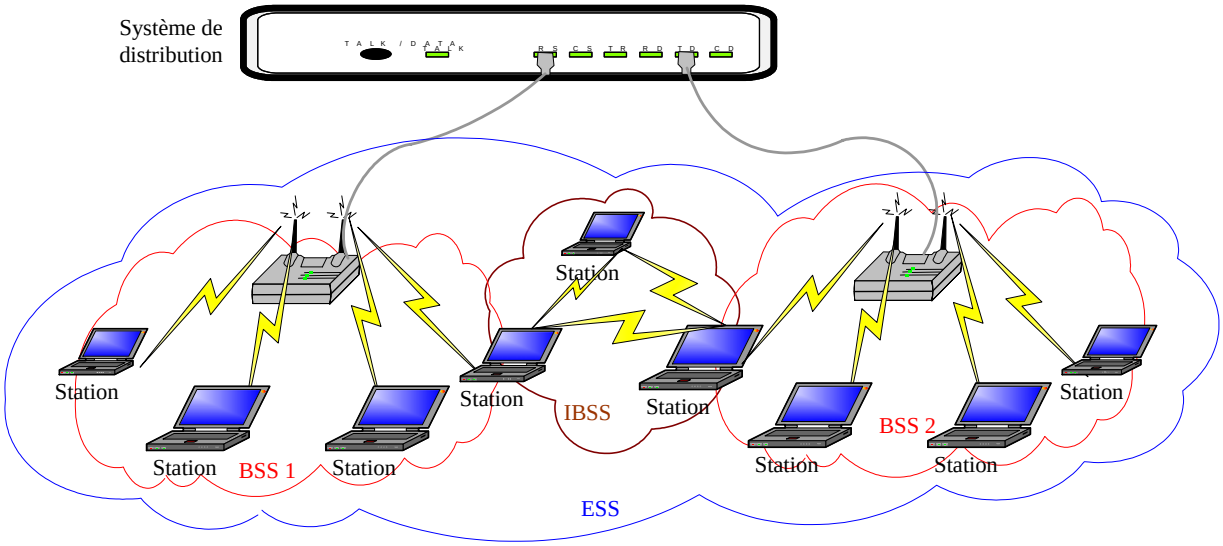

Figure 3.1. L'architecture de 802.11 sans fil

3.2.3 Le mode ad hoc

Ce mode représente un ensemble de stations 802.11 sans fil qui communiquent entre elles sans avoir recours à un point d'accès ou une connexion à un réseau filaire à travers le système de distribution [Hbook]. Chaque station peut établir une communication avec n'importe quelle autre station dans la cellule que l'on appelle cellule indépendante IBSS[37].

Dans les deux modes infrastructure et ad hoc, chaque réseau de service est identifié par un identificateur de réseau SSID[38]. Par conséquent, toute station désirant se connecter au réseau étendu doit connaître au préalable la valeur du SSID.

3.2.4 La sécurité dans les réseaux 802.11 sans fil

Les réseaux 802.11 sans fil ont introduit de nouveaux besoins en sécurité en comparaison aux réseaux fixes. En effet, le manque de protection physique des points d'accès au réseau et la transmission sur des liens radios sont les causes principales de la vulnérabilité des réseaux sans fil [Samfat].

Un réseau LAN assure un niveau de sécurité plus élevé qu'un réseau LAN sans fil puisque LAN offre une protection physique de ses équipements alors qu'avec le réseau WLAN, ces équipements ne peuvent pas être nécessairement protégés si aucun chiffrement n'est utilisé.

---

[35] Basic Set Service
[36] Extended Service Set
[37] Independent BSS
[38] Service Set IDentity

Dans les réseaux sans fil tel que le WLAN, le support est partagé, c'est-à-dire que tout ce qui est transmis et envoyé sur le support peut être intercepté. Afin de pallier ses faiblesses et de restreindre l'accès non autorisé au support et ensuite aux services offerts par le réseau local, la norme 802.11 a définit un mécanisme de contrôle d'accès réseau à base de port. Le but de ce mécanisme est d'authentifier et d'autoriser des équipements attachés au port d'un réseau local.

Dans la terminologie de 802.11 sans fil, ce port est appelé "association". Nous notons ici qu'avant toute association avec le point d'accès, la station doit s'authentifier auprès de ce point d'accès. Les techniques d'authentification entre la station et le point d'accès sont de deux sortes :

Open System Authentication : avec cette technique, aucune authentification explicite entre les deux entités n'est établie. De ce fait, la station mobile peut s'associer avec le point d'accès et commencer à envoyer et à recevoir de données.

Shared Key Authentication : cette technique établie une authentification basée sur un secret partagé. En effet, la station voulant s'associer avec le point d'accès lui envoie une requête d'authentification. Le point d'accès répond par une valeur aléatoire de 128 bits. La station chiffre cette valeur avec la clé secrète en appliquant un algorithme de chiffrement symétrique. A sa réception, le point d'accès déchiffre avec le même secret, le message chiffré et confirme l'authentification de la station si la vérification est correcte, sinon il répond par un message négatif.

La norme 802.11 utilise WEP[39] [WEP] comme un protocole de chiffrement symétrique afin de protéger l'interface radio. Ce protocole assure un niveau de sécurité similaire à celui de LAN fixe.

3.2.4.1 Le protocole WEP

Le principe de WEP consiste à utiliser RC4[40] [RC4] dans le chiffrement de données en mode flux (*stream cipher*). A partir d'une clé de longueur comprise entre 8 et 2048 bits, il génère une suite d'octets pseudo aléatoire nommée *KeyStream* (Ks). Cette série d'octets (figure ci-dessous) est utilisée pour chiffrer un message M à l'aide d'un protocole classique de Vernam, réalisant un OU exclusif (XOR) entre Ks et M ; C = Ks XOR M, où C est le message chiffré.

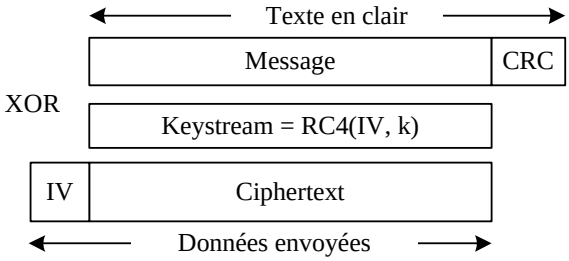

Figure 3.2. Le chiffrement WEP

WEP a été conçu dans le but d'empêcher l'espionnage et la modification (d'une partie) des données échangées entre un terminal et un point d'accès. La protection est établie via le chiffrement de trame radio en utilisant l'algorithme symétrique RC4. Ce protocole utilise une clé

[39] Wireless Encryption Privacy
[40] Rate Code 4

secrète partagée et installée sur tous les client (de 40 ou 128 bits) et un vecteur d'initialisation (IV[41]) de 24 bits transporté en clair dans chaque paquet.

3.2.4.2 Quelques vulnérabilités du WEP

WEP présente de nombreuses failles de sécurité ; notamment sur la gestion de clés (tous les utilisateurs ont la même clé). En effet, le WEP ne définit aucun moyen pour gérer les clés de chiffrement. C'est à l'administrateur de WLAN de créer les clés, de les distribuer, de les archiver/stocker d'une manière protégée, de clarifier qui a telles ou telles clés cryptographiques et de révoquer les clés compromises. Les spécifications de WEP n'ont pas pris en charge ces problématiques.

Plusieurs problèmes résident dans le protocole WEP [WEP-Hack] lui-même :

1) Comme les clés WEP sont partagées, la confidentialité de la communication n'est pas assurée.

2) Le nombre de *keystream* est limité à 2^24 clés. Un pirate peut facilement générer des trames, enregistrer leur forme chiffrée puis déduire et stocker les *KeyStream* identifié par leur IV.

3) Avec WEP, la station mobile n'authentifie pas le point d'accès. Elle ne peut donc pas vérifier si elle s'accorde avec le vrai point d'accès dans le réseau WLAN.

4) L'intégrité des trames est assurée à l'aide d'une somme de contrôle CRC[42]-32. Cette fonction est linéaire par rapport à l'opération XOR et donc, nous pouvons modifier un ou plusieurs bits dans la trame chiffrée tout en recalculant une valeur correcte du CRC (CRC(A XOR B) = CRC(A) XOR CRC(B)). En effet, si C est le résultat du chiffrement du message M alors C = RC4(K) XOR (M || CRC(M)). L'attaque prévoit le choix d'une séquence de bits S destinée à modifier le message M et le calcul de sa somme de contrôle CRC(S) utilisant CRC-32. L'intrus est alors en mesure de répercuter toutes modifications de M sur le CRC(M) :

C' = C XOR (S || CRC(S)) = RC4(K) XOR (M || CRC(M)) XOR (S || CRC(S))

= RC4(K) XOR (M XOR S || CRC(M) XOR CRC(S))

= RC4(K) XOR (M XOR S || CRC(M XOR S)).

Il n'est pas nécessaire de connaître le texte en clair pour mener cette attaque. Elle peut être se réaliser en changeant le champ TTL[43] du paquet TCP qui indique le temps durant lequel un paquet est considéré comme étant valide. Un homme au milieu peut changer la valeur de TTL durant le transport de la trame chiffrée. Le message initial sera donc perdu.

4) *Spoofing* de l'authentification : le principe de l'authentification avec WEP est que le point d'accès envoie un challenge de 128 octets en clair et que l'utilisateur doit lui renvoyer chiffré. Si la réponse de l'utilisateur est correcte, alors le point d'accès considère que la station possède la clé du WEP. L'attaque peut être réalisée en interceptant le message en clair et la réponse cryptée. Ensuite, l'attaque déduit RC4(IV,K) de la bonne longueur et il demande à s'authentifier.

5) L'attaque de Fluhrer, Mantin et Shamir [WEP-Hack]. Cette attaque permet de recouvrer le secret partagé de 104 bits après l'émission approximativement de quatre millions de trames chiffrées. Elle utilise des valeurs IVs dites résolvantes, de la forme (3+B, 255, N) avec B un octet

[41] Initialization Vector
[42] Cyclic Redundancy Check
[43] Time To Live

du secret partagé et N une valeur quelconque comprise entre 0 et 255. Environ 60 valeurs résolvantes suffisent à retrouver un octet du secret partagé. Un rapide calcul montre que l'on obtient une valeur résolvante toutes les 2^16 trames, soit 60 occurrences après environ 4 millions de paquets.

A cause des problèmes évoqués précédemment (nous ne parlons pas ici d'autres types d'attaques comme DoS[44], *Man-In-The-Middle*, etc.), il est fortement conseillé d'utiliser d'autres méthodes et mécanismes de sécurité que nous détaillons par la suite.

3.2.4.3 Les perspectives du WEP

Le protocole WEP présente des failles importantes et il n'accomplit pas correctement tous ses engagements de sécurité. De ce fait, la norme 802.11b a annoncé une version évoluée du WEP appelé WEP2 qui apporte des améliorations en terme de sécurité, mais ne résout pas tous les problèmes de sécurité posés par WEP. En effet, la grande faiblesse de WEP provient de l'utilisation de clés statiques RC4 et de l'absence de la distribution dynamique de clés de session et du mécanisme d'authentification mutuelle.

Afin d'affronter ces problématiques, la norme IEEE 802.1X [802.1X] a été introduite dans le but de gérer l'authentification via un serveur centralisé. Cette norme détaillée ci-après, assure aussi la distribution et le renouvellement périodique et automatique des clés.

Le groupe de travail 802.11i [802.11i] est formé au sein de l'IEEE 802.11 dans le but d'étudier une architecture destinée à combler les lacunes de WEP et de spécifier d'une manière détaillée une méthode de distribution des clés. Ce groupe a défini une solution intermédiaire appelée WPA[45] [WEP-WPA]. Elle regroupe le 802.1X (pour l'authentification mutuelle) et le protocole TKIP[46] (pour le chiffrement avec une clé dynamique).

3.3 Le protocole IEEE 802.1X

Afin de pallier au manque de sécurité du standard 802.11, l'IEEE propose le standard 802.1x [802.1X] comme base pour le contrôle d'accès, l'authentification et la gestion de clés. Le 802.1x était initialement conçu pour la gestion sécurisée des accès des réseaux câblés à base de commutateurs de paquets. Son objectif est de bloquer le flux de données d'un utilisateur non authentifié. C'est-à-dire qu'il permet une authentification lors de l'accès au réseau et donc une contrôle d'accès aux ressources.

Le 802.1x utilise un modèle qui s'appuie sur trois entités fonctionnelles :

- Le système à authentifier (*supplicant*) : c'est un poste de travail (terminal informatique) demandant un accès au réseau.
- Le certificateur (*authenticator*) : c'est l'unité qui contrôle et fournit la connexion au réseau. Un port contrôlé par cette unité peut avoir deux états : non autorisé ou autorisé. Lorsque le client n'est pas authentifié, le port est dans l'état non autorisé et seulement le trafic nécessité par l'authentification est permis entre le terminal et le certificateur. Le certificateur transmet la requête d'authentification au serveur d'authentification en utilisant le protocole EAP[47]. Les

[44] Denial of Service
[45] Wi-Fi Protected Access
[46] Temporal Key Integrity Protocol
[47] Extensible Authentication Protocol

autres paquets sont bloqués lorsque le port se trouve dans l'état non autorisé. Cependant, à la fin de ces échanges, le certificateur analyse le message notifiant l'échec ou le succès de la procédure et filtre les trames de la station mobile en fonction du résultat.

- Le serveur d'authentification : il réalise la procédure d'authentification avec le certificateur et valide la demande d'accès. Durant cette phase, le certificateur n'interprète pas le dialogue entre le serveur et le terminal. Il s'agit d'un simple relais passif. Si la requête d'accès est validée par le serveur, le port est commuté dans l'état autorisé et le client est autorisé à avoir un accès complet au réseau.

802.1x définit plusieurs techniques d'encapsulation utilisées dans le but de transporter les paquets d'authentification et de gestion du protocole EAP [EAP]. Cette technique est connue sous le nom EAP sur LAN entre la station et le point d'accès et sous le nom EAP sur RADIUS[48] entre le point d'accès et le serveur d'authentification RADIUS.

Schématiquement, l'insertion d'une station mobile dans un environnement 802.1X se déroule de la manière suivante :

- Après la phase d'association avec le point d'accès, ce dernier envoie au terminal une requête d'identité *EAP-Request.Identity*.
- La station mobile produit en retour une réponse *EAP-Response.Identity*. Cette réponse comporte l'identité du client et les méthodes d'authentification supportées.
- A ce moment, le point d'accès transmet au serveur d'authentification le message *EAP-Response.Identity* encapsulé dans une requête RADIUS. Durant l'échange des messages EAP (requêtes et réponses) entre le serveur d'authentification et la station mobile, le point d'accès agit comme un simple relais passif.
- Le serveur d'authentification prend la décision d'accepter ou de refuser l'accès au réseau et il indique le succès ou l'échec de la procédure d'authentification via le message *EAP-Success* ou *EAP-Failure*. Si le serveur d'authentification accepte le client, alors l'état du port change. Il passe à l'état autorisé, sinon, le port reste dans l'état non autorisé.

[48] Remote Authentication dial-In User Service

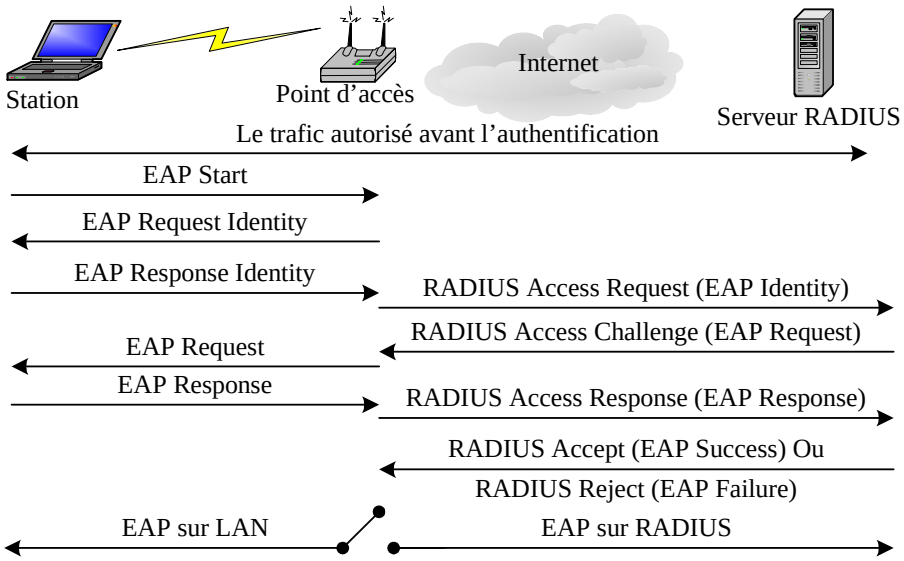

Figure 3.3. L'architecture 802.1X

Après le succès de la phase d'authentification, le serveur transmet une clé (*Unicast*) de chiffrement (partagée avec la station mobile) au point d'accès qui l'utilise pour la génération de clés servant à chiffrer les données échangées avec la station.

3.3.1 Les méthodes d'authentification de 802.1X

EAP est l'actuel protocole d'authentification utilisé par les entités de 802.1X. Il propose une extension des méthodes d'authentification utilisées pour le protocole PPP. EAP est un protocole défi-réponse qui peut être prolongé afin de fonctionner au-dessus de n'importe quel mécanisme de transport. Dans le cas des réseaux 802.11 sans fil, les messages EAP sont transportés par de trames EAPoL.

Le protocole EAP réalise une enveloppe générique pour de multiples méthodes d'authentification. Un message EAP comporte un entête de cinq octets et un champ optionnel des données. Les messages EAP se décomposent en quatre classes :

- Requête : utilisé pour les demandes d'authentification.
- Réponse : utilisé pour répondre à des demandes d'authentification.
- Succès et Echec : utilisés afin d'informer le point d'accès et la station mobile à propos du résultat de la procédure d'authentification.

La requête et la réponse sont étiquetées par le même *Identifier* compris entre 0 et 255 et elles ont une longueur totale codée sur deux octets. En plus, le message EAP contient un champ *Type* sur un octet qui désigne le protocole d'authentification transporté ou des opérations particulières.

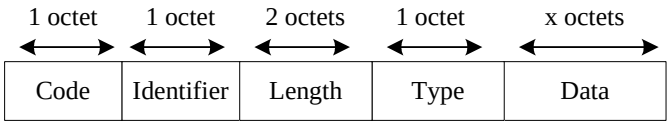

Figure 3.4. Le format du message EAP

Le protocole EAP est utilisé avec 802.1X d'une manière transparente entre la station mobile et le serveur d'authentification au travers du certificateur. Cependant, 802.1X nécessite la coopération entre un serveur d'authentification (comme RADIUS) et une méthode d'authentification. La méthode d'authentification est une couche au-dessus de EAP et elle définit des mécanismes de sécurité et de distribution de clés. Néanmoins, 802.11 n'a pas précisé la façon d'implémenter EAP avec 802.1X. Pour cette raison, plusieurs couches sont définies au-dessus de EAP, citons EAP-TLS, EAP-TTLS et PEAP.

3.3.1.1 EAP-TLS

EAP-TLS [EAP-TLS] est une méthode d'authentification standardisée par l'IETF[49], elle est basée sur le protocole TLS qui est actuellement une solution crédible pour la sécurisation des échanges.

L'authentification de TLS avec EAP est simple. Le serveur et la station mobile utilisent la phase "*Handshake*" de TLS afin de s'authentifier en utilisant des certificats de type X.509. Durant leurs transports, les messages TLS seront fragmentés et encapsulés dans des paquets EAP-TLS. La fragmentation a lieu pour une simple raison, la taille d'un *record* TLS est d'au plus 16384 octets alors que le protocole RADIUS [RADIUS] limite sa charge utile à 4096 octets et de surcroît la taille des trames 802.11 est limitée à 2312 octets. En conséquence, EAP-TLS supporte un mécanisme de segmentation de *record*s TLS.

Bien qu'une session TLS établisse un canal chiffré entre le serveur d'authentification et la station mobile, ces derniers ne l'utilisent pas. D'ailleurs, la clé de la session TLS est utilisée afin de générer d'autres clés servant à établir un canal chiffré entre la station et le certificateur. En effet, le serveur transmet la clé de la session au certificateur qui l'utilise pour le chiffrement WEP. Les étapes suivantes expliquent en détail le déroulement d'une session EAP-TLS :

La session EAP-TLS commence typiquement entre la station et le point d'accès. Après la phase d'association, le point d'accès envoie une requête d'authentification à la station.

La station répond avec l'identifiant de l'utilisateur. Ce message est relayé par le point d'accès vers le serveur d'authentification.

A ce moment, le serveur d'authentification initie le processus d'authentification de TLS en envoyant le paquet *EAP-TLS/start*.

La station répond par le paquet *EAP-Response* contenant le message *ClientHello* du TLS. Ce message contient, entre autres, la version du protocole TLS supportée par le client, une valeur aléatoire et une liste d'algorithmes cryptographiques supportés par le client mobile dans un ordre décroissant de préférence.

[49] Internet Engineering Task Force

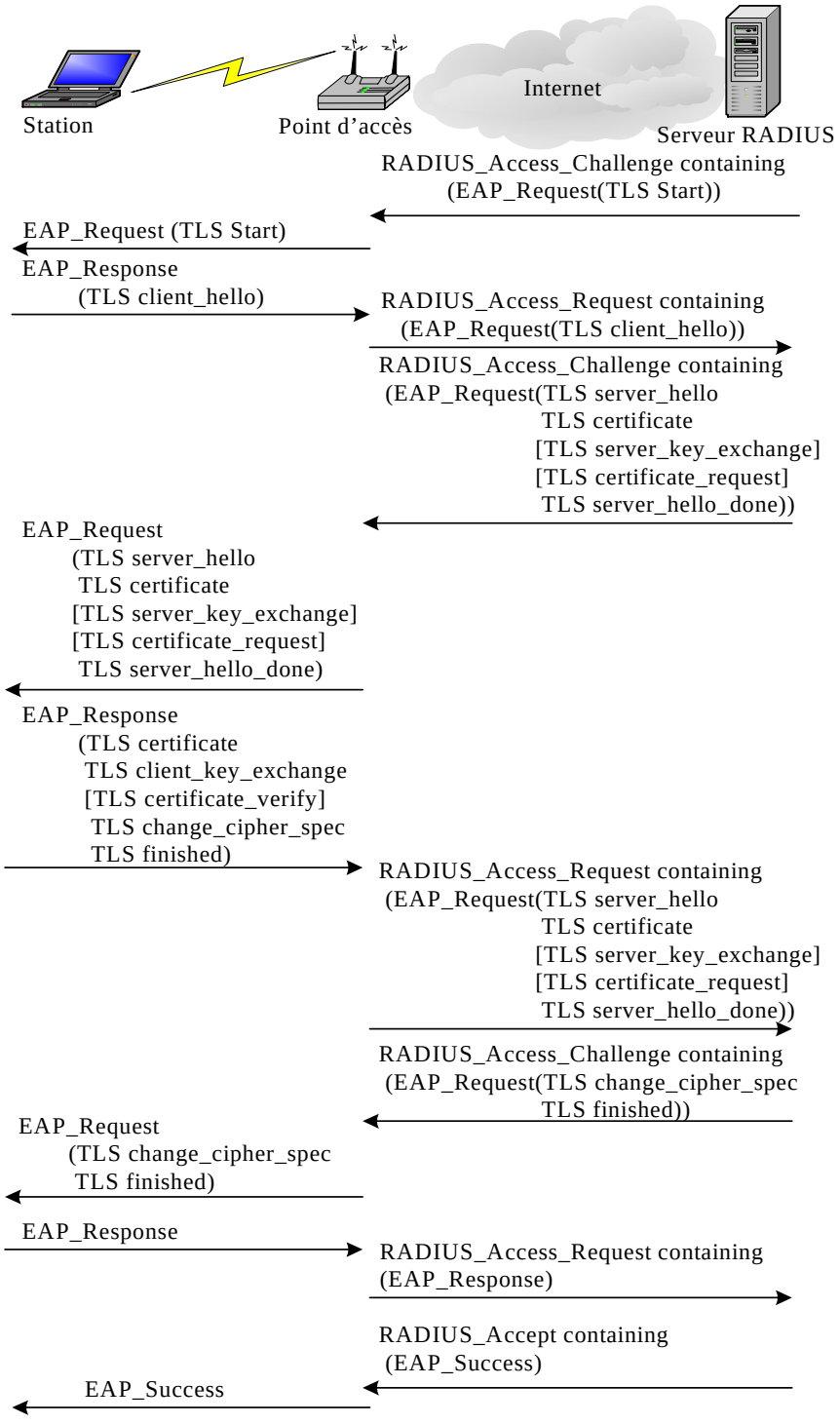

Figure 3.5. L'authentification EAP-TLS

Le serveur doit répondre par un paquet *EAP-Request* contenant le message *ServerHello* du TLS, sinon une erreur fatale se produira et la connexion sécurisée ne sera pas établie. Les attributs de sécurité correspondants à cette étape sont les mêmes définis par TLS. Le message hello du

serveur est suivi par son certificat contenant sa clé publique, par une demande du certificat du client et par le message *ServerHelloDone* afin d'indiquer la fin de son hello.

Le client répond par un paquet *EAP-Response* contenant les messages certificat, le *ClientKeyExchange* comportant la clé secrète *pre_master_secret*, *CertificateVerify*, *ChangeCipherSpec* et *Finished*.

Le client et le serveur définissent une clé de session et calculent les clés de chiffrement et d'authentification et les vecteurs d'initialisation. Ensuite, le serveur répond par un paquet *EAP-Request* dont le champ de données encapsule les messages *ChangeCipherSpec* et *Finished* de TLS. Le message *Finished* termine la phase d'authentification de TLS mais la clé de session n'est pas utilisée pour chiffrer la suite des échanges.

Le client envoie ensuite un paquet *EAP-Response* dont le champ de données est vide et le serveur répond par le massage *EAP-Success*.

A la fin de la phase d'authentification, le serveur envoie la clé de chiffrement au point d'accès en utilisant l'attribut RADIUS "MS-MPPE-Recv-Key". Cette clé sera utilisée par le point d'accès afin de chiffrer et signer la clé (par exemple WEP ou WPA) avant d'être transmise au client.

La méthode EAP-TLS assure une authentification efficace à l'aide de certificats numériques. L'utilisation du certificat protége les entités contre plusieurs attaques ; notamment l'attaque *Man-In-The-Middle*. Cependant, durant le déroulement d'une session EAP-TLS, le serveur est relié à l'Internet et donc il peut vérifier l'identité et la validité du certificat du client et de contrôler si le certificat est révoqué ou non. Cependant, le client peut ne pas avoir aucune connexion Internet et donc il ne peut pas suivre la chaîne de certification et vérifier si le certificat du serveur a été révoqué ou non. Néanmoins, le client peut achever cette opération après l'établissement de la session EAP-TLS.

L'utilisation des certificats avec EAP-TLS exige une infrastructure à clé publique (PKI) [PKI]. Cette infrastructure ne peut pas être toujours déployée dans plusieurs types d'entreprises. En effet, elle nécessite la distribution des certificats aux clients et la révocation de celles qui ne sont plus valides. En plus, elle entraîne un surplus important en terme de gestion et de ressources machines et humaines. Notons que le certificat du client passe en clair sur le réseau et donc EAP-TLS n'offre pas la protection d'identité.

Enfin, l'absence de l'intégrité et du chiffrement des messages de notification de la session EAP-TLS (échec ou succès) rend possible la réalisation des attaques de type DoS et MitM. Un intrus peut toujours remplacer le massage *EAP-Success* par le massage *EAP-Reject* et vice versa.

3.3.1.2 EAP-TTLS[50]

EAP-TTLS [EAP-TTLS] est un *Internet Draft* définissant une méthode qui utilise le protocole EAP et le tunnel TLS dans le but d'établir une session authentifiée et sécurisée entre la station 802.1X et le serveur d'authentification. Elle étend l'EAP-TLS et elle est presque aussi sûre que la méthode EAP-TLS, tout en simplifiant le déploiement du PKI. Les certificats PKI ne sont en effet nécessaires que sur le serveur d'authentification, leur utilisation sur les postes de clients est une

[50] Tunnel TLS

option. L'authentification du client peut se réaliser par d'autres moyens que le certificat ; mot de passe (CHAP[51] [CHAP], MSCHAPv2 [CHAPv2]) ou carte à puce.

EAP-TTLS distingue deux phases d'authentification :

- Tunnel TLS, c'est une session TLS avec l'authentification du serveur par un certificat valide. Elle sert à protéger les échanges de la deuxième phase.
- Identification du client par le serveur en utilisant une méthode simple ((CHAP[52] [CHAP], MSCHAPv2 [CHAPv2]), carte à puce, etc.).

EAP-TTLS a un avantage par rapport à EAP-TLS, elle assure la protection de la notification du mécanisme d'authentification et de l'identité du client grâce au tunnel TLS établi durant la première phase. Ce tunnel garantit la confidentialité des échanges pour la deuxième phase. Ce qui donne l'avantage à l'administrateur du réseau de choisir une simple méthode d'authentification pour ses utilisateurs (mot de passe transit en clair dans le Tunnel TLS) et donc de supprimer la complexité de gestion liée aux certificats et à l'infrastructure à clé publique.

En revanche, EAP-TTLS n'est pas toujours protégée contre l'attaque *Man-In-The-Middle* ([MITM1], [MITM2]). En effet, le protocole d'authentification utilisé durant la deuxième phase ne peut pas être conscient s'il est accompli en mode tunnel ou non, c'est-à-dire que EAP-TTLS ne relie pas cryptographiquement les clés de session dérivées par le protocole interne avec celles dérivées par le protocole externe. Ajoutons à cela le fait qu'avec l'absence de la connexion vers l'Internet durant la phase d'authentification, le client ne peut pas proprement vérifier le certificat du serveur.
L'intrus peut intervenir comme un utilisateur légitime dans la première phase du protocole EAP-TTLS. Cela lui permet d'établir une connexion TLS avec le serveur et d'avoir les clés du tunnel TLS. Cette étape est possible puisque aucune authentification n'est exigée du côté de l'intrus.

L'intrus cherche ensuite à provoquer les clients qui sont attachés avec son point d'accès. Son objectif étant de les inciter à utiliser un mécanisme d'authentification autorisé avec le tunnel. D'autres conditions doivent être obtenues afin de réaliser cette attaque : le client doit utiliser les mêmes crédits (i.e. login et mot de passe) en mode tunnel et non-tunnel et l'intrus doit être capable de jouer le rôle du serveur d'authentification.

Il est évident que le client et le serveur peuvent utiliser une méthode d'authentification avec des clés (pré) partagées. Puisque l'intrus n'a pas cette clé, il ne peut donc pas déchiffrer les données échangées entre le client et le serveur. Cependant, même si le client utilise une clé partagée, le serveur utilisera la clé de session connue par l'intrus. En conclusion, l'intrus écartera les données envoyées par le client, ce dernier arrêtera la session à cause de l'absence de la réponse et l'intrus continuera ses échanges avec le serveur comme un utilisateur légitime.

3.3.1.3 PEAP

PEAP[53] [PEAP] et EAP-TTLS sont presque semblables. Les deux protocoles utilisent un canal TLS pour protéger un second échange EAP. Ils conservent les fondations cryptographiques de TLS mais appliquent d'autres mécanismes d'authentification du côté client. Par exemple,

[51] Challenge-Handshake Authentication Protocol
[52] Challenge-Handshake Authentication Protocol
[53] Protected EAP

l'authentification du client PEAP n'ayant pas un certificat, elle peut être établie en utilisant MS-CHAPv2 durant la phase 2.

3.3.2 Le serveur d'authentification RADIUS

L'évolution du standard 802.1X intègre l'ensemble des évolutions du protocole RADIUS spécifié par le RFC 2869 [RFC2869]. Le protocole RADIUS est défini par le RFC 2865 [RFC2865] et ses attributs sont définis par le RFC 2866 [RFC2866].

Le protocole RADIUS s'appuie sur une architecture client/serveur et permet l'accès à distance. Il assure également l'interface avec une infrastructure AAA[54] gérant les comptes utilisateurs en collaboration avec un annuaire LDAP[55] ([LDAP], [LDAPv3].

RADIUS est utilisé par les ISP[56] afin d'authentifier les utilisateurs distants. Il assure également le transport des données d'authentification, d'autorisation et de facturation entre un serveur d'authentification partagé et les NAS[57] distribués qui désirent authentifier leurs clients. Le NAS est responsable du transfert des informations envoyées par le client vers le serveur RADIUS.

Par exemple, lorsqu'un client veut accéder au réseau d'un fournisseur de services, il doit envoyer son accréditation d'authentification (login et mot de passe) au NAS. Le NAS réalisant un pont entre le client et le serveur RADIUS transmet l'accréditation à ce serveur par le message "*Access-Request*". A la réception de ce message et après l'émission et la réception des messages *Access-Challenge* et *Access-Request*, le serveur RADIUS vérifie l'identifiant du NAS et l'appartenance des crédits du client à sa base de données. Ensuite, il indique au NAS le succès ou l'échec de la demande du client à l'aide de l'un des deux messages *Access-Reject* ou *Access-Success*. Nous notons ici que le NAS ne peut pas interpréter les crédits d'authentification du client puisqu'ils sont chiffrés entre le client et le serveur RADIUS. Les échanges d'autorisation entre le NAS et le serveur RADIUS sont authentifiés par l'utilisation de la procédure HMAC-MD5[58] associée par un secret partagé. Certaines architectures s'appuient sur IPSec pour renforcer la sécurité du lien entre le NAS et le serveur d'authentification.

Dans les réseaux 802.1X sans fil, le point d'accès joue le rôle du NAS: il transmet les informations d'authentification (accréditation) fournies par la station vers le serveur RADIUS qui joue le rôle du serveur d'authentification 802.1X. Dans ce cas, le protocole RADIUS transporte le message EAP. Le transport quasi transparent EAP par RADIUS permet de mettre en place une architecture générique indépendante de la méthode d'authentification utilisée par les ISP.

3.3.2.1 Scénario d'une session d'authentification EAP avec RADIUS

Le scénario d'une session d'authentification EAP peut se dérouler comme suit :

- Le NAS transmet au serveur RADIUS le message *EAP-Response.Identity* dans un *Access-Request*.

[54] Authentication, Authorization and Accounting
[55] Lightweight Directory Access Protocol
[56] Internet Service Provider
[57] Network Access Servers
[58] Hash-based Message Authentication Mode-Message Digest 5

- Le serveur RADIUS recherche l'utilisateur dans la base de données de clients. Il obtient le type du scénario d'authentification (TLS, TTLS, MD5, etc.) et les lettres de crédits nécessaires (mot de passe, secrets partagés, certificats, etc.).
- Plusieurs paires de messages *EAP-Request*.type et *EAP-Response*.tye sont transmises par des paquets RADIUS *Access-Challenge* et *Access-Request*. Le succès de l'opération est notifié par un *Acces-Success* (et le message *EAP-Success*). L'échec est indiqué par un *Access-Reject* (et le message *EAP-Failure*).

Au bout de l'authentification, certaines méthodes EAP calculent une clé nommée Master Key (MK). La norme 801.1X ne précise pas comment ce secret est transmis depuis le serveur d'authentification vers le point d'accès. Dans les environnements Microsoft, le secret MK est un couple de clés *MS-MPPE-Send-Key* et *MS-MPPE-Recv-Key* (2 fois 32 octets). Ces éléments sont transportés par RADIUS dans le message *Access-Success* et sont chiffrés par une clé déduite du secret partagé, et du nombre aléatoire (le champ *Authenticator*) contenu dans un précédent message *Access-Request*. Avec ce mécanisme, le point d'accès choisit une clé WEP, réalise son chiffrement à l'aide de MK et délivre le résultat au *Supplicant* en utilisant une trame *EAPoL-Key* (chiffrée par *MS-MPPE-Send-Key* et signée par *MS-MPPE-Recv-Key*).

3.3.3 Les risques avec l'authentification 802.1X

La plupart des méthodes d'authentification offrent une phase d'authentification et une phase de chiffrement et d'intégrité. Etant donné que le protocole 802.1X est purement un protocole d'authentification, cette deuxième phase reste inutilisable. Autrement dit, le protocole 802.1X par sa conception n'offre aucun moyen pour l'intégrité des données. En plus, il ne résout pas totalement le problème du WEP mais il tente plutôt de réduire les risques d'utiliser des solutions basées sur le WEP.

3.3.3.1 Une authentification à sens unique

Le premier problème de 802.1X réside dans son traitement asymétrique d'authentification entre la station et le certificateur. Selon le standard, le port devient contrôlé après le succès de la phase d'authentification qui n'est pas applicable pour la station dont le port reste toujours à l'état authentifié. L'authentification à sens unique expose la station à plusieurs attaques ; notamment "*Man-In-The-Middle*" et "*Denial-of-Service*".

3.3.3.1.1 L'attaque "Man-In-The-Middle"

L'intrus agit comme un légitime certificateur pour la station et comme un client authentifié pour le réseau. En effet, dans l'architecture 802.1X, la machine d'état de la station n'accepte que les requêtes EAP du certificateur et ne lui répond que par des réponses EAP. Par conséquent, la machine d'état du certificateur n'accepte aucune requête EAP de la station. Les états de machine n'effectuent donc qu'une authentification à sens unique et un changement significatif sera donc obligatoire dans le protocole afin d'assurer l'authentification mutuelle.

L'attaque "*Man-In-The-Middle*" peut contourner tout type de mécanisme d'authentification de niveau supérieur et le rend inefficace quel qu'il soit. En effet, tous les mécanismes d'authentification se terminent par une notification de succès ou d'échec envoyée du certificateur à la station à l'aide du message *EAP-Success* ou *EAP-Echec*. Malheureusement, ce message ne contient aucune information qui conserve son intégrité et donc il est possible pour l'intrus de forger son propre message *EAP-Success* et de se subtiliser au certificateur.

3.3.3.1.2 Le détournement des sessions

Le standard 802.1X propose une architecture robuste pour la sécurité du réseau appelée RSN[59] reposant sur 802.1X pour l'authentification et la distribution de clés et fournit un contrôle d'accès basé sur une authentification forte de la couche supérieure. Puisque l'authentification des couches supérieures n'aura lieu qu'après l'association/réassociation RSN, deux machines d'états doivent être créées : une pour 802.1X et une autre pour RSN. A cause de l'absence de communication entre les deux machines d'états et le manque de l'authenticité des messages, un intrus peut détourner la session en tirant profit de la faille de synchronisation entre les deux machines d'états. On suppose qu'il n'y a pas de chiffrement de données entre le client et le point d'accès légitime, l'attaque peut se dérouler comme suit :

- Phase 1 : la station s'authentifie auprès du certificateur en utilisant une méthode d'authentification EAP.
- Phase 2 : l'intrus envoie un message de gestion "802.11 MAC *disassociate*" en utilisant ("*Spoofing*") l'adresse MAC du certificateur. Ce message a pour but de dissocier la station et donc de changer l'état de machine de RSN à l'état *Unassociated* alors que l'état de machine de 802.1X du certificateur reste toujours à l'état *authenticated*.
- Phase 3 : l'intrus reprend le port authentifié du certificateur en "*Spoofing*" l'adresse MAC de la station.

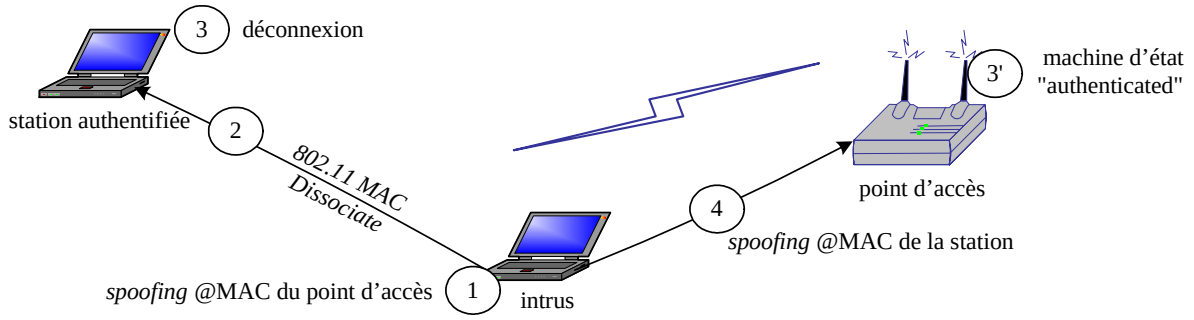

Figure 3.6. Détournement de la session

3.3.3.1.3 L'attaque "Denial-of-Service"

Parmi les attaques qui ont toujours lieu avec 802.1X ; nous rencontrons l'attaque Déni de Service (DoS) [DoS]. Le but de cette attaque est d'empêcher l'accès aux ressources du réseau. Il peut se réaliser en engorgeant le réseau avec du trafic inutile afin d'étrangler sa capacité de servir et de répondre aux besoins de ses légitimes utilisateurs.

Avec les réseaux 802.11 sans fil, DoS peut se réaliser facilement ([BSWN], [WSec]). Il peut intervenir à plusieurs niveaux ; notamment au niveau physique et au niveau de liaison. En effet, comme tout réseau mobile, les fréquences radio de 802.1X peuvent dépasser plus ou moins largement le bâtiment dans lequel se trouve l'antenne d'émission (point d'accès), et donc l'attaque DoS peut s'effectuer facilement. Ce qui permet à l'intrus d'avoir accès au moyen de transport du réseau.

- Au niveau physique, les spécifications 802.11 PHY définissent une collection de fréquences pour la communication. Les clients sont contraints de respecter cette collection et de réserver

[59] Robust Security Network

des canaux entre 2,4 et 2,484 GHz. Plusieurs appareils mobiles peuvent émettre à cette fréquence et sont capables d'engendrer du bruit, dégrader la qualité des transmissions et perturber le réseau. Cette attaque peut se réaliser avec un faible investissement de temps et d'argent ([BSWN], [WSec]).

- Au niveau liaison, le DoS a toujours la chance de se réaliser même si le chiffrement WEP est activé par le point d'accès. Comme dans l'attaque de détournement de session, le DoS peut toujours dissocier la station et le point d'accès. Un autre exemple peut se réaliser durant la phase d'authentification EAP dans lequel le DoS commence à envoyer une large quantité de messages *EAP-Start* et/ou *EAP-Response*.
- Au niveau réseau, le réseau peut permettre à n'importe quel utilisateur de s'associer avec ses points d'accès puisque 802.11 utilise un moyen de communication partagé. Le DoS peut saturer le réseau par du trafic envoyé au point d'accès et donc le rendre inefficace pour ses utilisateurs associés [BSWN].

3.3.4 Les perspectives de 802.1X

Le standard 802.1X a été conçu afin de protéger l'infrastructure de réseau Wi-Fi et non plus le terminal du client. Nous avons montré comment un intrus peut perturber la capacité du réseau à plusieurs niveaux et le rendre incapable de fournir les services commandés par ses utilisateurs.

Conçu pour sécuriser les réseaux 802.11 fixes, les protocoles EAP et 802.1X ne répondent pas aux besoins de sécurité du réseaux Wi-Fi et ils deviennent inefficaces face à plusieurs attaques ; notamment celles qui s'adressent à l'infrastructure du réseau. Une raison majeure de cette faiblesse réside dans la séparation entre la machine d'états de 802.1X et celle du RSN.

Plusieurs solutions ont été proposées afin de répondre à ces failles de sécurité. Des solutions proposent des changements dans les messages de données et de gestion en ajoutant des champs pour l'authenticité et l'intégrité afin de les protéger contre le détournement de session. D'autres solutions proposent d'ajouter un champ d'intégrité aux messages EAP identique à celui utilisé par le protocole RADIUS dans le but d'empêcher l'attaque "*Man-In-The-Middle*".

Actuellement, une nouvelle version du protocole EAP est en cours de construction par le groupe de travail EAP de l'IETF. Cette version s'occupera de la plupart des problèmes évoqués ; notamment l'authentification mutuelle, l'intégrité et la protection de la réponse de notification de la phase d'authentification (échec ou succès).

3.4 Le protocole 802.11i

Nous avons précédemment souligné les faiblesses du protocole WEP. Nous avons montré que le protocole 802.1X définit un cadre pour l'authentification mais ne spécifie pas en détails la méthode de distribution des clés. Ainsi, l'authentification 802.1X peut être la cible de plusieurs d'attaques.

Ajoutons à cela que le terminal ne participe pas au calcul de la clé globale, il n'y a pas de procédure de l'authentification mutuelle entre le client et le point d'accès mettant à profit l'existence d'une clé partagée (la clé *Unicast*).

Le groupe de travail IEEE 802.11i [802.11i] a pour but d'étudier une architecture destinée à combler ces lacunes. Bien que ce standard ne soit pas encore finalisé, un comité a édité une recommandation (WPA) basée sur un sous-ensemble de standard émergent.

Nous classerons les apports de 802.11i en trois catégories :

- définition de multiples protocoles de sécurité radio,
- éléments d'information permettant de choisir l'un d'entre les protocoles définis,
- nouvelle méthode de distribution de clés.

Ce standard s'appuie sur les réseaux sans fil 802.11 et utilise 802.1X pour l'authentification et le calcul d'une clé maître nommée PMK[60]. Cependant, dans le cas du mode *ad-hoc*, cette clé baptisée PSK est distribuée manuellement. La hiérarchie des clés cryptographiques est définie par [802.11i].

3.4.1 Protocoles de sécurité radio

Avec le standard 802.11i, trois protocoles de sécurité sont proposés : WEP, TKIP et CCMP.

3.4.1.1 TKIP (Temporary Key Integrity Protocol)

Afin d'améliorer les chiffrement des données, le standard 802.11i introduit le protocole TKIP, le successeur du protocole WEP. TKIP fournit une amélioration importante au chiffrement. Il ajoute un champ d'intégrité appelé MIC à chaque SDU MAC et génère des clés WEP dynamiques utilisant un mécanisme de ré authentification dynamique fourni par 802.1X.

TKIP utilise toujours le protocole RC4 et remédier aux limites du WEP ; notamment le re-jeu des paquets hors séquence (limite les attaques de type re-jeu), la mise en veille et le changement de clé en cas de soupçon de paquet forgé et le rafraîchissement dynamique de clés. En plus, il utilise une fonction cryptographique de chiffrement qui a comme paramètres d'entré l'IV et la clé afin d'obtenir la graine de RC4. Cette fonction permet alors de lutter contre le problème des clés faibles de CR4.

Ce protocole conserve l'architecture de sécurité de 802.11 et il fournit un niveau tolérable de sécurité [Urien]. Il peut être implémenté par une actualisation du software utilisé mais il n'assure pas la protection optimale. En conséquence, l'effet de *roaming* d'une station d'un point d'accès à un autre nécessite une ré authentification 802.1X pouvant être lente afin de supporter des applications en temps réel comme la voix sur IP. Une autre question se pose lorsqu'on veut détendre l'architecture de rafraîchissement des clés dans des réseaux en mode *ad-hoc*. L'actuelle architecture assume la structure d'une session orientée. C'est-à-dire, une association initiale est utilisée pour synchroniser la clé master et les clés de chiffrement. En plus, 802.1X assume l'authentification du serveur. Aucun de ces mécanismes d'authentification n'est disponible dans un réseau en mode *ad-hoc*. Nous notons finalement que TKIP ne peut pas offrir les fonctions de la distribution des clés en mode *multicast/broadcast* sans avoir un serveur d'authentification.

3.4.1.2 CCMP (Counter-Mode/CBC-MAC)

CCMP est une solution proposant des systèmes de chiffrement plus solides que le RC4 du WEP. Ce protocole utilise l'algorithme de chiffrement AES en mode CCM et une signature MIC. Il combine le mode CTR pour la confidentialité et le CBC-MAC pour l'authentification et l'intégrité.

---

[60] Pairwise Master Key

CCMP est plus fiable que TKIP. En effet, CCMP nécessite l'utilisation des nouveaux hardwares et se libère ainsi des contraintes des hardwares WEP. Il définit un vecteur d'initialisation de 48-bits utilisé comme un numéro de séquence contre les attaques de type re-jeu.

AES et CCMP ont un grand avantage par rapport à RC4 et TKIP. En effet, AES dans sa construction rend inutile l'utilisation des clés par paquet et permet ainsi au protocole CCMP d'utiliser directement les clés AES afin de fournir les services de sécurité.

3.4.2 Eléments d'information

Un point d'accès diffuse dans ses trames des éléments d'information (IE) afin de notifier à la station les indications suivantes [Urien] :

- la liste des infrastructures d'authentification supportées (typiquement 802.1X).
- la liste de protocoles de sécurité disponibles (TKIP, CCMP, etc.).
- la méthode de chiffrement pour la distribution d'une clé de groupe (GTK).

La station 802.11 notifie son choix par un élément d'information inséré dans sa demande d'association.

3.4.3 Distribution de clés

Nous avons donné des exemples sur la dérivation des clés de chiffrement et d'authentification après le succès de la phase d'authentification EAP. A l'issu de cette phase, le client et le serveur d'authentification partagent une clé PMK. Cette clé est délivrée par le serveur au point d'accès utilisant le protocole RADIUS. A l'aide d'un protocole à quatre passes (*4-way Handshake*) transporté par des trames *EAPoL-Key*, le client et le point d'accès déduisent une clé PTK[61]. Cette clé est calculée par la fonction PRF avec comme arguments d'entrée les nombres aléatoires (*ANonce* et *SNonce*) fournis par le client et le point d'accès, le secret partagé PMK et les adresses MAC du client et du point d'accès.

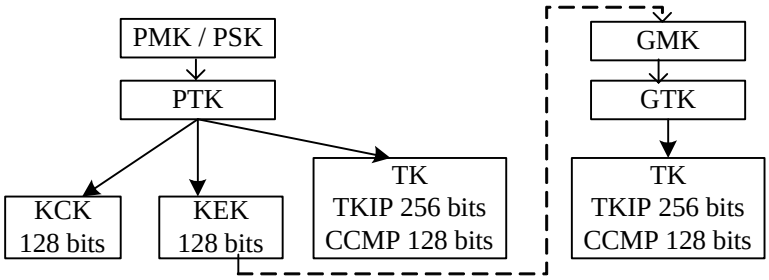

Figure 3.7. Hiérarchie des clés avec 802.11i

La valeur PTK se décompose en plusieurs sous clés : la sous clé KCK assurant la signature des messages *EAPoL-Key*, la sous clé KEK associée au chiffrement de la clé GMK et la sous clé TK utilisée pour la sécurité des trames de données.

Le point d'accès dispose d'une clé de groupe GMK. Un protocole à deux passes (2-way *Handshake*) permet de délivrer la valeur GMK (chiffrée par KEK) et de déduire, à l'aide d'un nombre aléatoire *GNonce*, une clé temporaire de groupe GTK. Au terme de cet *Handshake*, le client peut commencer à envoyer les données chiffrées en mode *multicast/broadcast*.

[61] Pairwise Transient Key

3.5 IPSec et WiFi

Plusieurs méthodes d'authentification utilisent les infrastructures à clé publique PKI afin de construire les chaînes de confiance entre les utilisateurs et les serveurs d'authentification. Ces PKIs peuvent ensuite être réutilisées afin d'établir un réseau privé virtuel (VPN[62]). Cet usage est raisonnable puisque plusieurs méthodes EAP n'assurent que l'authentification et non pas la confidentialité (exemple, EAP-TLS).

VPN est largement utilisé pour fournir aux utilisateurs un accès à ses réseaux d'une manière sécurisée en traversant un réseau publique non sécurisé comme l'Internet. Dans cette architecture, le VPN établit un canal protégé entre l'utilisateur et son réseau.

Dans ce contexte, plusieurs protocoles comme PPTP[63] [PPTP] et L2TP[64] [L2TP], peuvent être utilisés en conjonction avec un serveur centralisé d'authentification comme RADIUS afin d'établir le canal.

Dans les réseaux mobiles comme le WLAN, nous pouvons toujours utiliser le canal VPN. Dans ce contexte, le réseau non sécurisé devient le réseau mobile et les points d'accès sont configurés en mode *Open System Authentication* sans le chiffrement WEP. Ces points d'accès doivent être isolés du réseau d'entreprise de l'utilisateur par le serveur VPN et ils peuvent être connectés entre eux par un réseau virtuel LAN. L'authentification et le chiffrement sont établis au-dessus du réseau sans fil entre le client et le serveur VPN. Dans ce cas, le serveur VPN joue le rôle d'un pare-feu ou d'une passerelle entre le client et le réseau interne.

La connexion VPN peut s'établir en utilisant le protocole IPSec. Ce dernier est un protocole destiné à définir une extension de sécurité pour le protocole IP afin de permettre aux extrémités d'une connexion de sécuriser ses échanges (confidentialité, intégrité, authentification) et d'être protégée contre la plupart des attaques (*Man-In-The-Middle*, re-jeu, etc.). IPSec fournit ses services de sécurité en utilisant deux extensions d'IP :

- AH[65] assure l'authentification du *datagrammes* IP sans les chiffrer. Le principe de cette extension est d'adjoindre au *datagramme* IP un champ supplémentaire qui permet au récepteur de vérifier l'authenticité des données transportées par le *datagramme* IP. Cette extension permet aussi une protection contre le re-jeu en ajoutant un numéro de séquence.
- ESP[66] est l'extension qui assure la confidentialité et optionnellement l'authenticité des données. Son principe est de générer pour chaque *datagramme* IP, un autre *datagramme* dans lequel les données et l'en-tête original sont chiffrés. Elle assure, comme AH, l'authenticité de données et la protection contre le re-jeu.

Le protocole IPSec utilise deux modes de protection :

- le mode transport qui permet de protéger le trafic entre deux machines (postes client, serveurs) sans toucher l'entête d'IP,

[62] Virtual Private Network
[63] Point-to-Point Tunneling Protocol
[64] Layer Two Tunneling Protocol
[65] Authentication Header
[66] Encapsulating Security Payload

- le mode tunnel qui fournit un moyen d'encapsulation de chaque paquet IP dans un nouveau paquet. Ce mode porte sur tous les champs du paquet IP avant d'être encapsulé dans le tunnel, il est utilisé par les équipements réseaux tels que les routeurs.

La première version d'IPSec a été développée au sein du groupe de travail de l'IETF. Plusieurs standards sont encore publiés et intégrés dans les produits de plusieurs fournisseurs mondiaux. Toujours à l'IETF, un groupe de travail nommé PANA a été construit afin de spécifier l'utilisation du protocole IPSec dans les réseaux 802.11 sans fil. Ce groupe est en train de développer un protocole pour l'authentification du client par le point d'accès du réseau en utilisant le protocole IPSec. L'objectif de ce protocole sera ainsi d'établir une association de sécurité entre le terminal et le serveur après le succès de la phase d'authentification.

3.5.1 IPSec/VPN remplace-t-il 802.1X ?

Bien que le protocole 802.1X offre un cadre de chiffrement au niveau 2 (liaison) grâce à WEP et/ou à TKIP (WPA) [WEP-WPA] , le réseau VPN utilisant IPSec peut offrir un chiffrage au niveau 3 entre la station mobile et la destination. Avec EAP et dans le cas où le NAS supporte IPSec, il est très favorable d'abandonner les services de sécurité fournis par la couche applicative du RADIUS et d'exécuter RADIUS sur IPSec en mode ESP *non-null transform* [RFC3162].

Malgré les avantages de la solution IPSec/VPN, son utilisation à la place de 802.1X entre une station mobile et un agent dans le réseau apparaît comme un faux pas puisque :

- la norme 802.1X offre une protection au niveau 2 en chiffrant le canal avant qu'un utilisateur mobile soit authentifié et ait une adresse IP. Dans le cas d'IPSec, le chiffrement sera établi au niveau 3 après l'association entre le client mobile et le point d'accès et après l'authentification. L'obtention d'une adresse IP du serveur DHCP avant l'authentification du client (l'absence de 802.1X) donne la possibilité de réaliser plusieurs attaques ; notamment l'attaque "*Man-In-The-Middle*".
- l'IPSec ne standardise pas le moyen pour supporter la transmission de trafic en mode *broadcast/multicast*. A son origine, IPSec a été construit pour les liens point à point. En plus, IPSec ne défini pas les fonctions de la distribution de clés en mode *multicast/broadcast*.
- les clients peuvent changer leur point d'accès à n'importe quel moment. Durant leur mobilité, les clients peuvent s'associer avec un point d'accès dans un autre sous réseau (*subnet*). Cette étape oblige les clients mobiles à se ré authentifier dans le cas où ils gardent la même adresse IP (obligation d'utiliser le NAT[67] [NAT]). Si l'adresse IP change, les clients doivent achever une authentification complète car la session établie dépend de l'adresse IP spécifique. Ce qui augmente donc la charge au niveau du serveur d'authentification et des clients mobiles.
- l'utilisation de l'IPSec requière une puissance significative de traitement nécessaire pour établir un canal VPN par un terminal mobile. Dans 802.1X, le chiffrement WEP utilise RC4 et il est achevé par un co-processeur intégré dans la carte adaptée sans fil, il n'ajoute donc aucune charge cryptographique aux processeurs du mobiles. Cependant, IPSec utilise DES [DES] et 3DES [3DES] comme algorithmes de chiffrement symétrique, ils sont plus coûteux en temps d'exécution que RC4. En plus, IPSec n'est pas intégré dans de cartes adaptées sans fil mais il utilise directement le CPU du terminal mobile et du serveur. Ce qui oblige les

[67] Network Address Translation

serveurs fréquemment accessibles d'ajouter des accélérateurs hardware ou software afin de réduire le coût du calcul cryptographique exigé par l'IPSec.

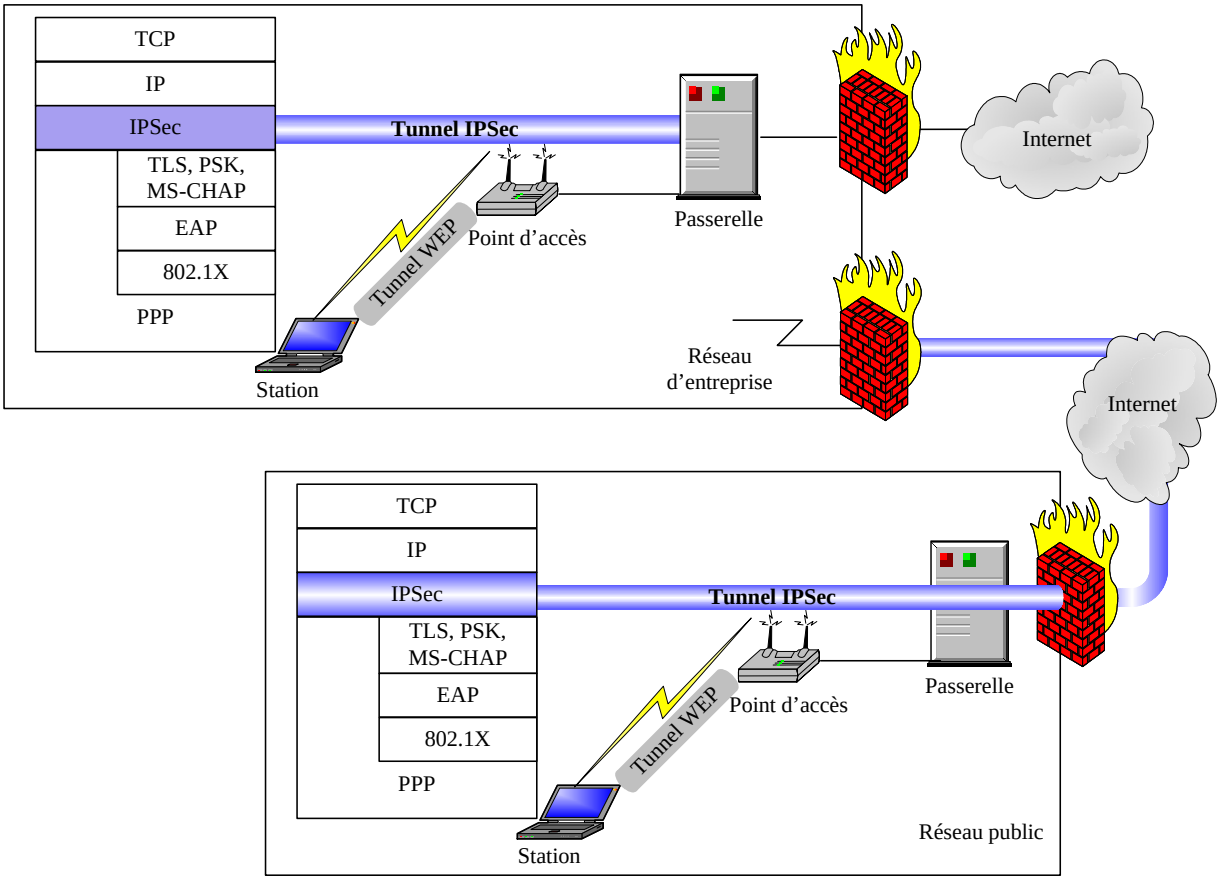

Figure 3.8. VPN avec les WLAN publiques et privés

Il reste à noter que Microsoft a publié le protocole PPTP afin d'accéder à un réseau privé par l'intermédiaire de l'Internet ou d'un autre réseau public au moyen d'une connexion à un réseau privé virtuel VPN. Développé en tant qu'extension du protocole PPP, PPTP lui confère un niveau supplémentaire de sécurité et de communications multi protocoles sur Internet. PPTP [PPTP] utilise MPPE[68] afin de crypter les données des connexions VPN PPTP. Cette solution est moins robuste qu'IPSec. En effet, *PPTP-MPPE* ne définit aucun mécanisme de génération dynamique de clés. De plus, ce protocole n'a pas l'équivalent de TLS dans 802.1X ou d'IKE dans l'IPSec. Sans la génération dynamique de clés, PPTP souffre de l'attaque par dictionnaire. D'ailleurs, il n'ajoute aucun champ d'intégrité (exemple, le MIC[69]) aux messages et donc un intrus peut modifier le contenu et donc réaliser les attaques *Man-In-The-Middle* et re-jeu.

3.6 Conclusion

Les services fournis par les réseaux mobiles sont confrontés à des véritables problèmes de sécurité. Plus particulièrement, la partition du support de transmission et la mobilité des

[68] Microsoft Point-to-Point Encryption
[69] Message Integrity Check

utilisateurs rendent les réseaux mobiles plus sensibles à plusieurs attaques et elles introduisent plusieurs failles de sécurité.

Dans ce chapitre, nous avons présenté les failles de sécurité liées aux machines d'états de 802.11 et 802.1X qui ne sont pas corrélées et aux machines d'états 802.1X qui ne fournissent pas une authentification mutuelle. Nous avons montré ensuite les problèmes du protocole WEP, une tentative échouée pour la distribution de clés et pour assurer la confidentialité et l'authentification entre le mobile et le point d'accès.

Nous avons analysé également la norme 802.1X et ses limites en terme de résistance contre plusieurs attaques telles que MitM et DoS dont le véritable problème est le manque d'authentification mutuelle. Nous avons présenté également une étude critique de plusieurs méthodes d'authentification basées sur TLS en terme de conception et de coût d'application.

Ce chapitre nous a amené à bien préciser les besoins des réseaux 802.11 sans fil. Parmi ces besoins, nous citons l'intégrité, la protection contre DoS et MitM, l'anonymat, la protection des messages du notification, la ré authentification rapide et la nécessité de sécuriser la mobilité de niveau IP (entre domaines). Ces besoins nécessitent la définition de nouvelles méthodes d'authentification intégrant l'introduction des équipements plus robustes et plus sécurisés utilisés dans le stockage de clés secrètes et pour la mobilité ; notamment la carte à puce.

Partie II : La sécurisation des échanges WAP

Chapitre 4

Solutions de sécurité pour WAP basées sur TLS

Résumé du contenu

4.1 Introduction

La première partie du présent chapitre résume les travaux réalisés dans le cadre du projet SWAP. Ce projet propose plusieurs services sûres, en utilisant une pile WAP ; notamment l'envoi des mails signés et le téléchargement par un mobile, depuis un serveur HTTP publiquement accessible, de fichiers préalablement signés (off line) par un organisme de confiance.

Dans la deuxième partie, nous présentons deux solutions afin de répondre à la faille de sécurité des versions 1.x de la pile protocolaire WAP. La première proposition est réalisée en ajoutant des extensions au protocole WTLS utilisé entre le client et la passerelle alors que la deuxième solution est assurée par l'introduction d'un serveur de distribution de clés. Une étude analytique comparative avec d'autres solutions montre les avantages des nos propositions.

4.2 La sécurité au niveau de la couche applicatif du WAP

Nous avons travaillé sur SWAP (Sécurité dans WAP), un projet RNRT sous la direction de l'ENST [SWAP]. Ce projet porte sur l'intégration, dans les terminaux mobiles GSM, GPRS et UMTS, dans les modules d'identité d'abonnés associés (SIM [SIM], USIM, WIM [WIM]) et dans les serveurs/passerelles applicatifs du monde mobile, de nouveaux mécanismes, des protocoles et des algorithmes de sécurité indispensables pour qu'au-delà de leurs fonctions traditionnelles de protection de l'accès au réseau de l'opérateur (authentification, chiffrement du tronçon radio), les combinés sans fil permettent d'accéder de façon sûre à des serveurs WAP et Web et de réaliser de nouveaux types de transactions sécurisées (paiement, réservation, commande, vote, télé action, etc.).

Le projet a comporté dans sa première phase l'analyse critique et l'implantation d'un ensemble cohérent d'éléments de spécifications de sécurité issues du forum WAP ; notamment le protocole (W)TLS, le module WIM et la bibliothèque *WMLScript* Crypto.

Avec l'évolution de WAP1.x à WAP2.0, le projet SWAP a été réorienté vers la couche d'application pour deux raisons fondamentales. D'une part, le protocole WTLS n'offre pas de sécurité de bout en bout entre un mobile et un serveur. D'autre part, la couche d'application offre une extension cryptographique qui permet de signer/vérifier et de chiffrer/déchiffrer un document donné avec une chaîne complète (du mobile au serveur d'application).

4.2.1 La couche d'application et la bibliothèque crypto *WMLScript*

Les spécifications de Forum WAP définissent la bibliothèque *WMLScript* Crypto [WMLSC] pour un environnement WAP plus sécurisé. Dans cette bibliothèque, deux fonctions sont disponibles : *EncryptText* et *SignText*. La première permet de signer une portion de texte. Cette opération peut être faite via l'utilisation du module WIM et permet d'authentifier le client auprès du serveur final. La deuxième nous permet de chiffrer les données afin de s'assurer de leur intégrité.

4.2.1.1 La fonction *EncryptText*

WapEnvelopedData est un format de chiffrement de données. Il est associé à la fonction *encrypt()* et il contient les informations indispensables au déchiffrement de données : les algorithmes utilisés, le document chiffré, les informations d'identification de la clé publique utilisée pour chiffrer la clé de session et le cryptogramme contenant la clé de session chiffrée sous cette clé publique.

Nous utilisons l'algorithme RSA pour la distribution de clé et l'algorithme 3-DES pour le chiffrement symétrique.

4.2.1.2 La fonction *SignText*

4.2.1.2.1 Format de données signées - SignedContent

Les normes du Forum WAP définissent un format propriétaire, noté *SignedContent*, pour la transmission de données signées en provenance ou à destination d'un terminal WAP. A la différence des normes de l'IETF (Internet), tel que PKCS#7[70] [PKCS7], ce format est très léger et facile à implémenter.

Les données contenues dans ce nouveau format sont de quatre sortes :

- Une signature
- Des informations sur les données signées (le message est optionnellement inclus)
- Des informations sur le signataire (valeur nulle, condensât de la clé publique ou du certificat)
- Des attributs (heure GMT ou valeur aléatoire).

La signature ainsi obtenue ne porte que sur le message à signer. Les informations complémentaires du format *SignedContent* (informations sur le signataire, attributs) ne sont pas protégées.

[70] Public Key Cryptography Standard #7

Il est avantageux d'inclure le message initial dans les données signées car alors, le format *SignedContent* suffit pour transmettre un message et la signature associée.

Un exemple du format *SignedContent* est donné ci-dessous :

```
Struct {
Uint8                  version;
// 0x01h
      Signature              signature;
// stocke la signature
      SignerInfo             signer_info<0..2^16-1>;
// stocke les informations relatives au signataire
      ContentInfo            content_info;
// stocke les informations relatives au contenu qui est signé
AuthenticatedAttribute  authenticated_attributes<0..255>;
// stocke des informations supplémentaires concernant le signataire
     } SignedContent;
```

Nous citons ici que nous avons également le champ *algorithm* (sur un octet) qui contient le type de l'algorithme utilisé pour la signature. Pour l'instant, la norme en définit 3. Dans l'exemple ci-dessus, il a pour valeur 0x01h, c'est-à-dire rsa_sha_pkcs1. Ensuite, nous retrouvons la signature sous forme d'opaque vecteur, pour laquelle la taille est indiquée sur 2 octets. C'est donc une signature de 128 bits. La fabrication de la signature se fait suivant le standard PKCS#1 [PKCS1] et elle se décompose en 3 étapes.

La première étape consiste à calculer le condensât du message (le texte que nous voulons signer) en utilisant l'algorithme SHA-1 [SHA1]. Le condensât obtenu a une taille de 20 octets. La seconde étape consiste à intégrer ce condensât dans une structure DER [ASN-DER] décrivant une structure XDR *DigestInfo*. Le tout forme alors une chaîne de 35 octets. Cette chaîne est ensuite chiffrée avec la clé privée de l'utilisateur. Cela donne la signature que nous intégrons dans le champ signature.

4.2.1.2.2 Détail de la fonction *SignText*

Cette fonction est conforme au document [WMLSC] :

signedString = Crypto.SignText(stringToSign, options, keyIdType, keyed)

Elle permet à l'utilisateur de signer un message. L'utilisateur doit entrer le code confidentiel pour activer la clé de signature de la carte WIM. Ce code doit être saisi pour chaque signature générée.

Un message au format base-64 est alors généré, il contient un message *SignedContent*. Les propriétés du message *SignedContent* généré sont:

Signature Algorithms	RSA/SHA selon PKCS #1 v1.5 RSA/SHA selon PKCS #1 v1.5, compatible PKCS #7
Signer Info Types	Implicit (the signer is implied by the content) The SHA-1 hash of the public key X509, WTLS and URL certificates are not supported.
Content Types	Text
Content Present	TRUE FALSE
Authenticates Attributes	Signer nonce

Tableau 4.1. Les propriétés du message *SignedContent*.

4.2.1.3 La fonction *Verify*

La fonction *Verify* n'est pas spécifiée par le Forum WAP. Dans le cadre du projet SWAP, nous avons choisi l'interface :

authenticatedString = Crypto.verify(signedString)

Elle permet à l'utilisateur de vérifier l'intégrité d'un message signé, d'afficher l'identité du signataire, et suivant le type de contenu :

- d'afficher le texte signé à l'écran,
- de sauvegarder les données signées dans un fichier (cas d'un exécutable).

Les ressources WIM ne sont pas utilisées par cette fonction parce que la vérification de signature est faite dans le mobile. Les propriétés du message *SignedContent* sont illustrées dans le tableau suivant.

Signature Algorithms	RSA/SHA selon PKCS #1 v1.5 RSA/SHA selon PKCS #1 v1.5, compatible PKCS #7
Signer Info Types	The SHA-1 hash of the public key
Content Types	Text (il s'agit d'un message signé à afficher) Data (il s'agit de données à sauvegarder dans un fichier)
Content Present	TRUE Les données sont limitées à 64 Ko.
Authenticates Attributes	1) Signer nonce : valeur aléatoire. 2) GMT Time : temps universel.

Tableau 4.2. Les propriétés des messages *Verify*.

Dans ce cadre, les applications développées (mail signé, téléchargement des fichiers signés, etc.) manipulent différents formats de signature. Ces formats sont cités dans l'annexe B.

4.2.1.4 Le passage de *SignedContent* à PKCS#7

Rappelons que la signature d'un message au format PKCS#7 porte sur : le message à signer, un attribut (heure GMT ou valeur aléatoire) et sur des valeurs fixes, alors que dans le cas de *SignedContent*, la signature ne porte que sur le message à signer.

Un message de type *SignedContent* peut se convertir à tout moment en un message de type PKCS#7 et réciproquement. Par contre, la signature n'est pas identique d'un format à l'autre puisque les données signées diffèrent.

Pour permettre cette conversion, il est nécessaire d'enrichir la fonction *SignedContent* et d'introduire un nouveau schéma de signature côté mobile. Ce nouveau mode consiste à signer, pour le message de type *SignedContent*, les données qui seraient signées dans un message de type PKCS#7. Le reste du traitement est inchangé.

La norme *WMLScript Crypto Librairie* [WMLSC] définit des "formulaires" pour effectuer le passage d'un format *SignedContent* à un format PKCS#7. L'utilisation de ces formulaires se fait en réutilisant les champs d'un message *SignedContent* et en les intégrant dans une représentation DER d'une structure PKCS#7. La norme définit deux formulaires possibles en fonction du type *AuthenticatedAttribute* présenté dans le *SignedContent*. Elle définit aussi quatre nouveaux *Object Identifier* pour désigner d'objets du *SignedContent* en XDR [XDR] et leur codage DER. Ces objets sont :

```
Attribute          OID          OID en binaire
ContentType        pkcs9-3      2A 86 48 86 F7 0D 01 09 03
MessageDigest      pkcs9-4      2A 86 48 86 F7 0D 01 09 04
SigningTime        pkcs9-5      2A 86 48 86 F7 0D 01 09 05
SignerNonce        pkcs9-25-3   2A 86 48 86 F7 0D 01 09 19 03
```

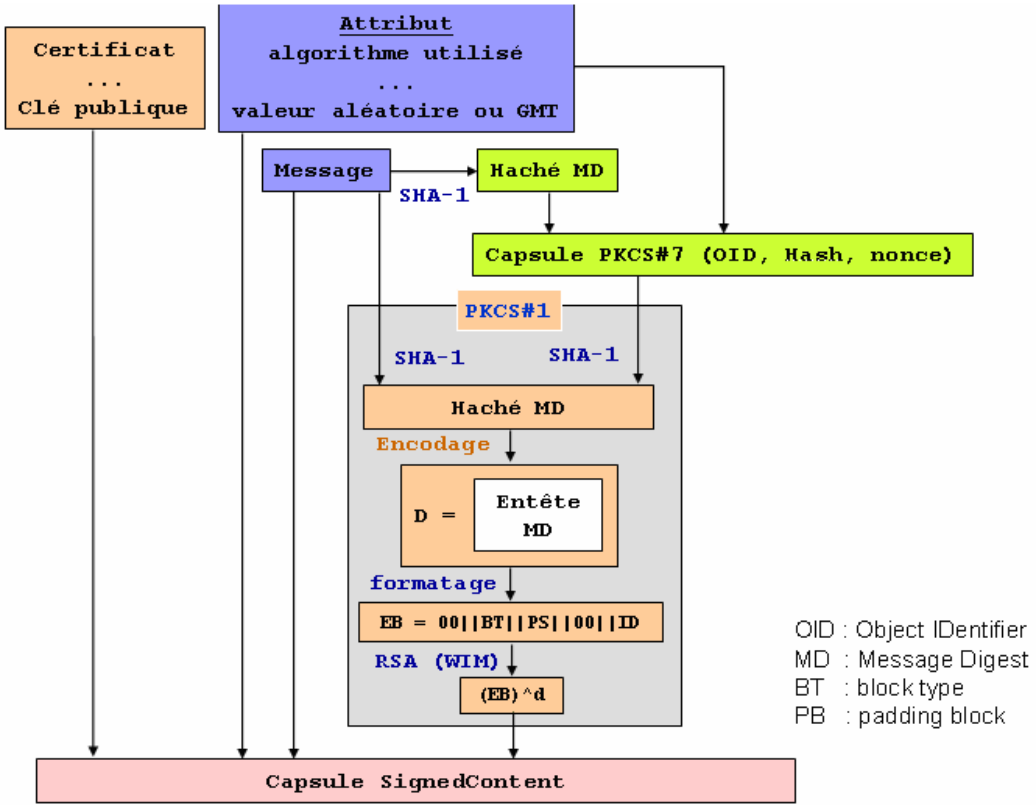

Figure 4.1. La conversion entre *SignedContent* et PKCS#7

Les formulaires changent selon le type d'attribut :

- dans le cas où l'attribute_type serait un temps gmt_utc_time, nous avons :

```
31 5D Set (93 octets)
.  30 18 Sequence (24 octets)
.  .  06 09 2A 86 48 86 F7 0D 01 09 03 – Object Identifier: contentType
.  .  31 0B Set (11 octets)
.  .  .  06 09 2A 86 48 86 F7 0D 01 07 01 – Object identifier: pkcs7-data
.  30 1C Sequence (28 octets)
.  .  06 09 2A 86 48 86 F7 0D 01 09 05 – Object Identifier: signingTime
.  .  31 0F Set (16 octets)
.  .  .  17 OD xx xx xx xx xx xx xx xx xx xx xx xx – UTCTime
.  30 23 Sequence (35 octets)
.  .  06 09 2A 86 48 86 F7 0D 01 09 04 – Object Identifier: messageDigest
.  .  31 16 Set (22 octets)
.  .  .  04 14 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
– Octet String (20 octets) SHA-1 digest
```

- dans le cas où l'*attribute_type* est une valeur aléatoire *signer_nonce*, nous avons :

```
31 59 Set (89 octets)
.  30 18 Sequence (24 octets)
.  .  06 09 2A 86 48 86 F7 0D 01 09 03   – Object Identifier: contentType
.  .  31 0B Set (11 octets)
.  .  .  06 09 2A 86 48 86 F7 0D 01 07 01 – Object identifier: pkcs7-data
.  30 18 Sequence (24 octets)
.  .  06 0A 2A 86 48 86 F7 0D 01 09 19 03 – Object Identifier: signerNonce
.  .  31 0A Set (10 octets)
.  .  .  04 O8 xx xx xx xx xx xx xx xx– Octet String: randomNonce
.  30 23 Sequence (35 octets)
.  .  06 09 2A 86 48 86 F7 0D 01 09 04 – Object Identifier: messageDigest
.  .  31 16 Set (22 octets)
.  .  .  04 14 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
– Octet String (20 octets) SHA-1 digest
```

4.2.2 Implémentation de l'application de téléchargement des fichiers signés

4.2.2.1 Implémentation sur le serveur

La mise en place d'un serveur de démonstration de distribution des fichiers signés accessibles à des PDA communicants (Sagem WA3050) comprend deux principales étapes :

- la production des capsules *SignedContent* : cette opération sera réalisée off line. Il en sera de même pour la production du cryptogramme des capsules *SignedContent* (sous une clé de session propre à chacune) si l'option de chiffrement est retenue.
- l'intégration des capsules *SignedContent* dans les pages HTML du site de distribution. Il s'agit ici de réaliser un lien vers le fichier signé au format *SignedContent* que l'utilisateur va télécharger. Dans la version de base de l'application, lorsqu'un client envoie une requête au serveur en vue de se faire envoyer un document, le serveur satisfait directement cette demande et lui envoie le fichier. Aucun traitement cryptographique n'aura donc à être effectué "on line" au moment du téléchargement d'un fichier *SignedContent*. Si l'option de chiffrement des capsules *SignedContent* est retenue, il sera par contre nécessaire (à la réception d'une requête d'envoi d'un fichier chiffré) de réaliser un traitement cryptographique spécifique au demandeur, à savoir le chiffrement RSA de la clé de session utilisée sous la clé publique du demandeur.

4.2.2.2 Implémentation dans les mobiles et les cartes WIM

Les traitements spécifiques à implanter dans les mobiles et les cartes WIM sont de deux sortes :

- un utilitaire de traitement d'une capsule *SignedContent* qui, lors de la réception d'une telle capsule, déclenchera les calculs de vérification et, en fonction du résultat, procédera ou non au stockage local du fichier signé,
- les calculs de vérification proprement dit.

4.2.2.3 Présentation du service SWAP-Mail

Le SWAP-Mail est un service Internet permettant l'échange de messages sécurisés pour les utilisateurs du SAGEM WA3050. L'accès à ce service est fait en utilisant le navigateur Internet Explorer du WA3050.

La connexion peut-être sécurisée en profitant de la sécurité intégrée au navigateur (TLS avec authentification du serveur).

Les messages échangés avec SWAP-Mail peuvent être protégés en intégrité (signés) ou/et en confidentialité (chiffrés). La protection est faite suivant deux techniques spécifiées par le Forum WAP :

- les messages *SignedContent* pour la signature,
- les messages *WapEnvelopedData* pour le chiffrement.

Une carte à puce WIM est utilisée pour le stockage et l'utilisation des clés privées de l'utilisateur. En particulier, elle permet de signer un message ou de le déchiffrer. La vérification de signature ou le chiffrement d'un message sont traités au niveau du téléphone.

Initialement prévu pour l'échange de donnée textuelle, il est possible d'utiliser cet outil pour le téléchargement de fichiers sécurisés.

Une passerelle permettant la conversion entre les formats *SignedContent*/*WapEnveloppedData* et PKCS#7 est envisageable, ce qui permettrait d'assurer une compatibilité avec l'ensemble des produits suivant les normes de l'IETF. En particulier, il serait possible de signer un formulaire en utilisant le SAGEM WA3050 puis de convertir le message dans un format compatible avec les normes européennes de signature électronique.

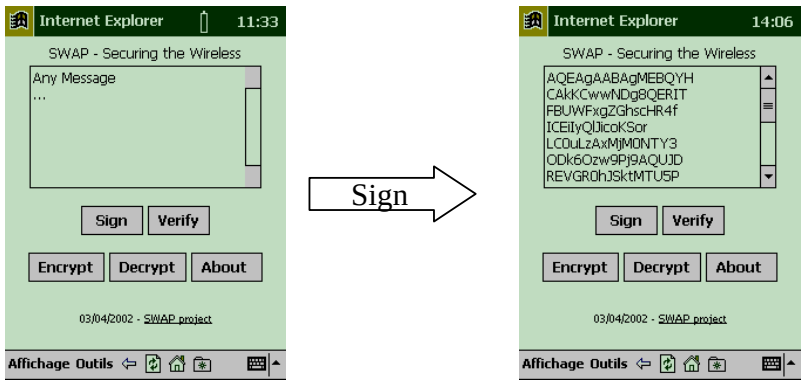

Figure 4.2. L'interface de la fonction *SignedContent*.

4.3 Extension du WTLS (E-WTLS)

Bien que la couche d'application de WAP nous permette de signer et/ou chiffrer des données, il est toujours préférable d'établir une session sécurisée et de créer un canal protégé de bout en bout indépendamment de l'application. Dans ce sens, nous proposons d'enrichir le protocole WTLS afin d'éviter le problème de la passerelle et d'assurer les services de sécurité entre l'utilisateur et le serveur destination. Dans ce sens, nous proposons le protocole E-WTLS [Bad13], une extension du WTLS afin d'avoir une sécurité de bout en bout entre le client et le serveur d'application indépendamment de la passerelle WAP. Le but de ce protocole est de :

- établir une authentification mutuelle (de bout en bout) entre le client et le serveur,
- partager une clé secrète entre le client et le serveur non visible par la passerelle,
- éviter d'avoir de messages en clair dans la passerelle,
- protéger les paires de la connexion contre les attaques par re-jeu,

- réduire la charge cryptographique du côté mobile.

Afin d'appliquer ce protocole, nous avons proposé d'introduire un tiers de confiance (*Trusted Third Party*-TTP) entre le client et le serveur et d'ajouter de nouveaux messages (*Certificate_OK*, *Client_Id*) à la phase *Handshake*. Le "*Handshake*" de notre protocole est basé sur l'*Handshake* de (W)TLS comme il est défini en détail dans les paragraphes suivants.

4.3.1 Pourquoi TTP ?

A la base, le tiers de confiance (TTP) est une passerelle WAP avec de nouvelles fonctions. Ces nouvelles fonctions sont indispensables pour plusieurs raisons ; notamment :

- une passerelle WAP traditionnelle n'assure pas une sécurité de bout en bout,
- la grande charge cryptographique nécessaire pour vérifier la chaîne de certification et valider si un certificat est révoqué ou pas,
- pour améliorer le control d'accès,
- pour réduire les efforts d'avoir un décodeur pour analyser et comprendre les certificats de type X.509.

Dans notre architecture, le TTP permet à un client WAP de récupérer le certificat du serveur à connecter. Le TTP vérifie la validité du certificat ainsi que la chaîne de certification. Ensuite, il récupère la clé publique du certificat du serveur et l'envoie au client d'une manière authentifiée et protégée.

4.3.2 Le protocole *Handshake*

Le protocole "*Handshake*" permet à un client WAP d'authentifier le serveur via le TTP et de négocier les algorithmes et les clés de chiffrement. Une fois la phase "*Handshake*" terminée, le client et le serveur partagent une clé secrète qui sera utilisée afin de construire un canal sécurisé permettant d'échanger les données.

4.3.2.1 La phase de la négociation des paramètres de sécurité

La phase "*Handshake*" commence avec le client en envoyant le message *ClientHello* au TTP. Ce message contient, entre autre, une valeur aléatoire utilisée comme une entrée au processus de la génération de clés (PRF) et une liste de *cipher_suites* supportés par le client dans un ordre décroissant de préférence. Le client spécifie l'adresse du serveur à connecter dans le champ *server_address*. Il ajoute également un champ nommé *Client_Id* utilisé pour s'authentifier auprès du TTP. Ce champ peut contenir n'importe quelle information reliée à l'identité du client. Il peut contenir l'URL du certificat du client afin d'indiquer au TTP sa capacité de signer. Cela est applicable dans le cas où le serveur exige l'authentification du client. En effet, si le serveur envoie une requête d'authentification au client et que le *Client_Id* reçu par le TTP ne contient pas un URL d'un certificat, le TTP envoie le message TLS "*no certificates*" au serveur. Sinon, le TTP envoie le condensât des messages *Handshake* au client pour lui permettre de s'appliquer sa signature et donc de s'authentifier auprès du serveur.

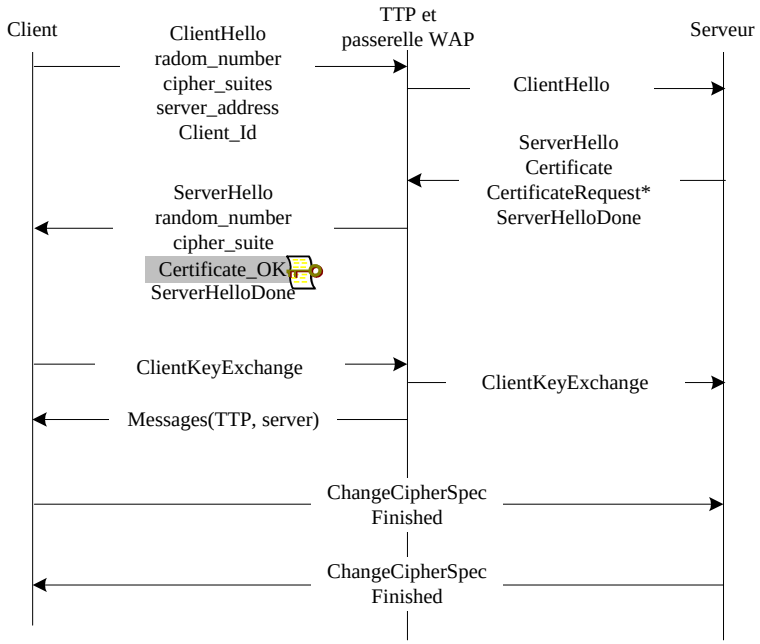

Figure 4.3. La phase *Handshake* du protocole E-WTLS

4.3.2.2 La phase d'authentification du serveur auprès du TTP

A la réception du message *ClientHello*, le TTP vérifie la signature du client. Ce qui permet au TTP de contrôler l'accès radio. Ensuite, il convertit le message *ClientHello* de son format WTLS au format TLS et délivre le résultat au serveur.

Le serveur répond par son message *ServerHello* comme il est défini dans le protocole "*Handshake*" de TLS. Ce message contient une autre valeur aléatoire et le *cipher_suite* sélectionné. Ensuite, le serveur envoie son certificat contenant sa clé publique afin de permettre au client de communiquer le secret aléatoire *pre_master_secret*. Le serveur envoie ainsi le message *ServerHelloDone* afin d'indiquer la fin de son hello.

Le certificat du serveur est confirmé par le TTP en vérifiant l'identité du serveur et la signature du CA. Ensuite, le TTP convertit le message *ServerHello* du format TLS au format WTLS et le délivre au client. Le TTP récupère la clé publique du serveur et l'envoie au client en utilisant le message *ServerKeyExchage*. A ce point, le TTP génère un nouveau message nommé *Certificate_OK*. Ce message contient le "*Distinguished Name*" et la clé publique du serveur, la valeur aléatoire du client et un bit prouvant la validité du certificat du serveur. A ce moment-là, le TTP signe le message *Certificate_OK* et l'envoie ensuite au client. L'objectif principal de ce message est de prouver au client l'identité et la validité du certificat du serveur et de protéger les applications contre l'attaque de re-jeu.

Certificate_OK = DN serveur + ClientHello.random + ServerPublicKey + IsTrue ;
IsTrue = 1 si le certificat du serveur est valide, sinon il est égal à 0.

4.3.2.3 La structuration du *Certificate_OK* en XDR

La structure de ce message en XDR est illustrée ci-dessous:

```
struct {
     Certificate_Proof CertificateProof;
```

```
        Signature Signed_Certificate_Proof;
    } Certificate_OK;

    struct {
        DistinguishedName server_name<3..2^8-1>;
        opaque client_random[32];
        select (ServerPublicKey)
            {
              case diffie_hellman:
                ServerDHParams params;
              case rsa:
                ServerRSAParams params;
             };
        IsTrue is_true
    }Certificate_Proof;
```

Signed_Certificate_Proof : contient à la fois le condensât du *Certificate_Proof* et la signature appliquée sur ce condensât.

```
md5_hash
            MD5(CertificateProof);
sha_hash
            SHA(CertificateProof);
```

Les paramètres de ce message sont définis en XDR comme suit :

IsTrue : indique si le certificate du serveur est valide ou pas.

```
enum { true, false } IsTrue;
```

DistinguishedName : transporte le nom du serveur.

```
opaque DistinguishedName<1..2^8-1>;
```

La structure du *ServerPublicKey* est :

```
enum { rsa, diffie_hellman } ServerPublicKey;
```

En cas du RSA : transporte les paramètres publiques de l'algorithme RSA du serveur.

```
    struct {
        opaque rsa_modulus<1..2^16-1>;
        opaque rsa_exponent<1..2^16-1>;
    } ServerRSAParams;

    rsa_modulus
        Le modulus de la clé RSA du serveur.

    rsa_exponent
        L'exponent publique de la clé RSA du serveur.
```

En cas du DH : transporte les paramètres publiques de l'algorithme Diffie_Hellman du serveur.

```
    struct {
        opaque dh_p<1..2^16-1>;
        opaque dh_g<1..2^16-1>;
        opaque dh_Ys<1..2^16-1>;
```

```
    } ServerDHParams;

dh_p
  Le prime modulus utilisé avec l'opération de Diffie-Hellman.

dh_g
  Le générateur utilisé avec l'opération de Diffie-Hellman.

dh_Ys
   La valeur publique du Diffie-Hellman du serveur (g^X mod p).
```

4.3.2.4 L'authentification du serveur auprès du client via le TTP

Le client vérifie la signature du TTP et confirme que les contenus sont corrects. Ensuite, il génère une valeur aléatoire appelée *pre_master_secret* et la chiffre en utilisant la clé publique du serveur. Le résultat est envoyé en utilisant le message *ClientKeyExchange*. Il est important de noter que le *pre_master_secret* est invisible au TTP puisqu'il n'a pas la clé privée du serveur.

4.3.2.5 La phase d'intégration des échanges

Le TTP envoie au client un nouveau message nommé *Messages(TTP, serveur)* contenant le condensât (utilisant MD5 et SHA-1) de tous les messages *Handshake* échangés entre le TTP et le serveur. Ce message est nécessaire puisque le client n'a pas tous les messages échangés entre le serveur et le TTP. Le condensât est utilisé afin de calculer la valeur du message *Finished*. Cette dernière est calculée comme suit :

$$Verify\_data = PRF(master\_secret, Finished\_label, MD5(Handshake\_messages) + SHA\text{-}1(Handshake\_messages))$$

La suite de la phase "*Handshake*" reste inchangée. Le client et le serveur continuent leurs échanges comme il est défini dans (W)TLS.

4.3.3 Le protocole *Record* du protocole E-WTLS

Contrairement au protocole *Record* dans les architectures WAP1.X, le protocole *Record* de E-WTLS n'utilise qu'une seule clé partagée entre le client et le serveur durant la phase "*Handshake*". Cependant, les paramètres suivants sont ajoutés à la couche *Record* du client E-WTLS : la clé publique du serveur et le *pre_master_secret* échangé entre le client et le serveur.

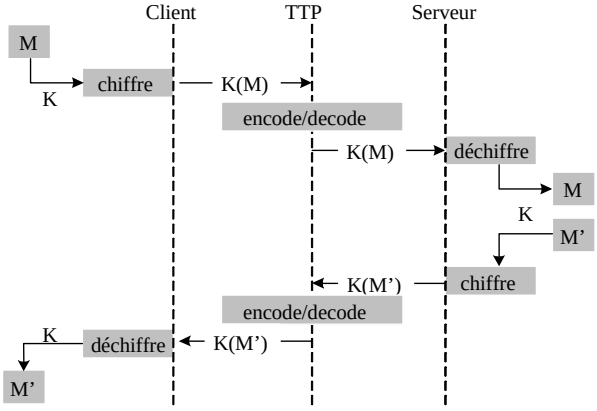

Figure 4.4. La couche *Record*

Avec E-WTLS, il n'y a aucun changement du côté serveur. En effet, les données avec E-WTLS sont chiffrées/déchiffrées de la même manière définie par TLS. Le rôle du TTP est d'encoder/décoder les données chiffrées du format WAP au format Internet et vice versa sans aucun changement dans le corps du message. Les données sont déchiffrées par le serveur (le client) en utilisant la clé secrète générée durant la phase *Handshake*.

4.3.4 Avantages et inconvénients

Le protocole E-WTLS assure les mêmes services du TLS (confidentialité, intégrité et la protection contre les attaques de re-jeu) et offre le service de l'authentification mutuelle entre le client et le serveur destinataire. En plus, ce protocole empêche la passerelle d'avoir en clair les messages échangés entre le client et le serveur et il ne nécessite aucun changement au niveau du protocole TLS entre le TTP et le serveur.

Il est vrai qu'un message erroné avec E-WTLS ne peut pas être résolu au niveau du TTP. En effet, ce message est chiffré par une clé inconnue par le TTP. Cependant, le client résout ce problème en re-demandant le message ou tout simplement en fermant la session. En plus, de nouvelles fonctions doivent être ajoutées à la passerelle afin de pouvoir gérer deux sessions en temps réel : une entre le client et la passerelle et une autre entre la passerelle et le serveur.

4.3.5 Performances

Afin d'évaluer sa performance, nous avons accompli un test pour le protocole E-WTLS afin d'estimer sa charge cryptographique durant l'établissement d'une session sécurisée. La charge cryptographique est mesurée dans le cas où le TTP et le serveur auraient le même CA. C'est-à-dire, la profondeur de l'arbre de l'infrastructure à clé publique (PKI), qui est l'équivalent du nombre de certificats dans la chaîne de certification, est supposée égal à un.

En comparant E-WTLS avec d'autres solutions de sécurité du WAP comme WAP2.0, nous avons analysé la phase "*Handshake*" de chacun de ces protocoles et nous avons estimé la charge cryptographique nécessaire pour le client afin d'ouvrir une connexion sécurisée et de chiffrer un message de 16 kilo octets. Le benchmark a été accompli sur un PC Intel avec 440 MHz et sur un iPAQ PDA Intel avec 205 MHz.

Les résultats montrent que l'établissement d'une session sécurisée avec E-WTLS nécessite 0.71 seconde, alors qu'il requiert 1.51 seconde avec TLS-WAP2.0. Cela révèle que E-WTLS arrive à réduire le temps du traitement cryptographique ainsi que la charge protocolaire du côté client. En plus, il est important d'indiquer que ce traitement cryptographique du côté client TLS-WAP2.0 augmente rapidement en parallèle avec l'accroissement de la profondeur de l'arbre du PKI (environ 1.4 seconde pour chaque certificat supplémentaire) tandis qu'il reste presque stable avec E-WTLS.

4.3.5.1 La charge protocolaire sur le réseau

Un autre facteur important dans la mesure de la performance est le temps requis par le support de transmission afin d'échanger les données reliées au protocole d'authentification. Par exemple, l'intervalle de temps (*RT, Round Trip*) entre la transmission d'un paquet de données utilisant le support GSM-CSD et la réception de l'acquittement correspondant est en moyenne de 2.438 secondes [AFG]. Comme résultat, le temps de transfert des données et le nombre de RT ont une influence considérable sur la performance globale de protocoles.

Si nous fixons le MTU (Maximum Transfert Unit) à 1500 octets, alors nous avons besoin de 3 RTs pour une session E-WTLS et 6 RTs pour une session TLS (les RTs entre le client et le TTP intègrent celles entre le TTP et le serveur). En plus, le temps de transfert de données de la phase "*Handshake*" (environ 550 octets) du E-WTLS est plus réduit que celui-ci nécessaire par le TLS (environ 6500 octets). Si nous utilisons GSM-CSD comme un moyen de transport, le temps de transfert est environ 0.35 seconde pour E-WTLS durant la phase d'authentification, alors qu'il est d'environ 4.03 secondes en utilisant TLS. Nous notons ici que le temps de transfert entre le TTP et le serveur durant la session E-WTLS est négligeable.

4.4 Une autre solution pour la sécurité dans WAP

Notre deuxième solution [Bad12] propose d'ajouter au tiers de confiance la fonction de distribution de clé d'authentification. La passerelle (implémentée normalement chez l'opérateur) agit comme un distributeur de tickets permettant à ses clients de s'authentifier auprès des serveurs d'applications.

Le ticket délivré par le tiers fournit à un client donné l'autorisation d'accéder aux domaines autorisés et le moyen de partager un secret avec le serveur destinataire d'une manière invisible à la passerelle. Ce protocole se compose de deux phases : *Application* et *Handshake*.

Il faut noter que nous pouvons partager un ticket directement avec le serveur sans passer par un tiers de confiance. Dans ce cas, le serveur joue le rôle du tiers.

4.4.1 La phase Application

La phase Application de ce protocole réutilise les fonctions fournies par la bibliothèque crypto du *WMLScript* ; notamment la fonction *SignText*. Cette phase est établie plutôt hors ligne ("off line"). Elle commence lorsque le client envoie une requête (*request*) au tiers. Cette requête contient l'URL du certificat du client, une valeur aléatoire et l'adresse du serveur à connecter. Le client peut appliquer la fonction *SignText* sur sa requête afin de s'authentifier auprès du tiers. Nous pouvons utiliser cette fonction de la manière définie dans le cadre du projet SWAP.

Le tiers vérifie la signature du client dans le cas où la requête est signée. Ensuite, la passerelle récupère la clé publique du serveur demandé par le client en utilisant un service de distribution de certificats tel que LDAP (*Lightweight Directory Access Protocol)* ou par ses propres moyens (base de données locale). Ensuite, le tiers génère un ticket permettant au client d'avoir l'accès au serveur destinataire. Ce ticket est envoyé dans une réponse contenant deux champs chiffrés :

- Le premier champ contient la valeur aléatoire envoyée par le client, la clé publique du serveur et le ticket généré. Le tiers chiffre ce champ en utilisant la clé publique du client ou une clé partagée avec le client. Par conséquent, le client peut détecter les attaques par re-jeu grâce à la variable aléatoire signée. En plus, le client peut avoir confiance au serveur via le tiers. En effet, le tiers ne doit pas envoyer la clé publique du serveur que dans le cas où le tiers admette une chaîne de confiance prédéfinie entre elle et le serveur et que le certificat du serveur soit valide.
- Le deuxième champ contient la même variable aléatoire du client et le même ticket envoyé dans le premier champ. Le tiers chiffre ce champ en utilisant la clé publique du serveur ou par une clé partagée entre le tiers et le serveur. Le résultat est signé par le tiers avant d'être envoyé au client. Le but de ce champ est d'authentifier le client auprès du serveur en comparant les valeurs aléatoires et les tickets.

Le client reçoit les deux champs. Il vérifie la signature du tiers et déchiffre, avec sa clé privée ou avec la clé partagée, la partie chiffrée afin de récupérer la valeur du ticket. Ensuite, il génère un nouveau champ nommé *TicketToEnter* contenant le deuxième champ de la réponse du tiers et d'autres paramètres chiffrés par la clé publique du serveur. Parmi ces paramètres, nous trouvons le ticket déchiffré, la même valeur aléatoire du client et la valeur aléatoire *pre_master_secret* générée par le client pour établir les clés de la session. La valeur de *TicketToEnter* sera utilisée dans la deuxième phase : la phase "*Handshake*".

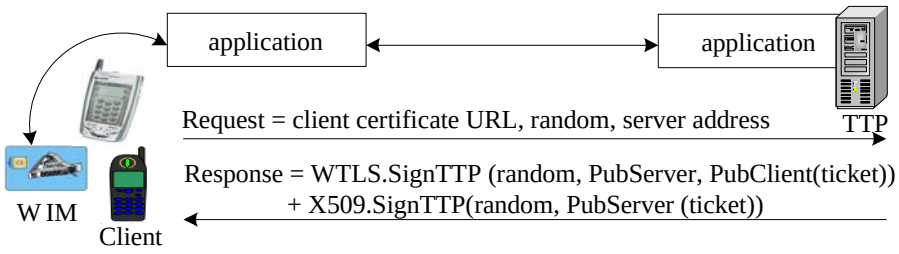

Figure 4.5. La phase Application

4.4.2 La phase *Handshake*

Avant de détailler cette phase, nous rappelons que TLS définit un autre type de "*Handshake*" : le "*Handshake*" abrégé. Ce type a été expliqué dans le chapitre 2 et il nous permet de reprendre une session sécurisée déjà établie. Dans le cas de notre architecture, nous avons décidé d'utiliser ce type de "*Handshake*" avec un changement simple dans le message *ClientHello*.

La phase "*Handshake*" commence toujours avec le client en envoyant le message *ClientHello*. Ce message contient les attributs de sécurité définis dans TLS et introduit un nouveau champ nommé *TicketToEnter*. La structure de ce message et de ses paramètres sont définis comme suit :

```
struct {
        TicketToEnter Ticket_To_Enter;
        ProtocolVersion client_version;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
       } ClientHello;

struct {
        SignedData Signed_Data;
        EncryptData Encrypt_Data;
       } TicketToEnter;
```

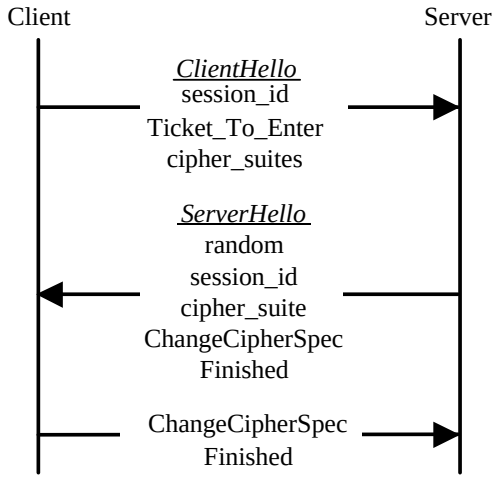

Figure 4.6. La phase *Handshake*

Quand le serveur reçoit ce message, il vérifie la signature du tiers envoyée dans *TicketToEnter*. Ensuite, le serveur déchiffre (en utilisant sa clé privée) la partie chiffrée afin de récupérer le ticket, le *client_random* et la *pre_master_secret*. Le serveur compare les tickets et les valeurs aléatoires envoyés dans les deux champs de *TicketToEnter*. Le but de cette comparaison est d'affirmer l'authentification du client. Après, le client et le serveur continuent la phase "*Handshake*" comme il est défini dans la phase abrégée du TLS. Nous notons ici que la *pre_master_secret* est utilisée dans la génération du *master_secret* et des clés de chiffrement.

4.4.3 Avantages

Cette proposition assure les mêmes services de sécurité fournis par E-WTLS et TLS. Une analyse de sa phase *Handshake* montre que ce protocole réduit le calcul cryptographique ; notamment en ce qui concerne la vérification du certificat du serveur et de la liste de révocation de certificats. En effet, le temps nécessaire pour établir une session sécurisée utilisant notre solution est équivalent à celui de E-WTLS (environ 0.76 seconde). En plus, durant la même phase, la charge des messages protocolaires est réduite d'une manière remarquable, il s'agit d'environ 500 octets. D'autre part, cette solution ne nécessite plus qu'un 1.5 RTs pour échanger les messages protocolaires entre le client et le serveur destinataire.

Il reste à noter que *Cisco Systems* se sert actuellement de cette contribution pour sa dernière méthode d'authentification EAP-FAST [EAPFast]. Cependant, Cisco suppose qu'une clé est déjà partagée entre le client et le serveur et donc la phase "Application" de notre protocole est omise.

4.5 Conclusion

La première solution définie dans ce chapitre (projet SWAP) vise à améliorer l'utilisation de la couche d'application de la pile WAP en utilisant la bibliothèque *WMLScript* Crypto afin de permettre de contrôler l'accès à des applications particulières telles que le Mail.

Nos deux autres solutions ont pour objectif d'établir des sessions sécurisées indépendamment de l'application utilisateur. Chacune d'entre elles montre les gains par rapport à TLS-WAP2.0 en terme de calcul cryptographique et de charge des messages protocolaires.

Il est clair que les solutions de sécurité présentées dans ce chapitre restent indispensables pour protéger l'accès au réseau de l'opérateur téléphonique et du fournisseur de services

(authentification, contrôle d'accès, etc.). Elles nous permettent également une combinaison entre le fixe et le sans fil avec un accès sûre à des serveurs WAP et Web ainsi qu'une réalisation de nouveaux types de transactions sécurisées (commerce électronique, réservation, etc.).

Partie III : La sécurisation des échanges 802.11 sans fil

Chapitre 5

Solutions de sécurité pour Wi-Fi basées sur TLS

Résumé du contenu

5.1 Introduction

La mise en œuvre du protocole Internet dans le cadre des applications à fortes exigences de sécurité et de mobilité implique l'utilisation des équipements sûrs comme la carte à puce. Cela s'attache à mettre en place une solution sécurisée pour permettre l'administration à distance sur ce même réseau des cartes et terminaux utilisés pour ces applications. Il convient de définir une architecture apte à garantir la sécurité et la fiabilité des données transportées de bout en bout, c'est-à-dire entre le poste d'administration, le terminal et la carte à puce. La conception de ce schéma sécuritaire s'attache à établir le cahier des charges applicable à chacun des acteurs de la chaîne (serveur, infrastructure réseau et points d'accès, terminal, carte à puce), à sélectionner et à préciser le cas d'emploi des mécanismes sécuritaires de l'Internet (apports IPv6, IPSec, IKE, TLS). Ce schéma est conçu et élaboré dans le cadre du projet RNRT-EPIS. Dans ce projet, nous embarquons un protocole sécurisé dans la carte SIM afin de permettre à un administrateur distant de traiter, modifier et de mettre à jour les données stockées dans une carte à puce (SIM).

Dans ce chapitre, nous poursuivons les travaux effectués sur les réseaux WLAN de façon à les intégrer dans le protocole de sécurité TLS, le protocole EAP et la carte à puce afin d'assurer des nouveaux mécanismes d'autorisation et de contrôle d'accès plus fiables et plus performants.

Nous proposons également une nouvelle méthode d'authentification EAP. Cette méthode couvre plusieurs services de sécurité ; notamment l'anonymat et le PFS. En plus, elle n'a pas besoin d'utiliser de certificats et de PKIs dans son mécanisme d'authentification.

Nous terminons ce chapitre avec une implémentation du protocole EAP-TLS sur la carte à puce. Son but principal est d'introduire des équipements plus robustes et plus sécurisés utilisés dans le stockage de clés secrètes et privées.

5.2 Le protocole TLS-PSK (Pre-Shared-Key)

5.2.1 Rappel sur la négociation abrégée

Le "*Handshake*" abrégé nous permet de reprendre une session sécurisée déjà établie. Ce type de "*Handshake*" commence par le client en envoyant l'identificateur de la session sécurisée à reprendre. Le serveur de son côté vérifie s'il existe une correspondance dans son cache.

Etant donné que la négociation abrégée est basée sur une phase de négociation préalable et que la négociation complète permet l'authentification à base de certificats, les spécifications de TLS suggèrent l'utilisation de la négociation abrégée dans un délai de 24 heures maximum. En effet, il est possible que le certificat utilisé dans la négociation complète devienne non valide au bout de 24 heures. En plus, un intrus ayant la clé de session peut imiter la partie compromise tant que l'identificateur de la session n'est pas retiré. De l'autre côté, les applications se déroulant dans des environnements non sûrs doivent se méfier du stockage de la clé secrète correspondante à l'identificateur de la session.

5.2.2 L'architecture de l'EPIS

Parmi les principaux objectifs du projet EPIS [EPIS] l'embarquement d'un protocole sécurisé qui permet à un administrateur distant de modifier et de mettre à jour les données stockées dans la carte à puce SIM. Un exemple d'application de l'EPIS est le réseau GSM où l'opérateur téléphonique (via le serveur OTA[71]) peut effectuer des opérations (mise à jour, ajout, suppression, etc.) destinées à la carte SIM. Dans cet exemple, l'opérateur communique avec la carte en utilisant la norme GSM 03.40 [SIM] opérée au-dessus d'un lien non fiable comme SMS. Afin d'assurer la sécurisation de l'échange, la carte et l'opérateur utilisent la norme GSM 03.48. Bien que la norme GSM 03.48 offre la protection contre le re-jeu, l'intégrité et la confidentialité de données (qui ne sont pas très efficaces), elle n'assure pas l'authentification mutuelle.

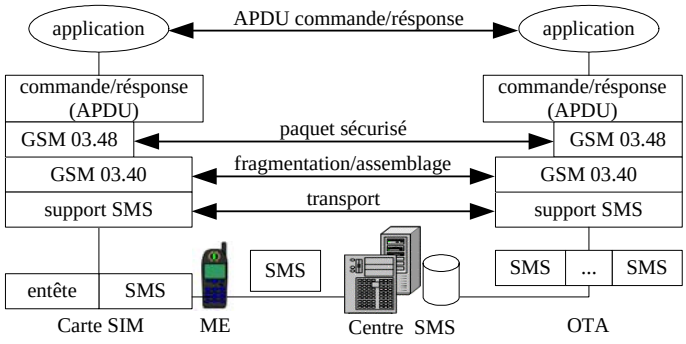

Figure 5.1. Echange des données entre le SIM et le serveur OTA.

[71] Over The Air

5.2.3 L'architecture proposée

5.2.3.1 Introduction

Dans le cas de notre architecture EPIS, nous avons décidé d'utiliser la négociation abrégée de TLS avec un stockage de longue durée de la clé secrète à l'intérieur d'une carte à puce (SIM) et sur le serveur d'authentification (opérateur téléphonique). L'utilisation d'une puce nous permet de stocker d'une façon sécurisée les données sensibles de l'utilisateur.

Le stockage de la clé secrète peut se réaliser durant la phase de la personnalisation de la carte. Cela consiste à créer un fichier de base servant à stocker des sessions représentées par des triplets (*session_id*, *master_secret*, *cipher_suite*) représentant des sessions partagées entre le client et les serveurs. Nous notons ici que la validité d'une session peut être équivalente au cycle de vie de la carte à puce.

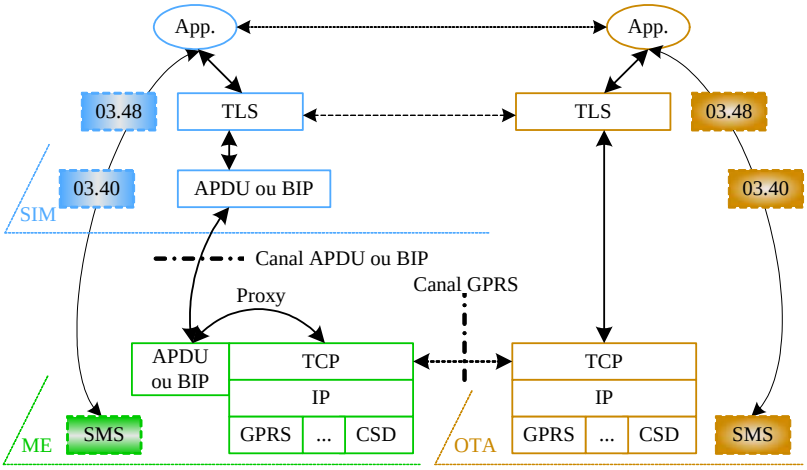

Figure 5.2. L'architecture proposée.

5.2.3.2 L'établissement d'une session sécurisée

Le protocole TLS [TLS] permet à un client et à un serveur de reprendre une session déjà établie. En effet, le protocole TLS permet à un serveur de commencer la négociation en envoyant le message protocolaire *HelloRquest*.

Lorsque le client reçoit ce message, il s'authentifie auprès de la carte (la carte authentifie le client via le code PIN) et déclanche la négociation avec le serveur correspondant. Le terminal du client agit donc comme un simple relais passif.

Si la carte arrive à trouver le triplet partagé avec le serveur à connecter, elle génère le message TLS *ClientHello* contenant l'identificateur correspond à la clé secrète. Le reste de cette négociation est identique à celle définie dans TLS. Nous notons ici que tous les calculs cryptographiques se déroulent sur la carte.

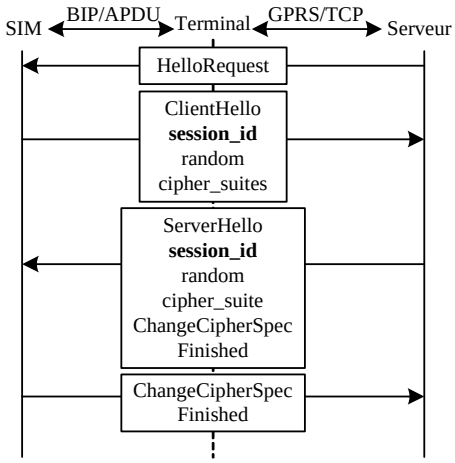

Figure 5.3. L'*Handshake* du protocole TLS-SIM ou TLS-PSK.

5.2.3.3 La fiabilité des échanges de bout en bout

Par sa conception, TLS nécessite une fiabilité de transport de bout en bout. La fiabilité de transport entre le terminal et le serveur est assurée par le protocole TCP. Entre le terminal et la carte à puce, la fiabilité est assurée par la norme ISO[72] 7816-4 [APDU] (qui standardise le format et les protocoles de transmission avec la carte à puce) ou par le protocole propriétaire BIP. Ce dernier est un protocole propriétaire à Gemplus qui permet à la carte d'interagir avec le terminal supportant des mécanismes exigés par les applications de la carte. Le BIP[73] permet au terminal d'offrir à la carte les moyens d'accès aux supports de transport supportés (exemple, TCP).

5.2.3.4 Services et performances

Notre solution assure un chiffrement plus robuste et plus solide que la norme GSM 03.48 grâce aux différents choix de chiffrement fournis par le protocole TLS. En plus, elle comble le manque de l'authentification mutuelle grâce à l'utilisation de la clé secrète dans les deux sens.

Un faible inconvénient de cette solution est que le client et le serveur utilisent la même clé secrète pour toutes les sessions. Cela peut laisser un indice ou une trace sur l'identité du client. Il est donc toujours possible, pour le client SIM et l'opérateur téléphonique, d'achever le rafraîchissement de ses triplets à travers une session TLS.

Nous nous apercevons que notre proposition épargne toute opération asymétrique exigée par la négociation complète de TLS. Elle réduit énormément la charge cryptographique et la charge protocolaire. En effet, nous établissons des benchmarks sur une carte à puce et nous trouvons qu'une session sécurisée de bout en bout (utilisant PSK) [Bad9] ne prend pas plus que 200 ms alors qu'une négociation complète nécessite approximativement 1266 ms (cette valeur augmente avec la longueur de la chaîne de certification). En plus, la charge protocolaire de notre approche ne dépasse pas 300 octets.

[72] International Standard Organisation
[73] Bearer Independent Protocol

5.2.3.5 Relation entre PSK, la carte à puce EAP et le réseau WiFi

La carte à puce EAP est dédiée aux réseaux sans fil comme les réseaux WiFi. Cette carte embarque un moteur EAP sécurisé traitant tous les messages EAP. Elle supporte plusieurs méthodes d'authentification afin de répondre aux besoins de sécurité pour plusieurs environnements. Cette carte définit une interface pour le WISP et une autre pour l'utilisateur assurant un lien avec le système d'exploitation du terminal client. Ces interfaces sont plus détaillées dans [EAP-SC].

Notre solution TLS-PSK peut être réutilisée dans la sécurité du réseau WiFi en introduisant la carte à puce EAP. Dans ce cas, il suffit de remplacer l'opérateur téléphonique par le serveur d'authentification et la carte SIM par la carte à puce EAP. Cette solution utilise alors les mécanismes définis par EAP-TLS comme la dérivation de clés. Sachant que la taille maximale d'un message protocolaire du TLS-PSK ne dépasse pas 100 octets, il n'est pas donc nécessaire de définir ou d'appliquer un mécanisme supplémentaire pour la fragmentation des données durant la phase d'authentification.

5.3 Le protocole TLS Express

Dans le chapitre IV, nous avons présenté une solution qui permet l'utilisation d'une clé pré partagée afin d'achever l'authentification mutuelle.

La norme [TLS-Extension] a définit un mécanisme générique pour permettre la définition de nouvelles extensions utilisées durant la phase "*Handshake*" de TLS. Par exemple, l'extension *max_fragment_length* permet de négocier une taille de segment plus petite que 2^14 octets (taille maximale d'un paquet TLS) en raison des limitations de la bande passante.

Le principe pour définir une nouvelle extension TLS est la suivante :

```
struct {
        ExtensionType extension_type;
        opaque extension_data<0.. 2^16-1>;
       } Extension;
```

Où "*extension_type*" identifie un type d'extension particulier et "*extension_data*" contient des informations spécifiques au type de l'extension.

Grâce à ce mécanisme générique, nous pouvons traduire notre solution comme une extension au TLS de la manière suivante :

```
struct {
        ExtensionType ticket_identity;
        opaque ticket_to_enter<0.. 2^16-1>;
       } Extension;
```

Le champ *ticket_identity* est l'identificateur de la clé partagée et le champ *ticket_to_enter* contient des informations liées à cette clé.

Il reste à ajouter notre extension sur la liste d'extensions définies par la norme *TLS Extensions* [TLS-Extension]. Notre extension réserve le numéro 8 dans la liste d'extensions. La nouvelle liste d'extensions devient alors [Bad2]:

```
enum {
      server_name(0), max_fragment_length(1),
```

```
client_certificate_url(2), trusted_ca_keys(3),
truncated_hmac(4), status_request(5), srp(6), cert_type(7),
ticket_identity(8),(65535)
} ExtensionType;
```

Lorsque le client souhaite ouvrir une session TLS avec une authentification basée sur la clé pré partagée, il suffit d'ajouter notre extension au message *ClientHello* qui transporte l'identificateur du ticket. Le reste de la phase "*Handshake*" est le même que celui de TLS abrégé.

5.4 Le protocole EAP-Double-TLS

La majorité des méthodes d'authentification EAP ne fournissent pas tous les services de sécurité ; notamment l'échange anonyme de données, la protection de l'identité du client, l'utilisation optionnelle des certificats et des clés dynamiques. En plus, la plupart des méthodes sont construites pour répondre à l'anonymat sont victimes de l'attaque *Man-In-The-Middle* [MEAP]. Dans cette partie, nous présentons notre protocole EAP-Double-TLS [Bad3] et nous montrons ses services.

Le protocole EAP-Double-TLS se compose, comme les protocoles PEAP [PEAP] et EAP-TTLS [EAP-TTLS], de deux phases établies entre la station mobile (*Supplicant*) et le serveur d'authentification. La première phase consiste à ouvrir un tunnel sécurisé entre les deux entités en utilisant par exemple le protocole TLS-PSK ou TLS Express [Bad2]. Cette phase permet aux entités de s'authentifier via une clé pré partagée et de calculer les clés de chiffrement de la session afin de permettre aux entités de protéger leurs échanges durant la deuxième phase en utilisant la couche *Record* de TLS-PSK. La phase 2 permet ainsi de rafraîchir les clés partagées pour les sessions futures.

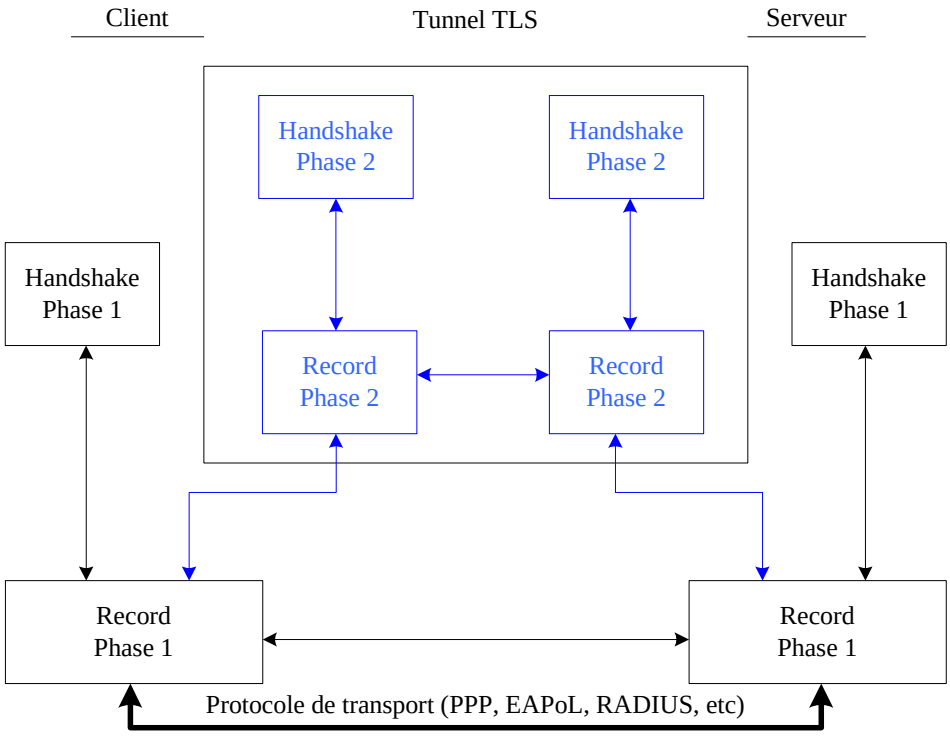

Figure 5.4. La phase *Handshake* du protocole EAP-Double-TLS.

5.4.1 La protection de l'identité de l'utilisateur EAP avec Double-TLS

Au début de la session EAP, l'identité EAP (EAP ID) est envoyée en clair entre la station et le serveur d'authentification. Ce paramètre est utilisé afin de permettre au point d'accès d'expédier les messages EAP vers le serveur destinataire. Le serveur utilise cette identité dans le mécanisme d'authentification et de la gestion de la base de données de l'utilisateur EAP.

Dans notre protocole, nous proposons d'utiliser le *session_id* (dans la terminologie de TLS) comme une valeur de l'identité du client (user ID) transporté par le paquet *EAP-Response*.Identity. Dans le cas où l'utilisateur et le serveur ne sont pas dans le même domaine (*roaming*), l'identité du client s'écrit comme suit : *session_id@serveur.com*.

Cette méthode protége l'intimité de l'utilisateur contre la surveillance et réussit à rendre impossible de tracer les crédits de l'utilisateur par des personnes écoutant les échanges. En effet, le *session_id* courant sera remplacé par un autre généré aléatoirement durant la phase 2 du EAP-Double-TLS.

5.4.2 Vue générale de la négociation EAP-Double-TLS

Afin d'appliquer l'utilisation du TLS-PSK, nous proposons de partager une session entre le client et le serveur. La session est simplifiée par un triplet présenté comme suit :

{session_id, master_secret, cipher_suite}

Le paramètre *session_id* est utilisé pour identifier le triplet. Il correspond aux valeurs de la clé secrète partagée et à l'option cryptographique supportée par le client et le serveur. Cette option est initialisée à un *cipher_suite* particulier.

EAP-Double-TLS consiste en deux phases. Durant la phase 1, la négociation TLS-PSK est utilisée afin d'établir l'authentification mutuelle et la génération de clés de la session. Cette phase utilise une option cryptographique permettant la protection du *record*s de TLS. La négociation de la phase 2 de EAP-Double-TLS est la même que celle du *Handshake* complet de TLS. Durant cette phase, les *record*s TLS sont échangés dans le tunnel établi durant la phase 1. Bien que la négociation TLS exige l'échange de certificats, notre protocole laisse cette échange comme une option. En effet, l'authentification par une clé partagée est largement suffisante entre les deux entités. Durant cette deuxième phase, un nouveau triplet (*session_id*, *master_secret*, *cipher_suite*) est calculé et négocié. Ce triplet remplace celui utilisé durant la phase 2 et il sera utilisé dans la prochaine session EAP-Double-TLS.

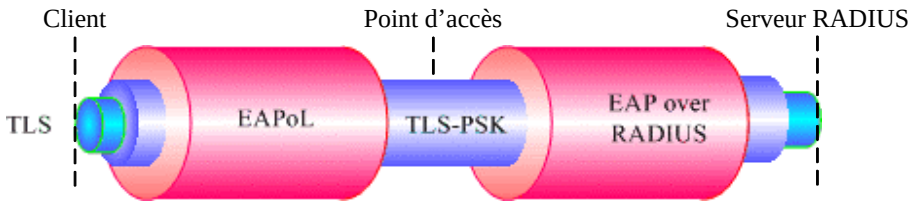

Figure 5.5. Principe de EAP-Double-TLS.

5.4.2.1 Phase 1 : la négociation EAP-TLS-PSK

Cette phase commence avec le certificateur, en envoyant une requête d'authentification à la carte à puce via la station de l'utilisateur. La carte répond avec l'identifiant de l'utilisateur transportant

le *session_id* qui correspond au triplet partagé. Ce message est relayé par le point d'accès vers le serveur d'authentification. Le reste de cette phase est le même que celui défini par TLS abrégé [TLS]. A la fin, le serveur doit envoyer le message *EAP-Double-TLS/HelloRequest* afin d'indiquer au client le succès de la première phase et le début de la deuxième. Dans le cas négatif, une *Alert*e expliquant la cause d'échec doit être envoyée au client.

5.4.2.2 Phase 2 : la négociation EAP-TLS

Dans cette phase, la couche *Record* du TLS-PSK est utilisée afin d'échanger d'une manière protégée les informations échangées entre les deux entités. Ces informations sont encapsulées dans des séquences des attributs TLS qui sont définis dans [TLS].

Comme il est déjà mentionné, le serveur commence cette phase en envoyant au client le message *EAP-Double-TLS/HelloRequest.* Ce message est défini par TLS et peut être envoyer par le serveur à n'importe quel moment. Le but de ce message est d'inviter le client à commencer la négociation d'une session TLS.

Le client et le serveur continuent à échanger les paquets EAP jusqu'à la fin de la phase *Handshake*, comme il est défini dans TLS. Si cette phase est achevée avec succès, le client doit envoyer une réponse EAP de type EAP-Double-TLS avec "*no Data*" et le serveur doit répondre avec le message *EAP-Success*.

A ce stade, le serveur délivre au point d'accès, la clé de chiffrement dérivée du *master_secret* de la première phase. Ensuite, le client et le serveur remplacent l'ancien triplet utilisé durant la phase 1 par celui calculé et négocié durant la phase 2. Le nouveau triplet sera utilisé dans la prochaine négociation EAP-Double-TLS.

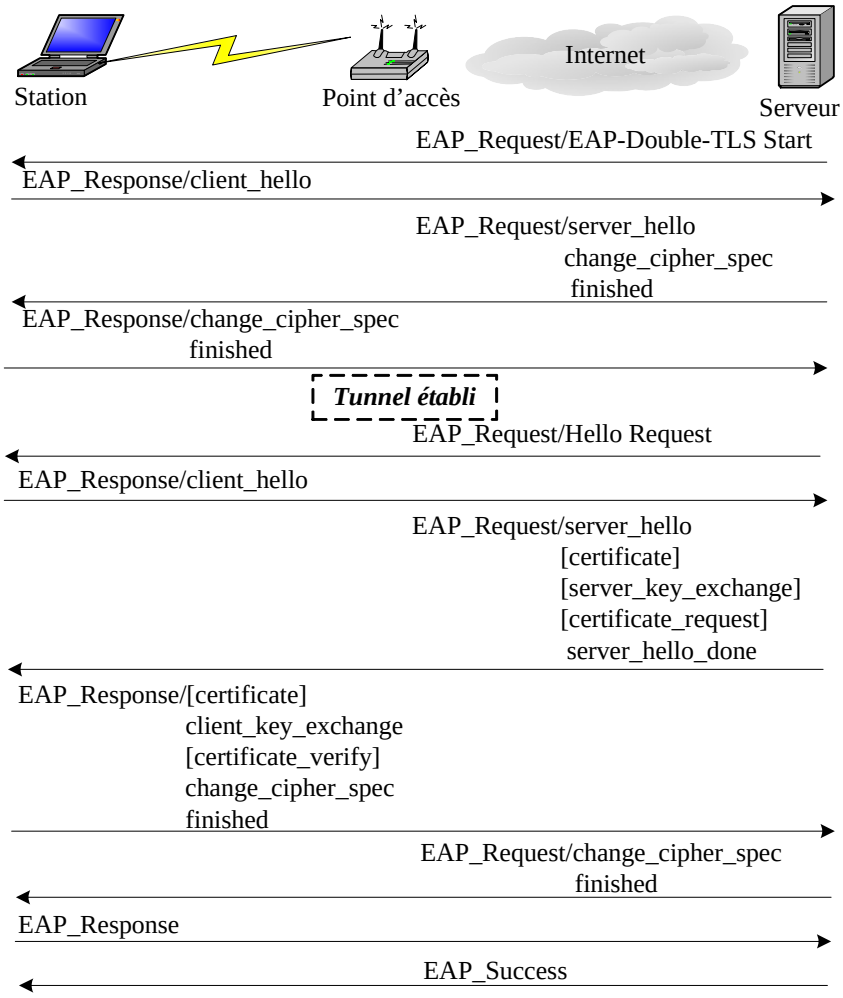

Figure 5.6. La phase d'authentification de EAP-Double-TLS.

5.4.3 Les caractéristiques de Double-TLS

5.4.3.1 La protection d'identité

Il est évident que la phase 1 de notre protocole assure l'authentification mutuelle et la génération de clés. Comme résultat, un canal chiffré est créé entre les deux entités. Cette phase n'exige pas l'échange de certificats.

Durant la phase 2, le serveur peut, selon la politique de sécurité et d'autorisation, envoyer son certificat et demander au client d'envoyer le sien. Etant donné que le certificat est envoyé dans des *record*s TLS chiffrés et protégés par le canal créé durant la phase 1, un homme au milieu ou un indiscret ne peut pas lire et récupérer l'identité du client. En plus la protection contre plusieurs attaques (notamment le *Spoofing* des messages *EAP-Success* et *EAP-Echec*) est assurée.

5.4.3.2 Autres services de sécurité

Notre protocole répond aux différents besoins de sécurité exigés par le standard EAP [RFC2284bis]. Ils sont résumés dans le tableau suivant :

Utilisation prévue	802.11 LAN sans fil, réseaux IP

Mécanisme d'authentification	Clé pré partagée, optionnellement renforcée par des certificats échangés d'une manière protégée
Négociation de *cipher_suite*	Oui, fournit par TLS
Authentification mutuelle	Oui, par la couche *Record* de TLS
Intégrité, Confidentialité, Protection contre re-jeu	Oui, par la couche *Record* de TLS
Dérivation de clés	Oui, réutilise du mécanisme défini dans RFC 3748
Taille de la clé	Comme dans TLS et dans RFC 3748
Protection contre l'attaque par dictionnaire	Oui, pas d'utilisation des mots de passe
Reconnexion rapide	Oui, TLS abrégé
Cryptographic binding	Oui
Session independence	Oui
Fragmentation	Oui, de la même manière de la norme EAP-TLS
Protection d'identité	Oui, en rafraîchissant l'identité de la clé par session
Protection à long terme de données	Oui, en utilisant des clés dynamiques

Tableau 5.1. Les services de sécurité recommandés par IETF-EAP WG.

5.4.3.2.1 *Cryptographic binding*

Ce service fournit la protection contre l'attaque MitM retrouvée dans EAP-TTLS et PEAP. Cela est assuré grâce à l'authentification mutuelle de la première phase de Double-TLS et à la computation du MAC sur l'ensemble de message de la deuxième phase.

5.4.3.2.2 *Session independence*

Cette propriété assure que les attaques passives et actives ne peuvent pas compromettre les clés dérivées de la *master_secret*.

5.4.3.3 Comparaison avec d'autres méthodes EAP

Notre protocole n'exige pas l'utilisation de PKIs et de certificats. De ce fait, la charge cryptographique et protocolaire est bien réduite. En effet, la taille d'un certificat X.509 peut avoir une taille de 1.5 kilo octets. De plus, si l'authentification du client s'établit via un certificat, alors le serveur doit envoyer sa liste de CA de confiance. Dans les implémentations actuelles de TLS, la taille de cette liste peut dépasser 3 kilo octets. Si nous fixons la taille des paquets Ethernet à 1.4 kilo octets, il nous faut 4 Round Trip pour échanger seulement le certificat du serveur et la liste de CA. D'un autre côté, l'utilisation des certificats exige l'utilisation des opérations cryptographiques à clé publique (signature, vérification, etc.). Le nombre de ces opérations et leur temps d'exécution a évidemment une influence non négligeable sur les performances des entités et du réseau. Le tableau suivant résume une comparaison entre notre protocole et les autres méthodes d'authentification.

	EAP-TLS	EAP-TTLS	PEAP	EAP-Double-TLS
Spécification	RFC 2716	Internet-Draft	Internet-Draft	Internet-Draft
Méthode d'authentification	Certificat X.509	Méthodes EAP,MS-CHAP, EAP-TLS		PSK, express, méthodes EAP
Structure basique	Etablissement d'une session TLS et validation de certificats des entités	Deux phases : - établissement d'une session TLS - échange de paires (attributs, valeurs) entre les entités	Deux parties : - établissement d'une session TLS - effectué une méthode EAP à l'intérieur du tunnel TLS	Deux phases : - établissement d'une session PSK - la 2ème phase est optionnellement établie afin de rafraîchir le triplet

Session résumée		Oui	Oui		Oui
Intégration du WEP		Oui	Oui		Oui
Certificat serveur		Requis	Requis		Optionnel
Certificat client		Requis	Optionnel		Optionnel
Vérification de la chaîne de certification		Oui	Oui		Optionnel
Conséquence de la compromission de la clé privée		Réémission de tous les certificats	Réémission le certificat du serveur et celle du client s'il est utilisé dans la 1ère phase		Réémission un simple nouveau triplet
Protection de l'identité du client		Non	Oui		Oui
Vulnérable à l'attaque *Man-In-The-Middle*		Non	Oui		Non
Nombre de RT nécessaires		7.5	- 1ère phase : 7 (5 sans le certificat du client) - 2ème phase (ex. CHAP) : 2 - total : 9 (7 sans le certificat du client)	- 1ère phase : 7 (5 sans le certificat du client) - 2ème phase (ex. MD5) : 3 - total : 10 (8 sans le certificat du client)	1ère et 2ème phases: - 5.5 avec l'exécution du TLS sans certificats en 2ème phase - 3.5 avec CHAP - 4.5 avec MD5
Taille de données à échanger et à transférer		6.5 Koctets	5.1 Koctets + la taille des données de la méthode d'authentification utilisée à l'intérieur du tunnel		1.3 Koctets
Opérations cryptographique asymétriques	client	1 signature, $(1 + x^a)$ vérifications et 1 chiffrement	$(1 + x^c)$ vérifications et 1 chiffrement		1 chiffrement
	serveur	1 déchiffrement et $(1 + y^b)$ vérification	1 déchiffrement		1 déchiffrement

a, b, c : les tailles des chaînes de certification

Tableau 5.2. Comparaison entre EAP-TLS, PEAP, EAP-TTLS et EAP-Double-TLS.

5.4.4 Implémentation

Par la suite, nous présentons notre plate-forme expérimentale du TLS express et du Double-TLS, puis nous en détaillons la réalisation qui comprend le prototypage du client et du module serveur.

5.4.4.1 Implémentation du TLS express

Dans ce prototype, nous avons choisi d'utiliser l'API d'OpenSSL. Cette dernière est une bibliothèque offrant une API pour la gestion de protocoles de communication sécurisés. Elle est écrite en ANSI C et implémente les protocoles TLS 1.0 et SSL 3.0. La bibliothèque est disponible sous une licence libre particulière tant pour un usage commercial que non commercial sous certaines conditions.

Bien que OpenSSL offre une bibliothèque pour la gestion du protocole TLS, il n'implémente pas le support pour les extensions du TLS et pour l'authentification SRP[74] avec TLS [SRPTLS].

[74] Secure Remote Password

La bibliothèque de l'OpenSSL comporte trois parties : une partie concernant le protocole TLS, une partie dédiée à la gestion des certificats et une partie cryptographie.

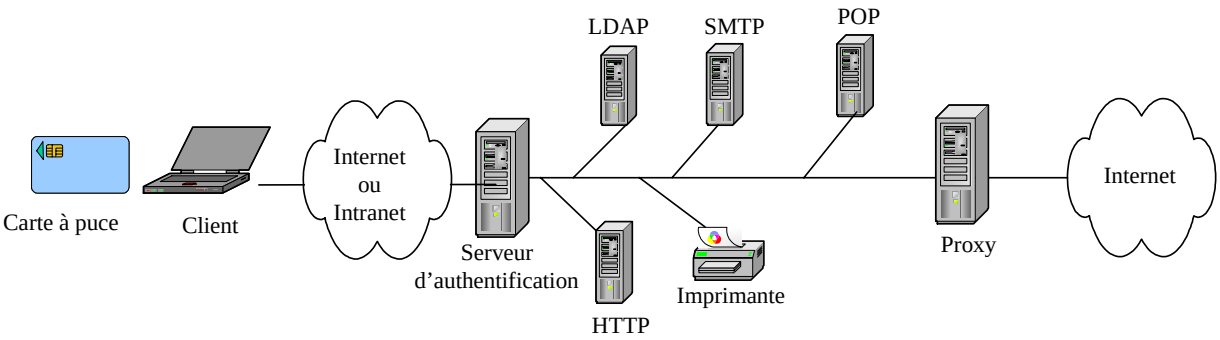

Figure 5.7. L'architecture du test.

Le serveur d'authentification implémente notre serveur OpenSSL, contrôle l'accès et authentifie les clients. Une fois l'authentification établie avec succès, le client peut accéder, en toute transparence, à l'ensemble des ressources autorisées, sur la base d'une authentification unique effectuée lors de l'accès initial au réseau.

TLS Express peut être utilisé pour les clients des réseaux mobiles et pour les clients pré configurés. Il peut jouer un rôle efficace pour le service SSO[75].

Du côté client, le terminal implémente notre client OpenSSL. Le contrôle d'accès et le stockage de la clé partagée sont renforcés par l'utilisation d'une carte à puce. Un composant logiciel doit être installé sur la machine de l'utilisateur pour servir d'interface entre la carte à puce et le réseau sans-fil.

L'état initial de ce prototype est l'état "*global state*". Cet état contient plusieurs fonctions utilisées durant l'établissement de la session ; notamment la fonction d'allocation de mémoire et la fonction d'analyse de XDR.

La structure "*credentials*" est utilisée par plusieurs méthodes d'authentifications (e.g. certificats). Cette structure peut contenir des certificats, des clés privées et des clés pré partagées.

"Session TLS " contient les fonctions et les procédures nécessaires afin d'ouvrir une session sécurisée. La session utilise les fonctions du "Couche Transport" afin de communiquer avec d'autres entités (utilisateurs).

Puisqu'une session TLS peut être résumée, les serveurs ont besoin d'une base de données "*database backend*" afin de stocker et de maintenir les paramètres de sécurité des sessions déjà établies.

[75] Single Sign On

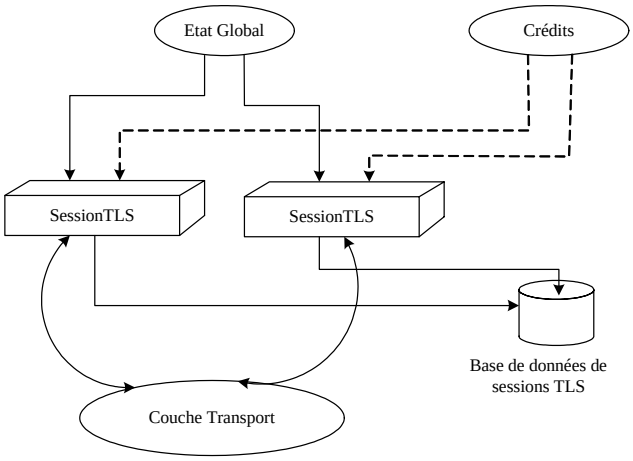

Figure 5.8. Les états protocolaires.

5.4.4.2 Implémentation du EAP-Double-TLS

L'environnement de test du EAP-Double-TLS contient plusieurs éléments :

- un système pour le serveur RADIUS : Debian, noyau 2.4.18
- un système pour le client (supplicant) : Windows XP Pro Service Pack 1
- une borne WiFi (AP)
- une carte WiFi

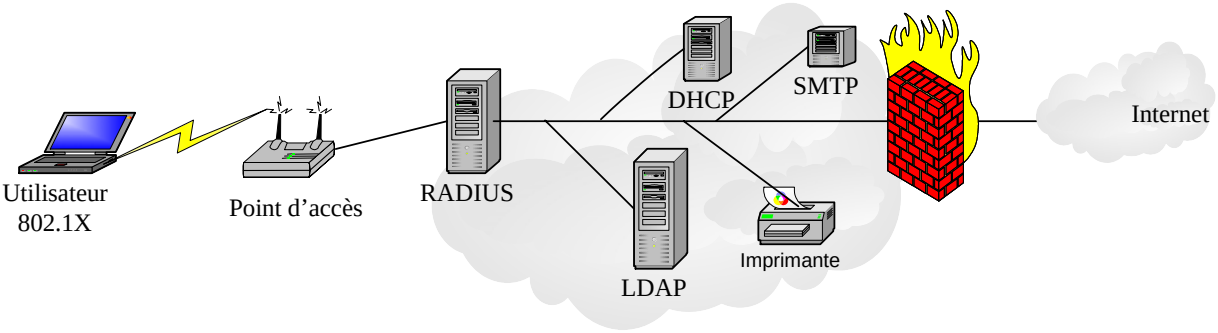

Figure 5.9. Architecture de test de EAD-Double-TLS.

Les programmes ont été installés et configurés dans notre plate-forme de test. Du côté serveur, nous avons effectué les tâches suivantes :

- installation et configuration d'OpenSSL.
- création des certificats CA, serveur et client. Nous citons ici que l'usage de certificats reste une option avec notre protocole.
- installation et configuration du serveur FreeRADIUS.
- configuration du point d'accès.
- développement du EAP-Double-TLS côté serveur.
- développement du EAP-Double-TLS côté client (terminal et carte à puce).

Les deux dernières tâches ne sont pas totalement réalisées (implantées) puisque nous n'avons pas le temps de traiter tous les points concernant cette implémentation. Cela ne nous empêche pas de les implémenter dans le futur prochain.

Avec OpenSSL, nous pouvons établir une session TLS standard et la reprendre ultérieurement. L'actuel API d'OpenSSL permet de reprendre une session existante en cache (du serveur et du client) et il définit l'interface pour enregistrer la session sur le disque. Cependant, OpenSSL n'a pas spécifié comment une session TLS peut être stockée puis re-utilisée dans un délai défini.

L'hypothèse de la première phase du protocole Double-TLS consiste à archiver une session TLS du côté client et du côté serveur. C'est comme le principe du login et du mot de passe. En effet, chaque session TLS est identifiée par un identificateur sur 32 octets (*session_id*). Un client connecté à un serveur donné doit envoyer l'identificateur de la session. Ensuite, le client et le serveur prouvent qu'ils ont la clé secrète de la session via les messages *Finished* du TLS. Notons ici que le client et le serveur stockent la session dans un fichier DER ou PEM et alors le nom du fichier est soit *session_id.pem* soit *session_id.der*.

Durant cette phase d'implémentation, les principales fonctions utilisées dans la gestion de sessions sont :

- *SSL_CTX _session :* créer le contexte de la session.
- *PEM_write_SSL_SESSION :* archiver les paramètres de la session (dans un fichier) sur le disque.
- *PEM_read_SSL_SESSION :* lire les paramètres de sécurité de la session stockée (en format pem).
- *SSL_CTX_add_session :* construire le contexte de la session à reprendre.
- *SSL_set_session :* utiliser le contexte de la session partagée afin d'ouvrir une session TLS abrégée.

La deuxième phase de l'implémentation consiste à ouvrir une nouvelle session à l'intérieur du canal sécurisé établis durant la première étape. La nouvelle session aura un nouvel identificateur (*new_session_id*) et des nouveaux paramètres de sécurité.

Comme Double-TLS n'utilise qu'une seule fois la session partagée, cette dernière sera remplacée par celle établie à l'intérieur du canal. Comme résultat, le fichier *session_id.pem* contiendra le contexte de la nouvelle session et son identificateur deviendra *new_session_id.pem*. Les différents états sont visualisés par la figure suivante :

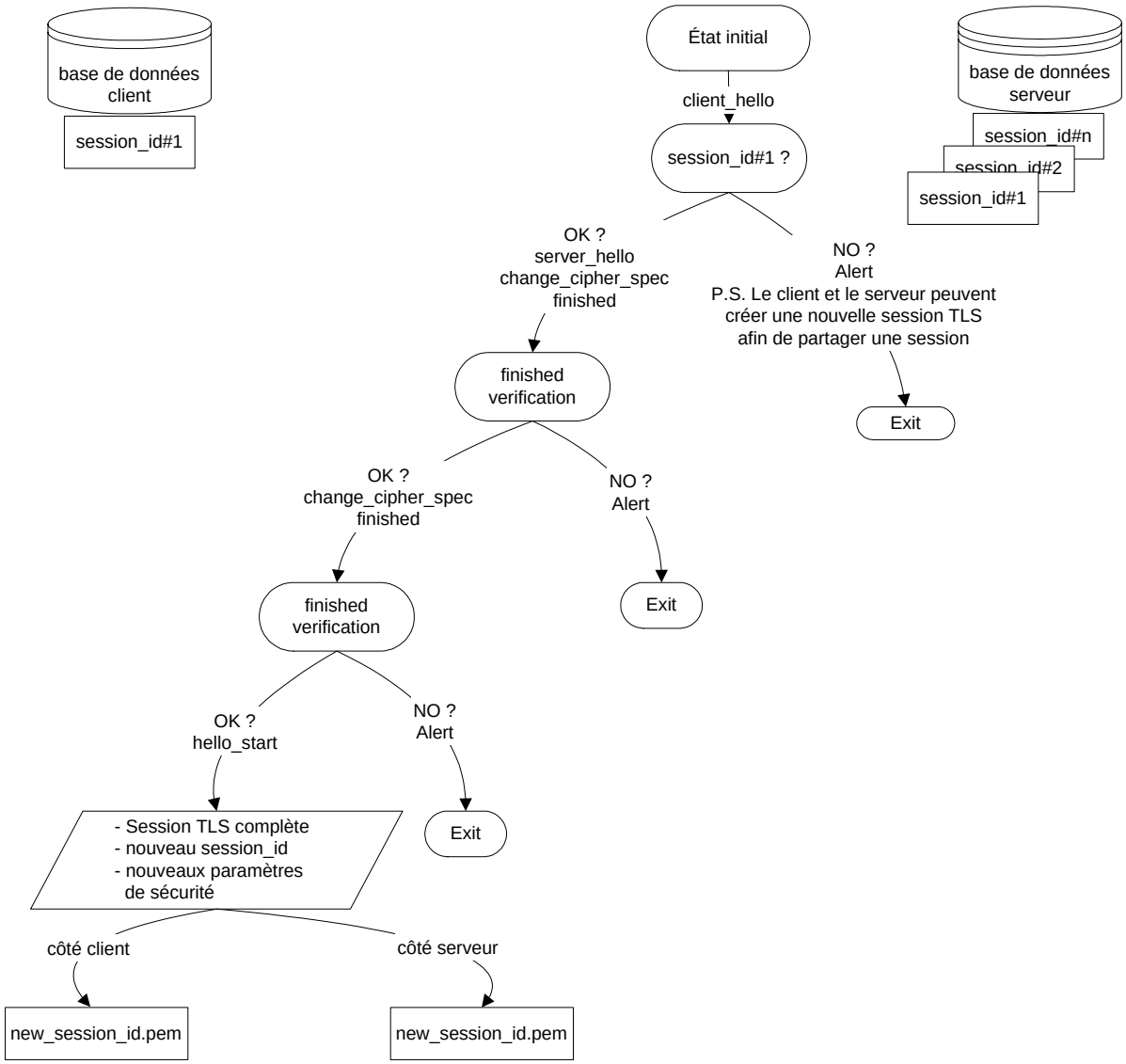

Figure 5.10. La logique d'une session Double-TLS.

Avec la présentation PEM, les paramètres d'une session TLS peuvent avoir la forme suivante :

```
-----BEGIN TLS SESSION PARAMETERS-----
MIIC8wIBAQICAwEEAgA1BCBHWYbS43ELhjrCDSqUUBeaXUGgFv67S8XJkvCt3NWX
+AQwLHrjimj+3FQ2fCvIZ9kMaiNFFQUKCJp5U3n7cEYWhmJ+dIjCg+TpqCsScDn3
8hP/oQYCBEEJFpyiBAICASyjggJ6MIICdjCCAd+gAwIBAgIBATANBgkqhkiG9w0B
AQQFADCBgDELMAkGA1UEBhMCRlIxCzAJBgNVBAgTAjc1MQ4wDAYDVQQHEwVQYXJp
czENMAsGA1UEChMERU5TVDEPMA0GA1UECxMGSU5GUkVTMRYwFAYDVQQDEw1CYWRy
YSBDQSBUZXN0MRwwGgYJKoZIhvcNAQkBFg1iYWRyYUBlbnN0LmZyMB4XDTA0MDcy
NzEzMjg0MVoXDTA1MDcyNzEzMjg0MVowaDELMAkGA1UEBhMCRlIxCzAJBgNVBAgT
Ajc1MQ0wCwYDVQQKEwRFTlNUMQ8wDQYDVQQLEwZJTkZSRVMxDjAMBgNVBAMTBWJh
ZHJhMRwwGgYJKoZIhvcNAQkBFg1iYWRyYUBlbnN0LmZyMIGfMA0GCSqGSIb3DQEB
AQUAA4GNADCBiQKBgQDOabAdRACFduBFQXi3T+UATTDPCeyaIbLqQXwkyXI94bcm
ctPzrWaoPLd1MlSDDfD1zA1TTcbUj+H/QnSRgc20J1lHImyxeJCIeaX1hyqnWf5d
xwaOyijcxvBpRGjjhuWfv4I1RzisQLBNfENuLc97IrU2CwsO4LMk1pFBwbdyUwID
AQABoxcwFTATBgNVHSUEDDAKBggrBgEFBQcDAjANBgkqhkiG9w0BAQQFAAOBgQAX
Ro0KX3nE5aLoB0JTjdWZrB76wdzL3HHg6KbHnwOxKbPuzrZ9KdnOBcw+1F/lsxxE
```

```
qf3/5JLAY8FCLdr1AY7QNHFqx07i4tgvkYj3JkydUwejXy4cLCmBdlkJ5dpL9a+H
bxQSyWWBKIKnrMZXj7ojfgzWN/dT+I3k3XskVkgc8qQGBAQBAAAA
            -----END TLS SESSION PARAMETERS-----
```

5.5 La carte à puce EAP-TLS

Les protocoles TLS-PSK et TLS express permettent donc l'authentification basée sur une clé secrète pré partagée. Cette clé peut s'installer sur la carte durant la phase de la personnalisation mais aussi durant une session TLS. En effet, la carte et le serveur d'authentification peuvent établir une session TLS standard et stocker ensuite la clé de la session TLS pour les futures sessions PSK-TLS ou TLS express. Dans ce but, nous étudions et analysons les contraintes et les problèmes de la première implémentation du protocole EAP-TLS sur la carte à puce ([Bad6], [Bad7]).

5.5.1 L'architecture

Dans l'architecture 802.1X, l'authentification est établie entre le terminal et le serveur d'authentification. Avec l'utilisation de la carte EAP, le terminal doit agir comme un "proxy" d'authentification vers la carte et ne doit transmettre que les paquets EAP. Dans ce cas, les paquets EAP seront traités au niveau de la carte alors qu'un composant logiciel installé sur le terminal devra être capable de récupérer les paquets EAP au niveau de l'interface réseau et de communiquer avec la carte. Les paquets EAP entre la carte et le terminal sont transportés par des APDUs ISO7816.

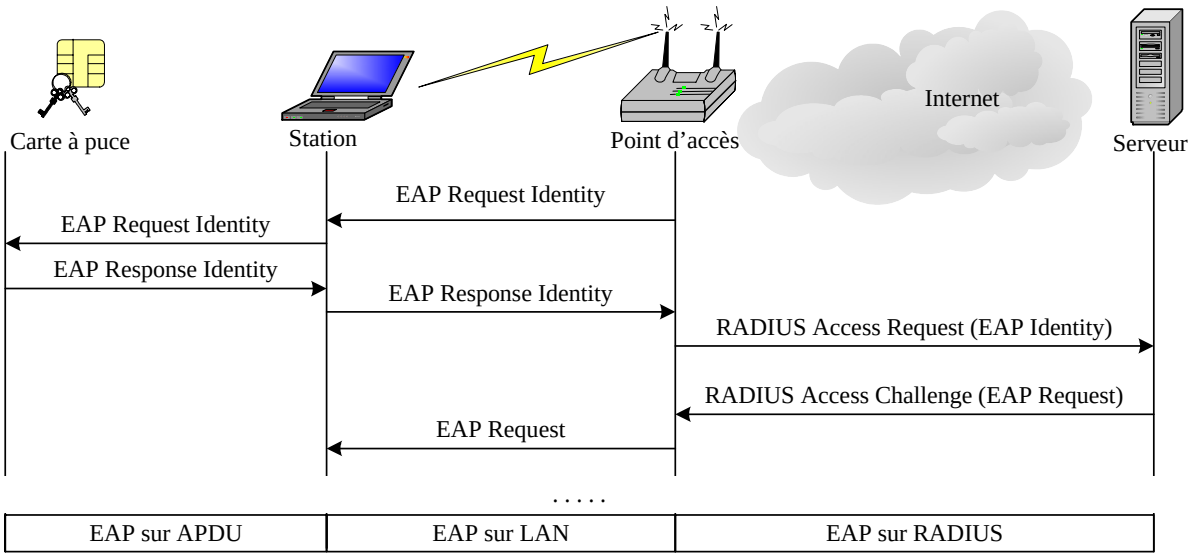

Figure 5.11. L'architecture implémentée.

5.5.2 La fragmentation

La taille des paquets EAP-TLS [EAP-TLS] échangés entra la carte et le terminal peut dépasser la taille maximale d'un APDU (256 octets). Cela nécessite la définition d'un mécanisme de fragmentation en assurant la fiabilité d'échanges.

La couche *Record* du protocole TLS fragmente les blocks d'information dans de *record*s TLS dont la taille est de 16384 octets ou moins. De plus, un message TLS peut transporter de

multiples *records* TLS. Puisque la couche MAC de l'IEEE 802.3 ne peut pas envoyer de fragments avec une taille plus grande que 1518 octets et que le protocole EAP ne fournit pas le moyen de la fragmentation de données, la méthode d'authentification EAP doit s'occuper de la fragmentation. La méthode EAP-TLS définit un processus pour la segmentation qui découpe les messages TLS dans de des blocs plus petits qui seront acquittés par le récepteur. Dans notre implémentation, le serveur RADIUS génère la requête d'acquittement et la station répond par la réponse correspondante.

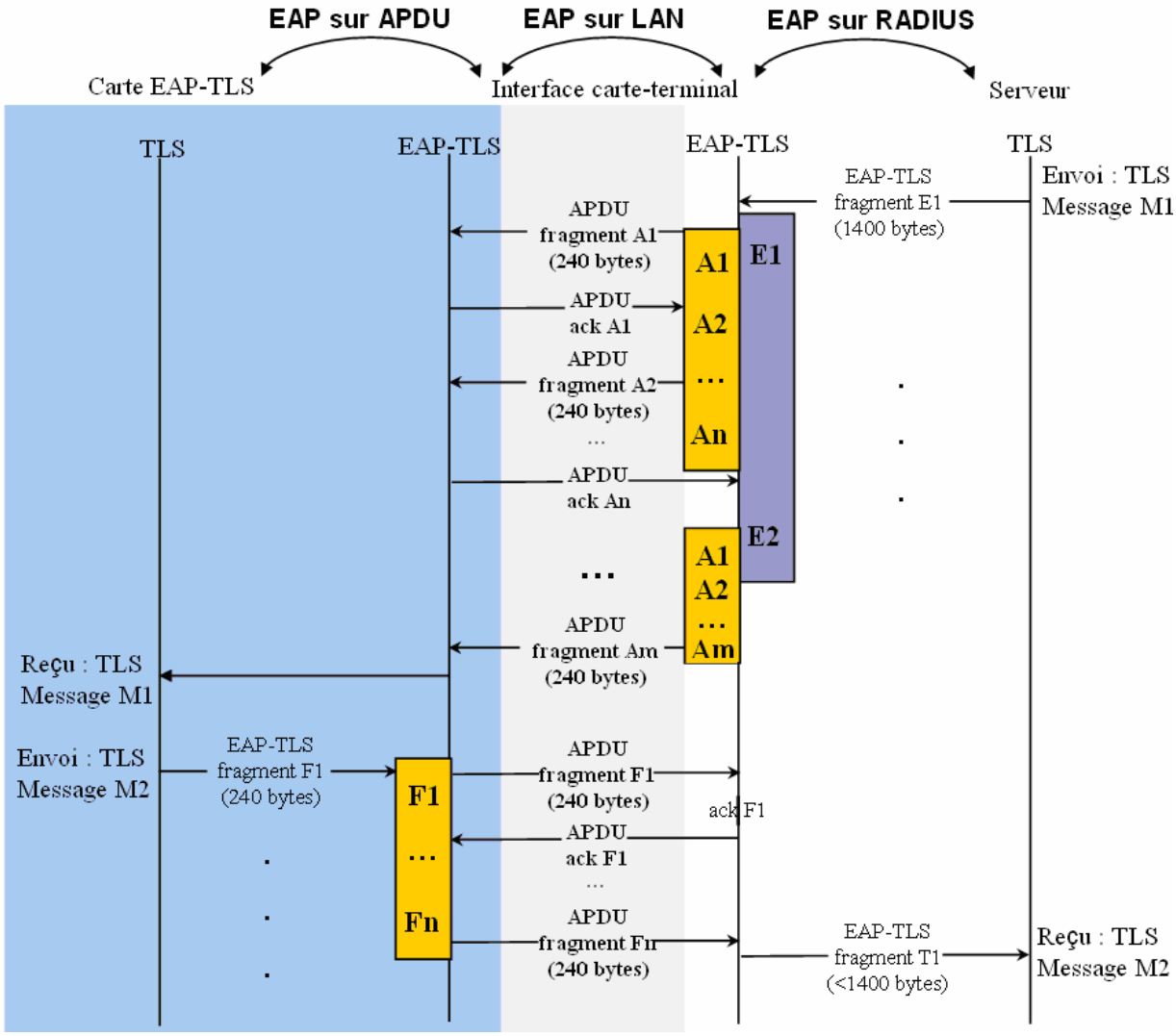

Figure 5.12. Le processus de la double segmentation entre la carte et le serveur.

Il est donc nécessaire d'offrir un moyen de fragmentation entre la station et le driver du lecteur de la carte à puce. Pour cela, une double fragmentation a été introduite par notre implémentation afin d'envoyer les paquets TLS vers la carte. Notre mécanisme est défini comme suit : Au début, les messages TLS du serveur sont découpés en petits fragments (E1, E2) de taille ne dépassant pas 1400 octets (figure 5.12). Ensuite, les segments sont encapsulés dans de paquets EAP-TLS qui à leur tours sont fendus dans une collection d'APDUs (A1 à An) en forme de commandes ISO 7816. Les APDUs (nous supposons que la taille moyenne d'un APDU est de 240 octets) sont alors envoyés vers la carte. Nous notons ici que pour chaque paquet APDU reçu par la carte, une

réponse APDU avec deux octets de données est générée afin d'informer la station de la situation de l'APDU (s'il est bien reçu et correctement traité ou pas).

Du côté de la carte, les messages TLS sont générés puis envoyés par des paquets EAP-TLS avec une taille moyenne de 240 octets. Ces fragments sont directement encapsulés dans des réponses APDU. Entre la station et le serveur, c'est l'interface de la carte (qui a lieu dans la station) qui s'occupe du re-assemblage des données et de la taille du segment.

5.5.3 Les composants de la carte à puce EAP-TLS et l'environnement de développement

Notre carte EAP-TLS est composée de deux entités logicielles écrites en classes *JavaCard*. Deux principales classes sont développées. La première classe implémente les principales fonctions du protocole EAP ; notamment la gestion d'identité et la vérification du code PIN. La deuxième classe supporte la fonctionnalité du TLS et du EAP-TLS. En effet, cette classe réalise toutes les opérations du TLS ; notamment la segmentation et le re-assemblage. Cette classe est capable d'analyser et de produire les messages TLS et d'assurer les fonctionnalités des couches *Handshake* et *Record* de TLS. Plus particulièrement, elle peut achever l'authentification mutuelle, la vérification du certificat, le chiffrement à clé privée/publique, le chiffrement/déchiffrement de données et le calcul de MAC. Nous notons ici que la carte contient, entre autres, le certificat du CA et les clés privées/publiques du client.

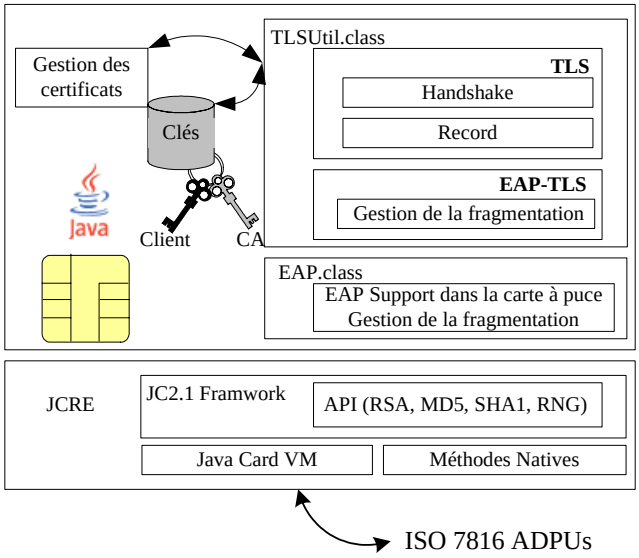

Figure 5.13. Les éléments de la carte à puce EAP-TLS.

Les classes sont donc développées en utilisant la plate-forme *JavaCard*2.1 (JC) qui fournit une API sécurisé et indépendant du vendeur de la carte à puce. Une carte à puce Java embarque une JVM dont les spécifications sont maintenues par Sun. La JVM d'une carte Java ne supporte qu'un sous-ensemble restreint du langage Java et elle contient une API dédiée au développement d'applets pour la carte à puce basée sur les standards ISO 7816. Cependant, elle offre un environnement d'exécution, appelé JCRE, qui supporte des fonctions de gestion d'applets telles que le mécanisme de sélection d'applets.

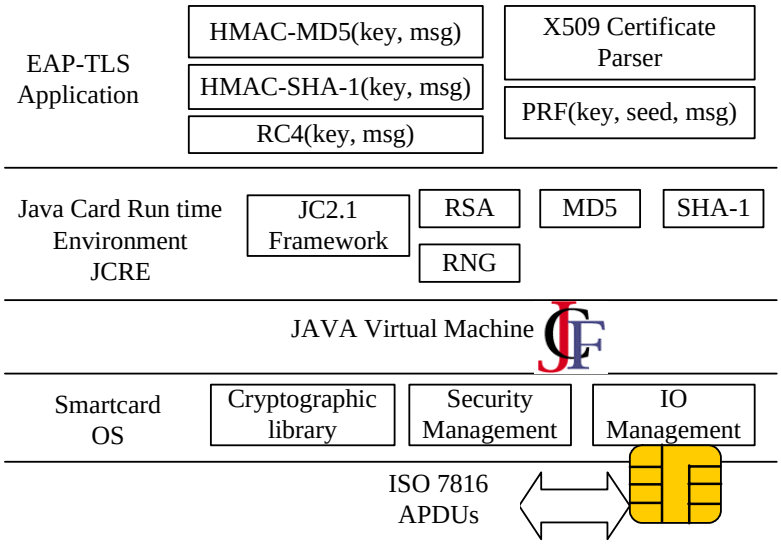

Figure 5.14. Relation entre les composantes de EAP-TLS et la plate-forme *JavaCard*.

D'un autre côté, JC fournit nativement les services cryptographiques essentiels exigés par TLS comme la génération de valeurs aléatoires (RNG), les fonctions de hachage MD5 et SHA1 et le chiffrement/déchiffrement RSA et DES. Cependant, quelques fonctions additionnelles ne sont pas fournies par JC mais sont introduites par TLS lui-même comme la fonction PRF, le chiffrement par RC4 et l'analyseur du certificat X509. Cet analyseur sert à traiter les données de la signature et à extraire la clé publique du certificat.

5.5.3.1 La carte à puce *JavaCard*

La technologie de *JavaCard* offre la possibilité de développer un seul code unique (applet) indifféremment du support physique qui exécute le code. Cependant, plusieurs contraintes et considérations citées dans [Hayder], par rapport au langage Java, doivent être respectées avec *JavaCard*.

Le cycle de vie d'un applet java commence par l'édition du code de l'applet en le produisant en *JavaCard*. Ensuite, le code source est compilé (sous un compilateur Java) afin d'avoir le fichier de classe. Le *JavaCard* définit le moyen de vérifier la conformité du code byte produit dans la phase précédente selon ses contraintes et à convertir le fichier de classe dans un format spécial pour la carte. Le résultat de cette conversion est un ensemble de composants (package) prêts à être téléchargé dans la carte [Book-SC]. Le téléchargement du package consiste à utiliser un ensemble de commandes/réponses APDUs.

5.5.3.2 Le serveur d'authentification

Le serveur d'authentification utilisé par notre implémentation est le serveur Windows 2003 (FreeRadius est un autre exemple pour les systèmes UNIX). Il offre le contrôle de l'accès aux ressources, la vérification de l'identité de l'utilisateur et il supporte la plupart des méthodes d'authentification EAP. Ce serveur décrit les attributs RADIUS pris en charge par le routage, l'authentification et l'accès distant pour les différents paquets RADIUS.

Le serveur Windows 2003 offre les fonctionnalités de LDAP qui est un protocole de communication conçu pour être utilisé sur les réseaux TCP/IP. Il définit la façon dont un client de l'annuaire peut accéder à un serveur d'annuaire et la façon dont il peut exécuter les opérations de

l'annuaire et partager ses données. Il définit les services d'annuaire Active Directory renfermant des informations relatives aux objets d'un réseau et facilite leur recherche et leur utilisation pour les administrateurs et les utilisateurs. L'*Active Directory* permet la gestion de l'authentification et de contrôle d'accès d'une manière efficace.

Le serveur Windows 2003 introduit aussi les moyens pour construire une autorité de certification et une infrastructure à clé publique servant à définir les services pour l'émission et la gestion de certificats utilisés.

5.5.3.3 Le *Supplicant*

La station mobile opère sous Windows XP de Microsoft et intègre un client 802.1X. Elle se connecte à la carte à puce via un lecteur de carte.

5.5.4 Benchmark, performances et analyse

La carte EAP-TLS a été développée en utilisant deux cartes *JavaCard*. Les caractères physiques et cryptographiques de ces deux cartes sont illustrés dans le tableau suivant. Les deux cartes incluent des co-processeurs cryptographiques capables d'achever le calcul de l'algorithme RSA en moins de 500 ms.

Deux tests sont effectués afin d'évaluer les performances (système d'exploitation et protocole d'authentification) de notre implémentation. Ces performances dépendent de plusieurs paramètres ; notamment la capacité du processeur, la taille du mémoire, les opérations cryptographiques invoquées par le protocole d'authentification et le temps de transfert des données entre les entités.

Carte	CPU	RAM octets	ROM Koctets	E2PROM Koctets	Max. Clock	MaxData Rate	RSA Co-processeur	RNG
A	8 bits	2304	96	32	10 MHz	424 kbit/s	1088 bits	oui
B	8 bits	4096	96	34	8 MHz	1000 kbit/s	4032 bits	non

Tableau 5.3. Caractéristiques du microcontrôleurs.

Puisque les deux cartes supportent un co-processeur RSA, nous avons remarqué que le temps d'exécution de RSA varie entre 100 et 500 ms. Durant sa phase *Handshake*, la méthode EAP-TLS effectue les opération RSA trois fois (vérification de la signature du CA et du client, et un chiffrement avec la clé publique du serveur). Dans notre test, ces opérations consomment approximativement une seconde.

EAP-TLS utilise les fonctions de hachage MD5 et SHA1. Dans notre implémentation, ces fonctions sont appliquées sur des blocs de 64 octets. Par conséquent, le temps de calcul des ces fonctions dépend de la taille du bloc utilisé. Les cartes utilisées dans cette implémentation ne possèdent pas de co-processeurs MD5 ou SHA-1. Elles sont donc développées dans les librairies cryptographiques et exécutées par le CPU de la carte. Les résultats de nos tests montrent que les temps d'exécution moyens d'un dual condensâtes (MD5 et SHA1) est respectivement (pour la carte A et B) :

- 155 et 120 ms pour des blocs de taille courte (128 octets),
- 2350 et 1100 ms pour des blocs de taille longue (6 kilo octets).

Durant sa phase *Handshake*, EAP-TLS applique trois fois le dual Hash sur de blocs de taille longue. Le temps d'exécution de ces trois opérations est estimé à 14.1 secondes pour la carte A et à 6.6 secondes pour la carte B.

Puisque l'API *JavaCard* ne donne pas les moyens natifs d'implémenter toutes les opérations cryptographiques du EAP-TLS, les fonctions cryptographiques suivantes sont ajoutées :

- La procédure HMAC : elle peut être utilisée avec MD5 et SHA1. HMAC-MD5 et HMAC-SHA1 sont appliquées sur une clé de taille moins de 64 octets (dans le contexte de TLS) et sur un message de taille moins de 128 octets. Dans ces conditions, HMAC utilise les deux fonctions de hachage et l'opération de l'addition. Nos tests montrent que le temps d'exécution de la procédure HMAC est de 2 secondes pour la carte A et d'une seconde pour la carte B.
- La fonction PRF utilisée pour calculer, entre autres, le master secret et les clés de chiffrement de la session TLS. Cette fonction utilise HMAC-MD5 et HMAC-SHA1 dans son calcul. Afin d'avoir une sortie dont la taille est égale à 80 octets, cette fonction appelle HMAC-MD5 dix fois et HMAC-SHA1 huit fois (nous utilisons ici une clé de 32 octets et un bloc de taille moins de 128 octets). Par conséquent, le temps d'exécution est approximativement 40 secondes pour la carte A et 20 secondes pour la carte B. Comme EAP-TLS utilise la fonction PRF cinq fois durant sa phase *Handshake*, le temps d'exécution est donc 200 secondes pour A et 100 secondes pour B.
- La procédure de RC4 : cette méthode est coûteuse en temps d'exécution puisqu'elle traite un tableau (*array*) de 256 octets stockés sur le E2PROM. Le temps de chiffrement/déchiffrement de 32 octets est 21 pour la carte A et 12.5 secondes pour B. Cette procédure est appelée deux fois durant la phase *Handshake*, ce qui nécessite 42 secondes pour A et 23 pour B.

5.5.4.1 Les performances générales

La figure suivante montre la répartition du temps de calcul durant la phase *Handshake* de EAP-TLS. La carte A achève cette phase en 266 secondes alors que l'exécution par la carte B nécessite 151 secondes. Le temps de transfert des messages EAP-TLS entre la carte et le terminal est 4.5 secondes pour la carte A et 12 secondes pour la carte B.

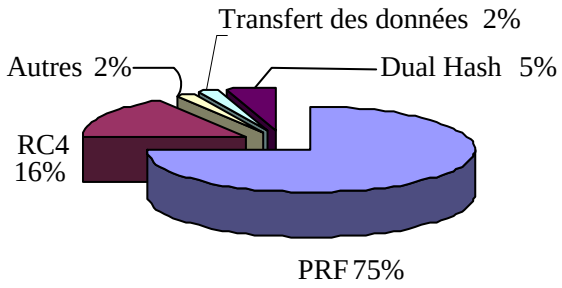

Figure 5.15. Répartition du temps de calcul cryptographiques pour A et B

Il est clair que la plupart du temps d'exécution est pris par les fonctions PRF et du hachage. Ce temps peut énormément se réduire avec l'utilisation du co-processeur cryptographique effectuant les fonctions du hachage. En effet, une opération HMAC-MD5 avec une clé de 64 octets s'exécute en 0,004 ms sur un PC Intel à 1200 MHz et 389 Méga octets de RAM, alors qu'une signature RSA avec une clé de 1024 bits prend approximativement 5,7 ms et 0,3 ms pour la vérification.

5.5.4.1.1 Optimisation software

Nous avons montré que la fonction PRF est le goulot d'étranglement dans les mesures de performances EAP-TLS. Nous avons trouvé aussi que le temps exigé par le traitement de quelques messages EAP-TLS par la carte à puce dépasse la valeur de temporisation (30 secondes) spécifié par [802.1XAD]. Cette valeur de temporisation correspond au temps pour avoir la réponse à un paquet 802.1X envoyé par le serveur. A cet effet et pour éviter de renvoyer plusieurs fois les paquets 802.1X, nous proposons d'améliorer la performance de EAP-TLS afin d'être fonctionnel avec des équipements comme la carte à puce.

Nous avons cité que l'utilisation de co-processeurs MD5 et SHA-1 peut améliorer énormément la performance de la carte à puce EAP-TLS. Cependant, une implémentation attentive peut améliorer aussi cette performance, cela en profitant des deux caractéristiques du langage JavaCard. En effet, ce langage supporte deux types d'objets : persistant et temporaire. Le première est associé aux opérations extrêmement lentes en écriture et elle utilise la mémoire non volatile (E2PROM, Flash, etc.) alors que le deuxième type d'objets est associé aux opérations vites. Ce deuxième type utilise la mémoire volatile (RAM).

Dans notre implémentation améliorée, nous proposons d'utiliser des tableaux de type temporaire avec les classes du TLS. Cela nous amène à réduire le temps d'exécution d'un facteur trois en utilisant la carte à puce B. Avec cette implémentation, le temps de traitement de n'importe quel message EAP-TLS est toujours plus petit que 30 secondes. Le temps de traitement des messages EAP-TLS est fournit dans le tableau suivant :

Opération	Temps de traitement (en seconde)
transfert du message *ServerHello* vers la carte	10.0
traitement du message *ServerHello*	24.0
transfert de la réponse de la carte (*ClientHello*)	2.5
traitement du message *Finished* du serveur	8.5

Tableau 5.4. Le temps maximal du traitement des messages EAP-TLS

Le nouveau benchmark peut se résumer par la figure suivante :

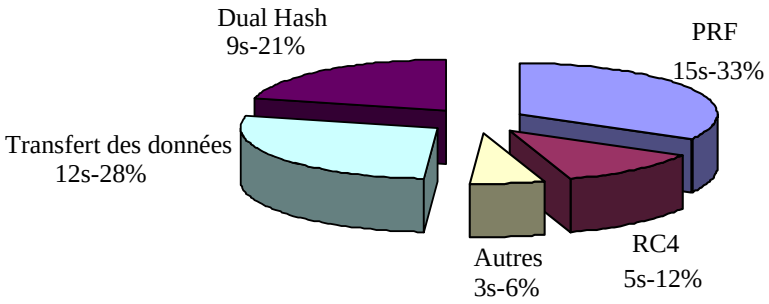

Figure 5.16. L'amélioration du temps de traitement de la carte EAP-TLS

5.6 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de EAP-TLS sur la carte à puce et nous avons étudié et analysé sa performance, ses avantages et inconvénients dans le contexte des réseaux sans fil.

Nous avons spécifié Double-TLS, un protocole de sécurité de bout en bout assurant plusieurs services indispensables aux réseaux mobiles comme l'anonymat. Nous avons proposé également deux solutions d'authentification basées sur la clé partagée avec TLS. L'utilisation de la clé partagée avec TLS est supportée actuellement par l'IETF et le 3GPP. Cette utilisation nous permet d'établir une session sécurisée avec une authentification mutuelle. Malheureusement, l'utilisation de la clé pré partagée n'assure pas tous les services ; notamment la protection contre les attaques actives, la protection à long terme de données et l'anonymat. Dans le chapitre suivant, nous proposons une solution intégrant le chiffrement symétrique et asymétrique afin de découvrir les différents services manquant dans les solutions existantes.

Partie IV : Une méthode d'authentification intégrant les extensions et les architectures des parties précédentes

Chapitre 6

TLS avec PSK-PKC[76]

Résumé du contenu

6.1 Introduction

Dans les chapitres précédents, nous avons présenté, analysé et proposé plusieurs solutions de sécurité basées sur TLS en introduisant d'autres mécanismes d'authentification comme celui de la clé partagée. D'autres contributions proviennent de Cisco [Cisco-Draft], Microsoft [Microsoft], Nokia et Siemens [Nokia], proposent encore l'utilisation des clés partagées. A l'exception de SRP-TLS [SRPTLS] (qui a un copyright et un brevet), les propositions actuelles n'assurent pas tous les services de sécurité ; notamment le PFS, l'anonymat et la protection contre les attaques par dictionnaire et les attaques passives.

Dans le but de répondre à ces exigences, nous présentons un protocole intégrant l'utilisation de la clé partagée et du PKC. La clé partagée est utilisée pour l'authentification mutuelle alors que la clé publique est utilisée pour générer une clé pour chaque session. Notons ici que les résultats de cette approche ont été utilisés pour une contribution à la normalisation au sein de l'IETF. Le futur standard de l'IETF-TLS traite l'intégration de la clé partagée et le chiffrement asymétrique de la même façon que celle définie par notre approche.

6.2 Historique

Depuis les années 90, les contributions sur l'utilisation de clés partagées avec TLS ne cessent pas d'apparaître. L'avantage d'une telle utilisation s'adresse aux environnements fermés où l'utilisation de l'authentification à base de certificats et de PKIs est indésirable à cause du temps du calcul cryptographique, des limitations de ressources et de la gestion mais aussi à cause du coût élevé de la maintenance administrative.

[76] Public Key Cryptography

Presque tous les réseaux mobiles utilisent des procédures à clé partagée pour authentifier leurs clients (exemple, Kc dans GSM) et ont défini leurs mécanismes de gestion et de maintenance de ces clés symétriques. Malheureusement, les protocoles d'authentification utilisés par ces réseaux sont propriétaires, n'assurent pas une authentification mutuelle et ne fournit pas de choix pour un chiffrement plus solide, robuste et négociable. Un protocole de sécurité flexible et extensible devient nécessaire pour l'interconnexion entre les réseaux mobiles ; notamment 3GPP et 802.11 sans fil.

Dans ce chapitre, nous proposons une méthode d'authentification pour TLS combinant le chiffrement symétrique et asymétrique. Elle n'utilise plus les PKIs et garde le même niveau de sécurité fourni par l'usage de certificats. Elle s'intitule "*pre-shared-key key exchange method for TLS*" et son abréviation est KEM-TLS ([Bad1], [Bad5]).

6.3 Le protocole KEM-TLS

Le protocole TLS utilise les certificats comme une méthode pour l'authentification mutuelle. Cette méthode est utilisée afin de négocier les clés de la session TLS en générant la clé *premaster secret*, puis la chiffrant par la clé publique du serveur. Cette clé publique doit être authentifiée. C'est à dire, il faut que le serveur ait une preuve que la clé publique est bien la sienne. Autrement dit, il faut que le serveur ait un certificat signé par une autorité de confiance et que le certificat contienne à la fois le nom du serveur et sa clé publique. L'usage de certificat empêche plusieurs attaques ; notamment l'attaque MitM.

Avec le protocole KEM-TLS, le serveur envoie toujours sa clé publique mais avec l'absence totale du certificat. Par conséquent, le serveur peut envoyer une clé temporaire non authentifiée par un certificat. Dans ce chapitre, nous montrons comment le serveur et le client s'authentifient via la clé pré partagée, comment ils établissent des clés dynamiques pour chaque session et comment ils luttent contre les différents types d'attaques.

6.3.1 Les messages protocolaires de KEM-TLS

Le protocole KEM-TLS est une extension du TLS et est soumis comme un *Internet Draft* à l'IETF [Bad1]. Il utilise la même notion utilisée par TLS pour la représentation et la structuration de messages. Dans un premier temps, nous expliquons la structuration des messages KEM-TLS, nous analysons ensuite ses avantages par rapports aux méthodes d'authentification existantes.

6.3.1.1 La structure du message *ClientHello*

Lorsqu'un client KEM-TLS veut ouvrir une session sécurisée avec un serveur KEM-TLS, il doit indiquer dans son message *ClientHello* qu'il supporte ce protocole. Deux solutions sont possibles pour cette action :

- soit le client envoie une liste de *cipher_suites* spécifiée pour l'utilisation de PSK avec TLS ; par exemple TLS_DHE_PSK_WITH_RC4_128_SHA. Dans ce cas, le client informe implicitement le serveur qu'il supporte l'authentification à clé partagée et qu'il peut échanger les clés en utilisant l'algorithme de Diffie-Hellman.
- soit le client ajoute une extension à son message *ClientHello* en utilisant le standard TLS Extensions [TLS-Extension]. Lorsque cette extension est envoyée, elle peut transporter une ou plusieurs méthodes d'échange de clés en mode PSK supportées par le client. Nous définissons la structuration de cette extension (en XDR) comme suit :

```
struct {
  PSKMethod psk_methods_list<0..2^16-1>;
} PSKKeyExchangeMethod;

struct {
  MethodType method_type;
    Select (method_type) {
     case rsa_psk : RSAPSK
     case diffie_hellman_psk : DHPSK
    } method;
} PSKMethod;

enum { rsa_psk(0), diffie_hellmen_psk(1), (255) } MethodType;
```

Le paramètre *PSKKeyExchangeMethod* fournit une liste de méthodes d'échanges de clés supportées par le client. Ici, deux méthodes sont supportées : RSA avec PSK et DH avec PSK.

6.3.1.2 La structure du message *ServerKeyExchange*

Lorsque le serveur reçoit le message *ClientHello*, il répond par son message *ServerHello* comme il est décrit dans le premier chapitre. Le serveur envoie ensuite son message *ServerKeyExchange*. Le but de ce message est de transporter la clé publique du serveur afin de permettre au client d'échanger le *premaster secret*.

Le message *ServerKeyExchange* peut se présenter comme suit :

```
struct {
    select (KeyExchangeAlgorithm) {
     case diffie_hellman:
        ServerDHParams params;
        Signature signed_params;
     case rsa:
        ServerRSAParams params;
        Signature signed_params;
     case rsa_psk: /*NEW/
        ServerRSAParamsPSK params;
        Signature signed_params; /*optional/
     case diffie_hellman_psk: /*NEW/
        ServerDHParamsPSK params;
        Signature signed_params;/*optional/
    };
} ServerKeyExchange;
```

Les lignes en *italic* sont celles ajoutées par notre protocole au message standard. Dans le cas où le serveur aurait choisi la méthode RSA avec PSK (rsa_psk), il envoie son *ServerKeyExchange* qui contient sa clé publique RSA. Le serveur de TLS standard doit envoyer soit une clé publique d'un certificat, soit une clé publique signée par un certificat (dans le cas où le certificat ne sert qu'à signer). Afin de garder l'interopérabilité, nous avons gardé cette option de signature de la clé publique, mais nous rappelons que c'est une option avec KEM-TLS.

6.3.1.3 La structure du message *ClientKeyExchange*

A la réception du *ServerKeyExchange*, le client génère la clé *premaster secret* comme elle est définie dans TLS en fonction de la méthode choisie. Le client chiffre ensuite ce secret et l'envoie via le message *ClientKeyExchange*. Les lignes en *italic* ou en **gras** sont ajoutées au message standard.

```
          struct {
           select (KeyEchangeAlgorithm){
            case rsa: Encrypted Premaster Secret;
            case diffie_hellman: ClientDiffieHellmanPublic;

            case rsa_psk: EncryptPremasterSecretPSK; /*NEW/
or          case rsa_psk:  /*NEW/
                   opaque psk_identity<0..2^16-1>;
                   Encrypted Premaster Secret;

            case diffie_hellman_psk: ClientDiffieHellmanPublicPSK; /*NEW/
or          case diffie_hellman_psk:  /*NEW/
                   opaque psk_identity<0..2^16-1>;
                   ClientDiffieHellmanPublic;

            } exchange_key;
          } ClientKeyExchange;
```

Puisque la clé pré partagée est identifiée par un identificateur, le client doit envoyer cet identificateur afin d'aider le serveur à retrouver la clé correspondante dans sa base de données de clés. A cet effet, deux méthodes sont possibles afin de permettre au client d'envoyer cet identificateur au serveur :

- La première méthode consiste à envoyer l'identificateur *psk_identity* de la clé en clair (ligne en gras). Cette méthode est utilisée si la protection d'identité n'est pas recommandée par le client et/ou par le serveur.
- La deuxième méthode est utilisée lorsque nous voulons protéger l'identificateur de la clé pré partagée. Bien que cette étape ne soit pas possible qu'avec RSA, elle est possible avec Diffie-Hellman en utilisant un mécanisme supplémentaire comme la gestion pseudonyme (définie plus tard dans ce chapitre). Avec la méthode rsa_psk, la *premaster secret* est structurée comme suit:

```
struct {
  ProtocolVersion client_version;
  opaque random[30];
  opaque psk_identity<1..2^16-1>;
  opaque[77] pad[16-psk_identity.length];
} PremasterSecret;

struct { public-key-encrypted PremasterSecret pre_master_secret;
} EncryptedPremasterSecretPSK;
```

Avec cette structuration, l'identificateur de la clé est concaténé avec une valeur aléatoire et un *padding*, construisant ensemble la *premaster secret* qui sera chiffrée avant d'être envoyée au serveur.

[77] Sachant que la taille du *premaster secret* est 48 octets, un padding est ajouté en cas où la taille de l'identificateur de la clé serait moins que 16 octets. Dans TLS, pas de conditions sur la taille du *premaster secret*, mais nous conservons sa taille suggérée dans [TLS] (48 octets). Ce padding aide le serveur à récupérer à la fois la valeur aléatoire et l'identité de la clé pré-partagée

6.3.1.4 Calcul des clés de session et l'établissement de la phase d'authentification

La *premaster secret* est déchiffrée par le serveur afin de récupérer l'identificateur de la clé pré partagée et de calculer la clé *master_secret*. Comme le serveur et le client n'utilisent pas de certificats pour l'authentification, cette dernière est établie en utilisant la clé pré partagée dans le calcul de clés. Nous proposons donc d'introduire la valeur du PSK dans le calcul de la fonction PRF. La clé *master_secret* est alors calculée en remplaçant la *premaster secret* par *premaster secret XOR PSK* comme il est montré dans la formule suivante :

```
master_secret = PRF(pre_master_secret XOR PSK,
                    "master_secret",
                    ClientHello.random + ServerHello.random)[0..47];
```

Si nous supposons qu'un intrus a envoyé sa clé publique à la place de celle du serveur, cet intrus sera obligé aussi de savoir la valeur de la clé PSK. Il n'y a donc que les entités qui ont la valeur PSK qui puissent calculer le *master_secret*.
Après la dérivation de la clé *master_secret*, le client et le serveur calculent la clé *key_block* de la même façon que celle définie par TLS. Une fois cette phase de dérivation de clés terminée, le client et le serveur peuvent échanger leurs messages *Finished*. Comme il est décrit dans TLS, ces messages représentent une preuve que le client et le serveur ont dérivé les mêmes clés (de chiffrement, de MAC, etc.). Dans notre KEM-TLS, les *Finished* représentent un service additionnel : les deux entités ont pris connaissance de la clé PSK. Autrement dit, personne ne peut calculer un *Finished* valide sans avoir le vrai PSK. D'où l'authentification mutuelle par le *Finished*.

6.3.2 Analyse du protocole KEM-TLS

Dans KEM-TLS, l'utilisation de certificats peut être totalement supprimée sans aucune influence sur le niveau de sécurité fourni par TLS.

Nous avons cité que dans l'actuelle spécification de TLS, le certificat est envoyé en clair et donc il n'y a pas moyen d'établir l'anonymat.

Pour le service d'anonymat, plusieurs contributions ; notamment [Rescorla] proposent l'établissement d'un canal de Diffie-Hellman anonyme et ensuite un *Handshake* complet à l'intérieur du canal. Cette solution n'est pas assez simple et elle est coûteuse en terme du calcul cryptographique, de charge protocolaire et de gestion. Double-TLS apparaît comme une solution pour l'anonymat. Une autre solution peut être établie en activant le chiffrement/déchiffrement avant d'envoyer le certificat du client. Pour cela, nous pouvons par exemple envoyer le message *ChangeCipherSpec* avant le certificat du client. Le client chiffre donc son certificat avant d'être envoyé au serveur. Bien que cette solution apparaisse appropriée, il faut lui consacrer une analyse plus profonde ; notamment sur son effet sur la machine protocolaire de TLS.

Avec KEM-TLS, l'anonymat est assuré nativement si nous choisissons d'utiliser RSA comme une méthode d'échange de clés. Avec Diffie-Hellman, un mécanisme supplémentaire doit être appliqué (voir section 6.3.2.3) pour l'anonymat.

6.3.2.1 La clé partagée remplace-t-elle le certificat

Nous rappelons que l'utilisation d'un mécanisme à clé publique basé sur des certificats a comme objectif de minimiser la complexité de la gestion des clés et d'offrir le service de non répudiation. A l'exception de la non répudiation, ce mécanisme n'a aucun intérêt par rapport à l'authentification par une clé pré partagée ; spécialement pour les environnements fermés et

mobiles. En effet, l'utilisation de PKIs par ces environnements nécessite plus de gestion, de bande passante, de la mémoire, de la capacité de CPU et d'autres paramètres formant ensemble le goulot d'étranglement pour ces environnements. D'où l'avantage d'utiliser les PSKs à la place des certificats et des PKIs.

Avec TLS-certificats, le service PFS n'est pas possible que dans deux cas précis : lorsque le serveur a un certificat pour signer et non plus pour chiffrer[78], ou lorsqu'il utilise des clés Diffie-Hellman anonymes signées obligatoirement par le serveur. Ces deux cas augmentent significativement le calcul cryptographique et la charge protocolaire de TLS. Ils nécessitent plus de chiffrement/déchiffrement asymétriques et plus d'échanges liés aux clés Diffie-Hellman.

Avec KEM-TLS, le client utilise toujours des clés RSA ou DH anonymes. Une différence majeure par rapport à TLS-certificats est que le serveur ne signe pas ses clés. Avec TLS, si le serveur ne signe pas ses clés, le serveur et le client restent sous le risque de l'attaque MitM. Avec KEM-TLS, le serveur s'authentifie auprès du client en offrant une preuve qu'il a la clé partagée. Cette preuve est fournie durant le calcul du message *Finished*. En effet, ce message est chiffré avec une clé dérivée du *master_secret* qui est calculée, entre autres, en fonction du PSK.

Plusieurs propositions proposent donc l'utilisation de PSK avec TLS mais aucune d'entre elles n'assure la protection d'identité et le PFS. En effet, dans ces propositions, l'identificateur de la clé transite toujours en clair sur le réseau. Même si l'identificateur de la clé n'indique aucune information, sa réutilisation à long terme peut permettre à un intrus d'employer l'analyse de trafic sur les identités des deux parties communicantes.

Le PFS n'est pas fourni par ces propositions parce qu'elles utilisent toujours la même clé dans la dérivation des clés de la session. Si un intrus arrive à récupérer la clé partagée, il peut alors déchiffrer toutes les sessions déjà établies. Avec KEM-TLS, ce n'est pas le cas puisque le serveur et le client utilisent une conjonction des deux clés secrètes : une partagée et une autre anonyme et temporaire générée durant chaque session. Par conséquent, si un intrus arrive à retrouver la clé partagée, il ne peut pas reproduire la clé *master_secret* de la session correspondante. Si l'intrus arrive à avoir la clé partagée, il doit casser encore DH ou RSA (ce n'est pas le cas aujourd'hui) ou récupérer la clé privée temporaire du serveur[79]. Même s'il arrive à récupérer la clé privée DH ou RSA, l'intrus ne peut pas déchiffrer que la session concernée. Les anciennes sessions restent protégées.

Il reste à noter que les autres contributions du PSK avec TLS ne résistent pas contre les attaques par dictionnaire et les espionnages passifs. A notre avis, la lutte appropriée contre ce type d'attaques est d'utiliser le chiffrement asymétrique et de générer de clés par session.

6.3.2.2 La charge protocolaire et cryptographique

Notre protocole enlève toutes les opérations cryptographiques liées à l'utilisation de certificats et de la PKI. Il épargne donc toute opération asymétrique exigée par ce type d'authentification. Cela réduit énormément non seulement la charge cryptographique mais aussi la charge protocolaire et donc le nombre de RTs nécessaires pour échanger et transporter les données liées aux messages de la phase d'authentification.

[78] En supposant que le serveur génère de clés asymétriques par chaque session.
[79] La clé temporaire doit être détruite d'une manière sécurisée après son utilisation.

6.3.2.3 Gestion pseudonyme avec KEM-TLS

Contrairement à d'autres propositions, notre protocole n'exige pas beaucoup de conditions, ni sur la taille de la clé, ni sur sa complexité. Autrement dit, les clients peuvent utiliser des clés faibles comme par exemple un mot de passe facile à se rappeler.

Comme nous avons déjà cité, lorsque DH est choisie comme une méthode d'échange des clés, l'identificateur de la clé passe en clair sur le réseau. A cet effet, nous proposons d'ajouter un mécanisme qui permet une gestion de pseudonyme afin de fournir un anonymat complet lorsque nous utilisons DH.

Afin d'appliquer ce mécanisme, nous proposons d'ajouter un *seed* à chaque couple (psk_identity, psk) et nous rafraîchissons le seed et le psk pour chaque session correctement établie. Avec ce mécanisme et durant la session i, le client et le serveur mettent à jour le seed et la psk de la manière suivante :

```
seed(0) est une valeur aléatoire sur 16 bytes.
seed(i) = P_MD5(seed(i-1) XOR psk_identity,
                "seed" +
                ClientHello.random + ServerHello.random)[0..16];
psk(i) = PRF(psk(i-1) XOR premaster secret[80](i), "pre shared key",
                ServerHello.random + ClientHello.random)[0..48];
```

Avec ce mécanisme, le *psk_identity* reste inchangé et ne passe jamais en clair sur le réseau, d'où l'anonymat. Cependant, lorsqu'un client se connecte à un serveur, il envoie le *seed* (*seed*(i-1) calculé durant la dernière session, session(i-1)) à la place du *psk_identity*. Le reste du KEM-TLS reste inchangé. Une carte à puce peut être utilisée afin de stocker, d'une manière plus sécurisée, les crédits du client.

6.4 Conclusion

Dans ce chapitre, nous avons présenté une solution qui fait cohabiter le PSK avec le chiffrement asymétrique sans aucune utilisation de PKI. Elle concerne particulièrement la conception d'un protocole basé sur TLS pour l'authentification mutuelle entre deux entités sans utilisation de certificats. Elle est élaborée en utilisant une clé pré partagée dans la procédure d'authentification et le chiffrement asymétrique dans la dérivation des clés.

Cette solution assure plusieurs services additionnels par rapport à la norme TLS. Ces services deviennent de plus en plus indispensables, non seulement pour les réseaux mobiles, mais aussi pour les fixes. Parmi ces services, nous citons la protection d'identité et la protection contre les attaques rétroactives sur la confidentialité des données (*Perfect Forward Secrecy*). La protection contre les attaques passives et par dictionnaire est ainsi améliorée.

Notons ici que cette solution a été proposée au groupe de travail TLS de l'IETF. Après la réunion 60 de l'IETF [Pasi] en août 2004, le groupe TLS de l'IETF a décidé de publier un standard unifiant notre solution et une autre, proposée par Nokia et Siemens ([Bad0], [TLSPSK]).

[80] La *premaster secret* de la session en cours de construction.

Chapitre 7

Conclusion et perspectives

Résumé du contenu

7.1 Conclusion

Les réseaux sans fil présentent des exigences particulières en terme de délai, de débit, de capacité de CPU et de mémoire. Ils introduisent de nouveaux problèmes non rencontrés dans les réseaux fixes ; notamment la partition du support de transmission et l'absence de mécanisme de distribution de clés au niveau liaison. La conception d'un protocole de sécurité doit donc respecter les limitations et les contraintes des réseaux mobiles et doit savoir répondre aux besoins de ces réseaux tels que l'anonymat, la protection à long terme de données et la protection contre les attaques passives. Dans cette optique, nous avons étendu et élargi TLS avec d'autres mécanismes d'authentification et de distribution de clés.

Dans un premier temps, nous avons analysé TLS et nous avons évalué sa performance et sa charge protocolaire sur les réseaux fixes et sans fil. Nous avons étudié aussi les failles et les besoins de sécurité dans deux cas du réseau sans fil. Dans le cas du WAP, la sécurité de bout en bout n'est pas fournie par les spécifications du WAPv1.x et elle devient coûteuse pour les mobiles dans la version 2. Nous avons donc proposé deux solutions. La première consiste à insérer un tiers de confiance entre le client mobile et le serveur d'application. Son rôle sert à réduire les lourdes charges protocolaires et cryptographiques de TLS sur le mobile. Cette solution est assurée sans aucune influence sur le niveau de la sécurité. La seconde solution consiste à établir une authentification mutuelle entre le client et le serveur par une clé pré partagée ou par une clé délivrée par un tiers de confiance. Bien que TLS soit basé sur les certificats dans son mécanisme d'authentification, cette solution de clés pré partagées présente un axe de base pour l'intégration de PSK avec TLS.

Ensuite, nous avons traité plusieurs méthodes d'authentification basées sur l'usage de TLS pour les réseaux 802.11 sans fil. Dans la nature de 802.11 sans fil, le client mobile ne peut pas avoir un accès Internet avant d'être authentifié. Autrement dit, le client ne peut pas vérifier si le certificat du serveur d'authentification a été révoqué ou pas. Les différentes méthodes présentées dans le

chapitre III de cette thèse n'assurent pas les moyens pour répondre à cet effet. Ces méthodes s'appuient sur TLS afin d'établir un canal permettant la protection des échanges des méthodes EAP comme MSCHAP ou MD5 et l'anonymat du client. Malheureusement, ces méthodes ne sont pas toujours protégées contre l'attaque MitM comme nous avons montré dans le chapitre III. Nous avons proposé plusieurs solutions pour remédier à cet inconvénient. Le protocole Double-TLS est un protocole anonyme, dynamique dans le sens où il génère des clés dynamiques pour chaque session, moins coûteux en terme de calcul cryptographique et de la charge protocolaire et il résiste contre l'attaque MitM grâce à l'authentification mutuelle sans aucune obligation d'utiliser les certificats et les PKIs.

La communauté de TLS à l'IETF est fortement intéressée par l'intégration de nouvelles méthodes d'authentification non basées sur des certificats. A son origine, le PKI est inventé afin de faciliter la gestion de clés pour des réseaux à grande échelle. Dans nos exemples de réseaux sans fil ou et de réseaux fermés, les clients peuvent être pré configurés. En plus, la plupart de réseaux sans fil intègrent par défaut des mécanismes pour la gestion de clés. Nous avons montré aussi l'influence majeure de PKI sur la performance des réseaux sans fil et filaire. Nous avons répondu avec la contribution *TLS Express* qui a comme objective d'intégrer l'usage de clés pré partagées avec TLS et de réduire au minimum la charge de TLS. Elle est utilisée aujourd'hui dans le protocole d'authentification EAP-FAST de CISCO et un *Internet Draft* unifiant les deux contributions est en cours de construction.

Nous avons noté que plusieurs solutions basées sur l'utilisation de clés pré partagées avec TLS sont disponibles et nous avons montré que toutes ces solutions, y compris TLS Express, utilisent une clé statique et symétrique pour l'authentification et pour la dérivation de clés de session.

Nous avons traité l'utilisation des clés statiques et le résultat dramatique de cet usage dans le cas où cette clé deviendrait compromise par un attaqueur. Ajoutons à cela, ces solutions n'assurent ni le PFS ni l'anonymat des échanges. Dans la contribution KEM-TLS, nous avons proposé d'intégrer l'usage de PSK et de PKC avec TLS. Cette solution assure les même services de TLS et elle intègre la protection d'identité, l'anonymat, le PFS, elle améliore aussi la résistance contre les attaques passives. Elle n'utilise plus le PKI et donc sa charge est énormément réduite en comparaison avec TLS.

Enfin, nous notons que cette solution a été soumise au groupe de travail TLS au sein de l'IETF. Elle représente aujourd'hui l'élément principal du prochain standard de TLS avec PSK. Après sa première publication comme un *Internet Draft*, TLS-IETF l'a publié à nouveau dans la version 1 du [TLSPSK].

7.2 Perspectives

TLS s'est imposé comme le protocole de la sécurisation des échanges. Deux facteurs ont été à l'origine de cette popularité : le premier est son intégration à tous les browsers et le deuxième est la conception qui le caractérise. Son adoption par plusieurs communautés de recherche comme IETF-EAP-WG et 3GPP font de lui le protocole phare de l'authentification et de la négociation de clés pour les réseaux sans fil.

Une de nos perspectives est de continuer à travailler sur les services d'authentification. Nous souhaitons renforcer cette contribution en intégrant une approche temporelle ou temporaire qui consiste à forcer une re-authentification après un temps donné ce qui peut correspondre à de nouveaux modèles économiques pour le contrôle d'accès au réseau. Nous comptons prolonger

l’intégration concrète et pragmatique de la clé partagée dans TLS par des études peut être un peu plus prospectives.

Dans notre démarche nous avons essayé de contribuer à l’évolution de TLS tout en préservant l’interopérabilité avec l’existant. Nous souhaitons continuer dans ce sens pour intégrer d’autres services comme les VPNs. Ce service est l’un des plus attendus et l’idéal serait d’avoir un seul protocole comme support des services de sécurité.

Néanmoins, l’extension "intensive" de TLS peut conduire à une "inondation" vrai à termes à la non interopérabilité. Il doit y avoir une limite de TLS qui devrait nous faire basculer à la conception d’un nouveau protocole qui devra intégrer en natif l’ensemble des extensions de TLS. La question qui est et sera toujours posé par la communauté est : "y-t-il de la place pour un nouveau protocole ?". Notre réponse à cette question est bien entendu "oui" tout en mettant le nécessaire par le faire valoir.

References

[3DES] ANSI X9.52-1998, "Triple Data Encryption Algorithm Modes of Operation", American National Standards Institute, 1998.

[3GPP] The 3rd Generation Partnership Project (3GPP), http://www.3gpp.org.

[3GPP2] The Third Generation Partnership Project 2 (3GPP2), http://www.3gpp2.org.

[802.11-part1] IEEE Std. 802.11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", 1997.

[802.11-part2] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (Phy) specifications - High-speed Physical Layer in the 5 GHz Band", 1999.

[802.11i] IEEE Draft P802.11i/D6, "Specification for Enhanced Security", October 2003.

[802.1X] IEEE Draft P802.1X/D11, "Standards for local and metropolitan area networks: Standard for port based network access control", Mars 2001.

[802.1XAD] IEEE P802.1X Approved Draft, "Port based Network Access Control", June 2001

[802.1X-Sec] M., Mishra, and W., Arbaugh, "An initial Security Analysis of the IEEE 802.1X Standard", http://www.cs.umd.edu/~waa/1x.pdf, February 2002.

[802Hack] W., Arbaugh, N., Shankar, and Y., Wan, "Your 802.11 Wireless Network has No Clothe" http://www.cs.umd.edu/~waa/wireless.pdf, 2001.

[APDU] ISO 7816-4 SmartCard Standard: Part 4: Interindustry Commands for Interchange.

[AFG] S., Aust, N.A., Fikouras and C., Grg, Enabling Mobile WAP Gateways Using Mobile IP. Proceedings of European Wireless, Florence, Italy, February 2002.

[ASN1-Part1] Specification of Abstract Syntax Notation One (ASN.1), ITU-T Recommendation X.208, 1988.

[ASN1-Part2] Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), ITU-T Recommendation X.209, 1988.

[ASN-DER] Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ITU-T Recommendation X.690, 2002.

[Bench-SC] H., Handschuh, P., Paillier, "Smart Card Crypto-Coprocessors for Public-Key Cryptography", vol. 1820 of Lecture Notes in Computer Science, page(s): 386-394, Springer-Verlag, 2000.

[BLEI] D., Bleichenbacher, "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1" inAdvances in Cryptology -- CRYPTO'98, LNCS vol. 1462, pages: 1-12, 1998.

[Book-SC] W., Rankl and W., Effing, "Smart Card Handbook", John Wiley and Sons Ltd, ISBN 0-471-98875-8, 2000.

[BSWN] Khan J., A. Khwaja, 2003, Building Secure Wireless Networks with 802.11, Widely, Indiana.

[CAT] ETSI TS 102 223, "Smart cards; Card Application Toolkit (CAT), (Release 6)", V6.1.0, 2003.

[CBCATT] B., Moeller, "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures", http://www.openssl.org/~bodo/tls-cbc.txt.

[CHAP] W., Simpson, "PPP Challenge *Handshake* Authentication Protocol (CHAP)", RFC 1994, August 1996.

[CHAPv2] G., Zorn, "Microsoft PPP CHAP Extensions, Version 2", RFC 2759, January 2000.

[Cisco-Draft] N., Cam-Winget, et al., "A TLS Hello Extension for Ticket Based Pre-Shared Keys", draft-salowey-tls-ticket-00.txt, (work in progress), May 2004.

[CORBA] The Object Management Group, "Common Object Request Broker Architecture Specification 2.2", http://www.omg.org.

[CRL] S., Boeyen, et. al., "Internet X.509 PKI operational protocols-LDAPv2", RFC2559, April 1999.

[CSM] R., Housley, "Cryptographic Message Syntax", RFC 3369, August 2002.

[CRT] S.M., Yen, et. al., "RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis", IEEE Transactions on Computers, Volume 52, Issue 4, April 2003, Pages:461-472.

[CTRB] J., Grobschadl, "The Chinese Remainder Theorem and its Application in a High-Speed RSA Crypto Chip", http://www.acsac.org/2000/papers/48.pdf

[DES] ANSI X3.106, "American National Standard for Information Systems-Data Link Encryption," American National Standards Institute, 1983.

[DESHack] E., Biham, A., Shamir, "Differential Cryptanalysis of the Data Encryption Standard", Springer Verlag, 1993.

[Matsui] M., Matsui., "Linear Cryptanalysis Method for DES Cipher", EUROCRYPT 1993. pp386-397

[DH] W., Diffie and M.E., Hellman, "New directions in cryptography", IEEE Transactions on Information Theory, Volume 22, Numero 6, Pages: 664-654, 1976.

[DoS] Denial of Service (DoS) Attack Resources, available at http://www.denialinfo.com.

[DSS] The National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", May 1994.

[DTLS] E., Rescorla, N., Modadugu, "Datagram Transport Layer Security", draft-rescorla-dtls-01.txt, Internet Engineering Task Force Draft, July 2004.

[EAP] L., Blunk, and J., Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, March 1998.

[EAP-Bis] L., Blunk, et al., "Extensible Authentication Protocol (EAP)", Draft-ietf-eap-rfc2284bis-06.txt, October 2003.

[EAPFast] N., Cam-Winget, et. al., "EAP Flexible Authentication via Secure Tunneling (EAP-FAST)", draft-cam-winget-eap-fast-00.txt, (work in progress), February 2004.

[EFF] EFF DES cracker, available at http://en.wikipedia.org/wiki/EFF_DES_cracker.

[EPA-SC] P., Urien, et al., 2004. "EAP support in smartcards", draft-urien-eap-smartcard-04.txt, *Internet Draft* (work in progress), August 2004.

[EAP-TTLS] P., Funk, and S., Blake-Wilson, "EAP Tunneled TLS Authentication Protocol (EAP-TTLS)", draft-ietf-pppext-eap-ttls-04.txt, Internet-Draft (work in progress), April 2004.

[EAP-TLS] B., Aboba, and D., Simon, "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999.

[EPIS] "Projet RNRT-EPIS", http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_02_11.htm.

[FTP] J., Postel, and J., Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[GPNRR] G. M., Ziegler, "The Great Prime Number *Record* Races", notices of the AMS, Volume 51, number 4, 2003.

[Hayder] H., Saleh, "Une Architecture Novatrice de Sécurité à base Carte à Puce Internet", Thèse, 5 juin 2002.

[HMAC] H., Krawczyk, M., Bellare, and R., Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[HBook] H., Labiod, and H., Afifi, "De Bluetooh à Wi-Fi", Hermes Sciences, 2004.

[HSRSA] C., Koç, "High-Speed RSA Implementation", RSA Laboratories, Version 2.0, November 1994.

[HTTP] T., Berners-Lee, R., Fielding, and H., Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[HTTPS] E., Rescorla, "HTTP Over TLS", RFC 2818, May 2000.

[Insecure-WiFi] N., Borisov, I., Goldberg, and D., Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11", In Proceedings of the 7th International Conference on Mobile Computing and Networking, Rome Italy, July 2001.

[IPSec] S., Kent and R., Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[IPSec-AH] S., Kent and R., Atkinson, "IP Authentication Header (AH)", RFC 2402, November 1998.

[IPSec-ESP] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[ISAKMP] D., Maughan, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[JFK] W., Aiello, S. M., Bellovin, M., Blaze, R., Canetti, J., Ioannidis, A. D., Keromytis, and O., Reingold, "Just Fast Keying (JFK)". IETF *Internet Draft* draft-ietf-ipsec-jfk-04.txt, (Expired), July 2002.

[L2TP] W., Townsley, et al., "Layer Two Tunneling Protocol (L2TP)", RFC 2661, August 1999.

[LDAP] W., Yeong, T., Howes and S., Kille, "Lightweight Directory Access Protocol", IETF RFC 1777, March 1995.

[LDAPv3] M., Wahl, T., Howes, and S., Kille, "Lightweight Directory Access Protocol (v3)", December 1997.

[MD5] R., Rivest, and S., Dusse, "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[MD5C] X., Wang, et. al., "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", Crypto2004.

[MEAP] N., Asokan, et. al., "Man-in-the-middle in Tunnelled Authentication Protocols", available at http://eprint.iacr.org/2002/163.

[Microsoft] D., Simon, "Addition of Shared Key Authentication to Transport Layer Security (TLS)", Draft-ietf-tls-passauth-00, (Expired), November 1996.

[MIME] N., Borenstein, and N., Freed, "MIME (Multipurpose Internet Mail Extension) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993.

[MITM1] Nokia Research Center, "*Man-In-The-Middle* attacks in tunneled authentication protocols", http://www.saunalahti.fi/~asokan/research/mitm.html, October 2002

[MITM2] J., Puthenkulam, V., Lortz, D., Dan Simon, and A., Palekar, "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04.txt, 27 October 2003.

[MSS] "Généralités sur la sécurité", Microsoft, 2004.

[NAI] B., Aboba, and M., Beadles, "The Network Access Identifier", RFC 2486, January 1999.

[NAT] K., Egevang, and P., Francis, "The IP Network Address Translator (NAT)", RFC 1631, May 1994.

[NESSIE] M., Ciet, and F., Sica, "Report on the Performance Evaluation of NESSIE Candidates I", Version 3.3, 20 November 2001.

[Nokia] P., Eronen, and H., Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", draft-ietf-tls-psk-00, (work in progress), May 2004.

[OCSP] M., Myers, R., Ankney, A., Malpani, S., Galperin, and C., Adams, "Internet X.509 Public Key Infrastructure: Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[Palm] A., Weimerskirch, C., Paar, and S. C., Shantz, "Elliptic Curve Cryptography on a Palm OS Device", The 6th Australasian Conference on Information Security and Privacy (ACISP 2001), 11-13 July 2001, Macquarie University, Sydney, Australia.

[Pasi] P., Eronen, "Some notes and thoughts from IETF60", August 2004, available at: http://www.imc.org/ietf-tls/mail-archive/msg04240.html.

[PEAP] A., Palekar, et al., "Protected EAP Protocol (PEAP)", draft-josefsson-pppext-eap-tls-eap-07.txt, Internet-Draft (work in progress), October 2003.

[PKI-CRL] R., Housley, W., Polk, W. Ford, and D. Solo, "Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[POP] J., Myers, M., Rose, "Post Office Protocol - Version 3", RFC 1939, May 1996.

[PPP] W., Simpson, "The Point-to-Point Protocol (PPP)", RFC 1661, July 1994.

[PPP-3DES] K., Hummert, "The PPP Triple-DES Encryption Protocol (3DESE)", RFC 2420, September 1998.

[PPP-DES] K., Sklower, and G. Meyer, "The PPP DES Encryption Protocol, Version 2 (DESE-bis)", RFC 2419, September 1998.

[PPTP] K., Hamzeh, et al., "Point-to-Point Tunneling Protocol (PPTP)", RFC 2637, July 1999.

[RADIUS] C., Rigney, A., Rubens, W., Simpson, and S., Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RC4-Hack] S., Fluhrer, I., Mantin, and A., Shamir, "Weaknesses in the key scheduling of RC4", Proc. 8th Ann. Workshop Selected Areas in Cryptography, Springer, 2001, pp. 1-24.

[RC4C] Knudson, et. Al., "Analysis Methods for RC4", University of Bergen, Belguim 1999.

[Rescorla] E., Rescorla, "SSL and TLS- Designing and Building Secure Systems", Addison-Wesley, 2000.

[RFC2865] C., Rigney, A., Rubens, W., Simpson, and S., Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC2866] C., Rigney, "RADIUS Accounting", RFC 2866, June 2000.

[RFC2869] C., Rigney, W., Willats, and P., Calhoun, "RADIUS Extensions", RFC 2869, June 2000.

[RFC3162] B., Aboba, G., Zorn, and D., Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

[RSA] B. Kaliski., J. Staddon., "PKCS #1: RSA Cryptography Specifications", Version 2.0, RFC 2437, October 1998.

[SRPTLS] D., Taylor, et. al., "Using SRP for TLS Authentication", draft-ietf-tls-srp-08.txt, *Internet Draft* (work in progress), August 2004.

[Samfat] D., Samfat, "Architecture de sécurité pour réseaux mobiles", Thèse à l'ENST, January 1996.

[SC-hack] M., Lassus, "Smart-card a cost effective solution against electronic fraud", European Conference on Security and Detection, ECOS 97, page(s): 81-85, 1997.

[SC-Security] S-c., C., Chan, "An Overview of Smart Card Security", available at http://home.hkstar.com/~alanchan/papers/smartCardSecurity/

[SC-WEW] J., Rees and P., Honeyman, "Webcard: A Java Card web server", Proceeding of 4th IFIP Smart Card Research and Advanced Application Conference (CARDIS), Bristol, UK, September 2000.

[SFA] "SSL-TLS : Sécurisation de flux applicatifs", available at http://cryptosec.lautre.net.

[SHA] Secure Hash Standard, NIST FIPS PUB 180-1, National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, May 1994.

[SIM] Digital cellular telecommunications system (Phase 2+), "Security mechanisms for SIM application toolkit; Stage2", (3GPP TS 03.48 version 8.8.0 Release 1999).

[SIM-ME] GSM Technical Specification GSM 11.11, "Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM – ME) interface", Version 5.0.0, December 1995.

[SIM-WEB] S., Guthery, R., Kehr and J., Posegga, "How to turn a GSM SIM into a Web Server", Proceeding of 4th IFIP Smart Card Research and Advanced Application Conference (CARDIS), Bristol, UK, September 2000.

[SMIME] B., Ramsdell," S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[SMTP] J., Klensin, "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[SMS] Digital cellular telecommunications system (Phase 2+), "Technical realization of the Short Message Service (SMS) Point-to-Point (PP)", (3GPP TS 03.40 version 7.5.0 Release 1998), ETSI TS 100 901 V7.5.0 (2001-12).

[SMS-1] Digital cellular telecommunications system (Phase 2+), "Technical realization of the Short Message Service (SMS) Point-to-Point (PP), (GSM 03.40)", (3GPP TS 03.40 version 7.5.0 Release 1998).

[SOCKS] "The Source for SOCKS Technology", http://www.socks.permeo.com.

[SSL] A., Freier, P., Karlton, and P., Kocher, “The SSL protocol, version 3.0”, *Internet Draft*, March 1996.

[SWAP] "Projet RNRT-SWAP", http://www.telecom.gouv.fr/rnrt/projets/res_00_71.htm.

[T0-T1] Smart Cards, "Transport Protocol for UICC based Applications, Stage 1 (Release 6)", V6.0.0, 2003.

[TCP] W. R., Stevens, "TCP/IP illustrated volume 1". Reading, Massachusetts: Addison-Wesley, 1994.

[Telnet] J., Postel, and J., Reynolds, "Telnet Protocol Specification", RFC 854, May 1983.

[TLS] T., Dierks, and C., Allen, "The TLS protocol, Version 1.0", RFC 2246, January 1999.

[TLS1.1] T., Dierks, and E., Rescorla, "The TLS Protocol Version 1.1", INTERNET-DRAFT, draft-ietf-tls-rfc2246-bis-08.txt, August 2004.

[TLS-Extension] S., Black-Wilson, et. al., "Transport Layer Security (TLS) Extensions", RFC3546, June 2003.

[TLSPSK] P., Eronen, et al., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", draft-ietf-tls-psk-01 (work in progress), August 2004.

[UDP] J., Postel, "User Datagram Protocol", RFC 768, August 1980.

[Urien] P., Urien, "Sécurité 802.11 Wi-Fi", ENST, 2003.

[WAE] Forum WAP, "Wireless Application Environment Specification", Version 2.0, Feb-2002

[WAPArch] Forum WAP, "WAP Architecture", Version 12-July-2001

[WEP-Hack] W. A., Arbaugh, "An inductive chosen plaintext attack against WEP/WEP2", http://www.cs.umd.edu/~waa/attack/s0003.htm, Mai 2001.

[WDP] Forum WAP, "Wireless Datagram Protocol", Version 14-Jun-2001

[WEP-WPA] B.R., Miller, and B.A., Hamilton, "Issues in Wireless Security WEP, WPA& 802.11i", Proceedings of the 18th Annual Computer Security Applications Conference, December 2002.

[Wireless-Hack] William A., Arbaugh, N., Shankar, and Y.C., Justin Wan, "Your 802.11 Wireless Network has No Clothes", In IEEE International Conference on Wireless LANs and Home Networks, Singapore, December 2001.

[WML] Forum WAP, "Wireless Markup Language Version 2.0", 26-June-2001.

[WMLS] Forum WAP, "*WMLScript* Standard Libraries Specification", 25-Sep-2000.

[WMLSC] Forum WAP, "*WMLScript* Crypto Library", Version 20-Jun-2001.

[WPKI] Forum WAP, "Wireless Application Protocol: Public Key Infrastructure Definition", Version 24-Apr-2001.

[Wsec] Miller Steward S., 2003, WiFi Security, McGraw-Hill, New-York.

[WSP] Forum WAP, "Wireless Application Protocol: Wireless Session Protocol Specification", Approved Version 5-July-2001

[WTLS] Forum WAP, "Wireless Transport Layer Security", Version 06-Apr-2001.

[WTP] Forum WAP, "Wireless Transaction Protocol", Version 10-Jul-2001

[WTP] Forum WAP, "WAP Wireless Telephony Application", Version 08-Sep-2001

[X.509] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1997, Information technology - Open Systems Interconnection - The Directory: Authentication framework.

[XDR] R. Srinivansan, "XDR: External Data Representation Standard", RFC-1832, August 1995.

Annexe A

La sécurité des réseaux et des systèmes informatiques dépend, entre autres, de la conception des mesures de sécurité appropriées. Ces dernières doivent prendre en compte l'ensemble des risques et menaces encourus par une application particulière.

Les menaces représentent tout événement malveillant et indésirable sur les différentes applications. Les failles d'une application ou du système d'exploitation facilitent la concrétisation des menaces. Une attaque menée contre une application désigne l'action malveillante d'un intrus, cherchant à mettre à profit certaines vulnérabilités pour donner corps à la menace. D'où le risque des dégâts que l'attaque peut infliger à l'application.

La protection du système d'exploitation peut être assurée en évaluant les menaces et les risques et en déterminant le type de menaces susceptibles d'affecter les applications. Ces menaces peuvent être classées par priorité en fonction des dégâts potentiels qu'elles peuvent provoquer contre les ressources [MSS]. Pour la protection des échanges, ces menaces s'étendent sur plusieurs niveaux ; de la falsification de l'identité à l'analyse du trafic. Par exemple, si les échanges ne sont pas protégés par un protocole de sécurité de bout en bout comme TLS ou IPSec, ils peuvent alors être interceptés, modifiés, ou endommagés par les intrus.

Dans cette annexe, nous présentons plusieurs attaques possibles sur les échanges des données entre deux entités. Ces attaques sont classifiées en deux familles : actives et passives. Une attaque qui peut modifier ou injecter de données dans le réseau appartient à la famille des attaques actives. Les attaques passives consistent à écouter sur les réseaux sans perturber leur fonctionnement et sans modifier les données échangées. Ces attaques sont généralement indétectables.

A.1. Les attaques actives

Une attaque active permet à un intrus de modifier les données échangées sur les réseaux. Dans ce type d'attaques, tous les coups sont permis. L'intrus peut intercepter les échanges, les remplacer par ses propres échanges et il peut aussi enregistrer des échanges et les rejouer.

Plusieurs moyens sont possibles pour réaliser une attaque active ; notamment :

- Attaque en force brute sur les clés du système. Il s'agit d'essayer toutes les clés possibles par un algorithme particulier afin de trouver le texte en clair.
- Cryptanalyse des codes en connaissant des morceaux de texte en clair du document (En-tête HTTP, Tags HTML, etc.).
- Rejouer des messages enregistrés : il s'agit d'enregistrer des trames et de les renvoyer sans forcément connaître le contenu exact de celles-ci.
- L'attaque *Denial of Service* : consiste à paralyser temporairement et de rendre indisponible des serveurs afin qu'ils ne puissent servir et répondre aux requêtes de ses utilisateurs légitimes. Pour y parvenir, cette attaque peut exploiter des vulnérabilités et faiblesses dans la conception ou l'implémentation des protocoles, des services et des applications. Par exemple, elle peut survenir quand un système est soumis à un tel engorgement du trafic qu'il est dans l'incapacité de traiter les demandes de services légitimes.

- L'attaque *Man-In-The-Middle* : c'est une redirection complète d'une connexion entre deux machines. Chacun des deux interlocuteurs croit dialoguer directement avec l'autre, mais en réalité, il adresse ses données à une troisième machine qui joue le rôle d'un routeur et renvoie les trames modifiées vers le véritable destinataire.

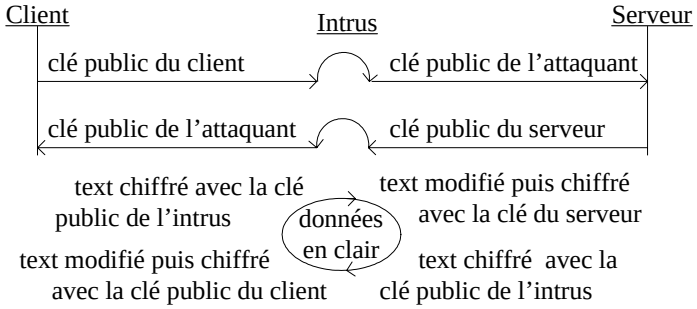

Figure 7.1. L'attaque Man-In-The-Middle

- Changement de messages : il s'agit de modifier le message réel en ajoutant, supprimant ou en changeant le séquence de messages. Cette attaque ôte le facteur de confiance du message et produit du trafic inutilisable.

A.2. Les attaques passives

Avec ce type d'attaques, un utilisateur non autorisé acquit l'accès aux données du réseau. L'intrus ici ne change pas le contenu des messages. Son but est d'espionner la conversation sans l'interrompre. Il s'agit donc d'essayer tout simplement, de découvrir les secrets.

Plusieurs moyens sont possibles pour réaliser une attaque passive, citons quelques exemples:

- Re-jeu (*Replay*) : il s'agit d'intercepter ou d'espionner les paramètres ou les messages d'un canal. Le but de cette attaque et de remplacer, de re-utiliser ou de répéter ces paramètres et messages.
- Espionnage (*Eavesdropping*) : il s'agit d'écouter les transmissions du réseau dans le but d'acquérir l'information circulant entre la source et le destinataire.
- Analyse du trafic (*Traffic analysis*) : il s'agit d'analyser le trafic afin d'obtenir des informations sur les entités communicantes ainsi que sur leurs paramètres (de sécurité).

A.3. Les principaux services de la sécurité informatique

Les échanges de nombreux applications (le commerce électronique, nouveau type de commerce sur Internet), ont lancé le débat sur le climat de confiance entre les participants à l'opération commerciale réelle effectuée par de monnaie dématérialisée.
Dans ce contexte, les menaces sur le déroulement des transactions sont nombreuses. Elles s'étendent sur plusieurs niveaux, de la falsification de l'identité à l'analyse du trafic. La sécurité d'échanges et particulièrement celle du commerce électronique, est donc une opération de bout en bout.

Une communication sécurisée est constituée du facteurs clés qui doivent être proprement rapportés. Parmi ces facteurs, nous citons la confidentialité, l'intégrité, l'authentification et la non répudiation.

- La confidentialité des données consiste à garantir que les données échangées ne sont communiquées qu'aux parties autorisées. Ce service est fourni par les méthodes de cryptographie.
- L'intégrité des données assure que le message n'a pas été modifié entre le moment où il a été expédié et le moment où il a été réceptionné. Les algorithmes de hachage offrent ce service d'assurance que les données reçues par le récepteur sont bien celles envoyées par l'émetteur.
- L'identification et l'authentification des participants sont deux points différents. Le premier consiste à vérifier des caractéristiques individuelles (par exemple le mot de passe). Le second consiste à prouver par des certificats attribués par une autorité de confiance que les participants sont bien ce qu'ils prétendent être. Plusieurs organismes officiels comme RSA Security, VeriSign et Certplus délivrent ce genre de certificats numériques.
- La non répudiation consiste à apporter une preuve, dans l'aspect juridique du terme, que les données sont intègres et leurs sources sont bien celles déclarées. Ce service permet d'éviter la répudiation de la transaction, que ce soit du côté émetteur ou du côté récepteur. La signature numérique est la technique utilisée pour assurer l'authentification des données.

Les techniques utilisées pour assurer les services de sécurité décrits ci-dessus sont définies en cinq points :

1) La cryptographie symétrique à clé secrète est utilisée dans le transfert des données après la phase préliminaire de la transaction. Cette technique de chiffrement est caractérisée par l'utilisation d'une seule clé de la part de l'expéditeur et du destinataire pour le chiffrement et le déchiffrement (Figure ci-dessous). Deux types d'algorithmes de chiffrement à clé symétrique sont définis : les algorithmes de chiffrement en bloc (bloc de données de taille fixe) et les algorithmes de chiffrement en flux.

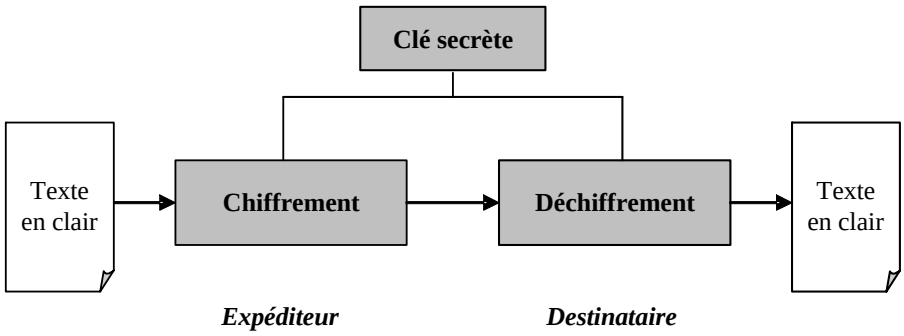

Figure 7.2. Le chiffrement symétrique.

2) La cryptographie asymétrique appelée à clé publique est utilisée dans la phase préliminaire de la transaction. Cette technique de chiffrement est caractérisée par l'utilisation d'une paire de clés pour chaque participant, l'une privée et l'autre publique. Un émetteur A voulant envoyer un message à un récepteur B, il chiffre les données en utilisant la clé publique de B. Le message est reçu par B et déchiffré par sa clé privée que lui seul possède (Figure ci-dessous).

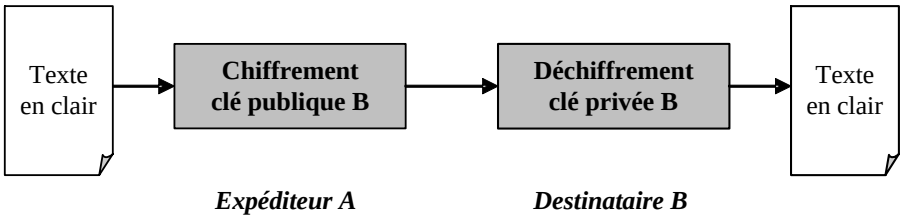

Figure 7.3. Le chiffrement asymétrique.

3) Le hachage est une fonction qui s'applique sur un document donné afin d'assurer son intégrité. Il crée une séquence de bits constituant une "empreinte" unique et infalsifiable. Le document et l'empreinte sont envoyés ensemble au destinataire. A la réception du message, le récepteur répète l'application de hachage sur le document et compare le résultat avec l'empreinte reçue. Aucune différence ne doit être détectée pour accepter l'intégrité des données. La fonction de hachage peut être utilisée avec les cryptographies symétriques ou asymétriques.

4) La signature électronique consiste à créer un condensât d'un document long pour l'employer dans le chiffrement. Ce mécanisme réduit le temps de calcul dans la procédure de chiffrement. Plusieurs types de signatures sont utilisés avec l'application du chiffrement symétrique ou asymétrique. La Figure 7.4 illustre la signature numérique à l'aide d'algorithmes symétriques. C, D et H sont respectivement les fonctions de chiffrement, déchiffrement et hachage.

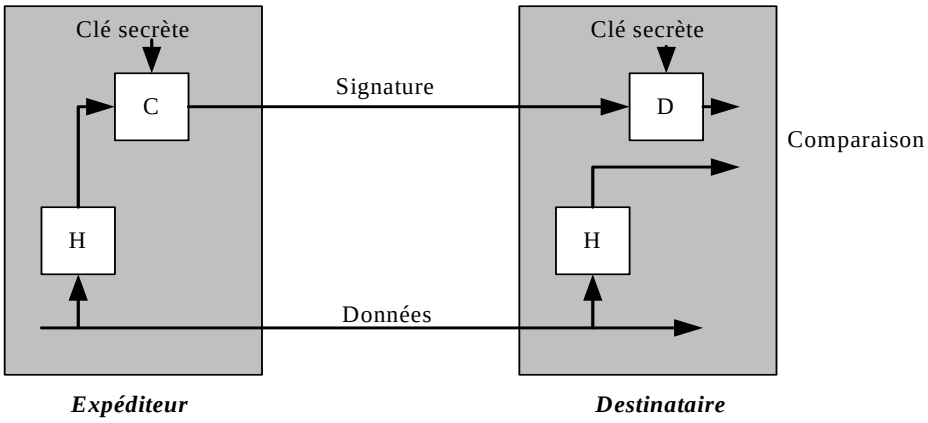

Figure 7.4. La signature numérique.

5) Les certificats sont l'équivalent des cartes d'identités numériques. Délivrés par des autorités de certification, ils sont utilisés dans l'authentification des participants. Le certificat serveur contient une liste d'informations comme le nom du domaine du serveur, la clé publique du serveur à certifier, etc.

A.4. Les Infrastructures à clé publique

Le chiffrement à clé publique ne résout pas complètement le problème de la distribution de clés. En effet, lorsqu'une entité A envoie sa clé publique à l'entité B, il est évident qu'un intrus installé au milieu peut falsifier la clé. Quand les entités A et B veulent communiquer, l'intrus intercepte leurs clés et envoie sa clé publique aux deux entités. Par conséquent, l'intrus prend le message chiffré par la première entité et le déchiffre. Il le modifie et le chiffre avant de l'envoyer à la seconde entité.

Afin de bien résoudre cette problématique, un lien entre l'identité de l'entité et sa clé publique est nécessaire. Afin d'établir ce lien, il est indispensable que la clé publique soit accompagnée d'informations descriptives de son propriétaire et de son usage. A cet effet, les certificats sont définis.

Le certificat est un document électronique contenant donc des données publiques concernant un utilisateur donné. Ce certificat doit pouvoir être distribué sur le réseau et être manipulé par des postes d'utilisateurs. Il est propre à l'entité pour permettre au manipulateur de certificat de trouver sans ambiguïté l'identité de l'entité à laquelle le certificat rapporte.

Le certificat contient la clé publique correspondant à la clé privée que seule l'entité à laquelle se rapporte le certificat connaît. Le contenu d'un certificat (nous ne détaillons pas les différents champs du certificat X.509) doit être garanti comme fiable par une autorité qui a pris la responsabilité de l'émettre. C'est-à-dire, le contenu doit être authentifié par l'autorité qui a pris la responsabilité d'émettre le certificat. Le certificat doit être en même temps infalsifiable et doit indiquer son usage (authentification, signature, chiffrement, etc.). Il est évident aussi que le certificat comporte l'indication de sa validité et un numéro de séquence.

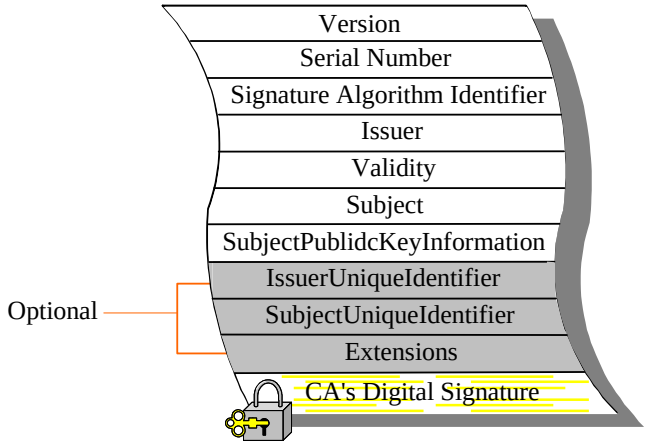

Figure 7.5. Le format du certificat X.509.

Le standard X.509 a normalisé un format de certificat utilisé dans le cadre de l'infrastructure à clé publique. Ce format est utilisé avec le plupart des protocoles de sécurité normalisés tel que SSL, TLS, IPSec et S/MIME. Le certificat est codé en format ASN.1 ([ASN1-Part1], [ASN2-Part1]).

Dans la notion de PKI, le récepteur fait confiance au certificat de l'émetteur via un tiers de confiance (Autorité de Certificat) qui émet et révoque les certificats falsifiés. En face de l'AC, nous trouvons les autorités d'enregistrement (AE) qui se portent garantes du lien entre une clé publique, l'identité du porteur du certificat et d'autres attributs. Le porteur de certificats auxquels sont attribués des certificats. Un composant important dans la PKI est le service de publication. Ce dernier service comprend les répertoires qui contiennent et qui rendent disponibles les certificats des clés publiques et les listes de certificats révoqués.

Le certificat est évidemment signé par le CA émettrice. Lorsqu'une entité reçoit un certificat signé par une CA de confiance, il vérifie la signature de l'autorité qui l'a émis et ainsi son intégrité. Si la signature est correcte, cela signifie que le certificat a bien été émis par le CA indiquée et que son contenu n'a pas été modifié durant son transfert. Ensuite, le récepteur contrôle la période de validité du certificat et/ou vérifie si un certificat a été révoqué ou non.

Dans le cas où il n'y a pas une relation de confiance directe entre le récepteur et la CA émettrice, le récepteur doit vérifier la chaîne de certification correspondante au certificat (liste de certificats de plusieurs CAs). Le récepteur doit premièrement vérifier le chemin de certification afin d'assurer qu'il a tous les certificats jusqu'au plus haut niveau de confiance dans l'arbre de la PKI. Ensuite, le récepteur vérifie la chaîne en commençant par une vérification de la validité du certificat du CA[81] le plus haut dans l'arbre. Le récepteur vérifie après la validité du chemin de certification (figure ci-dessous). Cette étape consiste à vérifier la signature de chaque CA appartenant au chemin de certification.

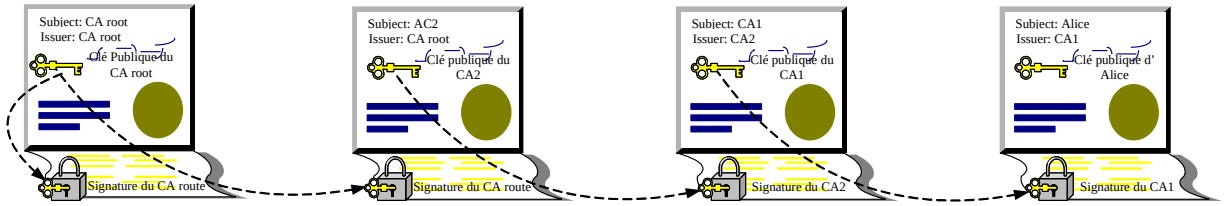

Figure 7.6. Vérification de la chaîne de certification.

[81] Ce CA peut être le CA route dans l'arbre de PKI. Le CA route a un certificat racine auto signé ou auto certifié. Il a les privilèges pour signer son certificat, un certificat dans lequel le champ "*Subject*" est égal au champs "*Issuer*".

Annexe B

Dans cette annexe, nous nous intéressons aux trois formats de signature. Ces formats sont :

B.1. CMS

La CMS décrit une syntaxe d'encapsulation pour la protection de données et elle prend en compte les signatures numériques et le chiffrement. La syntaxe permet des encapsulations multiples ; une enveloppe d'encapsulation peut être insérée à l'intérieur d'une autre enveloppe.

La CMS est suffisamment générale pour pouvoir prendre en compte un certain nombre de types de contenus (*content-types*) différents.

Les spécifications de RFC 3369 définit un contenu de protection nommé *ContentInfo*. Ce dernier encapsule un simple type de contenu identifié. Le type de contenu peut à son tour fournir plusieurs encapsulations.

RFC 3369 définit six types de contenus : les données (*data*), les données signées (*signed-data*), les données mises sous enveloppe (*enveloped-data*), le condensât des données (*digested-data*) et les données chiffrées (*encrypted-data*). Dans cet annexe, nous nous restreignons au type *signed-data*.

Le type de contenu de *signed-data* se compose d'un contenu de n'importe quel type et de zéro à plusieurs signatures. Ce type représente une signature numérique appliquée sur le contenu du type de contenu *data* et il peur être utilisé pour diffuser les certificats et les listes de révocation. Ce type a une structure qui encapsule d'autres champs ; notamment le champ *ContentInfo*. Ce champ a la structure suivante :

```
ContentInfo :: = SEQUENCE {
                        contentType       ContentType,
                        content [0] EXPLICIT ANY DEFINED BY contentType
                        }
ContentType :: = OBJECT IDENTIFIER
```

Le champ *ContentType* est un identifiant d'objet (OID) qui identifie de manière unique le type de contenu.

Le champ *Content* est le contenu lui-même. Il est considéré comme une chaîne d'octets codée en DER.

B.2. PKCS #7

PKCS#7 est un standard ouvert et normalisé par l'IETF. Il définit la syntaxe d'un message cryptographique et il indique un format général pour les messages chiffrés. Ce standard est largement utilisé sur Internet (SMIME, Certificats X509, etc.) et il est avantageux d'assurer une compatibilité au niveau du mobile.

B.3. S/MIME

S/MIME [SMIME] fournit une méthode cohérente pour envoyer et recevoir des données MIME et il offre des services de sécurité cryptographiques pour les applications de messagerie électronique tel que l'authentification, l'intégrité, la confidentialité et la non répudiation.

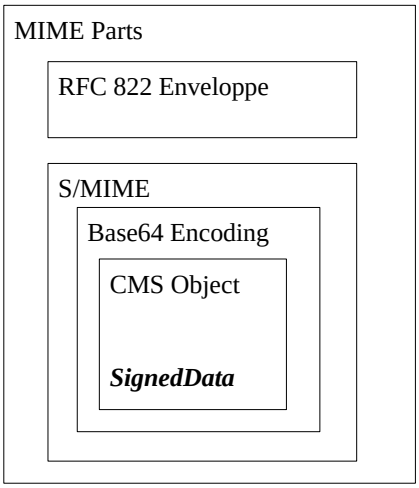

Figure 7.7. Le format du message SMIME

Les messages S/MIME sont une combinaison d'entité MIME et des objets CMS[82] [CSM]. Par contre, des restrictions sont définies sur l'utilisation des objets CMS. En effet, S/MIME n'utilise actuellement que les objets "*content-types*", "*data*", "*signed-data*" et "*enveloped-data*".

Nous nous intéressons à l'objet *signed-data*. La norme S/MIME définit trois types d'attributs "*signedAttributes*" (ils font partis des données signés) liés à cet objet :

- *SigningTime* : contient l'heure à laquelle la signature a été générée au format *UTCTime.*
- *sMIMECapabilities* : contient les informations relatives aux algorithmes de signature et de chiffrement.
- *sMIMEEncryptionKeyPreference* : permet de désigner sans ambiguïté la clé de chiffrement utilisée.

Les informations liées à la signataire sont représentées dans le champ *SignerInfo*. Ce champ contient, entre autres, les paramètres *SignerIdentifier* et *version*.

Le paramètre *SignerIdentifier* identifie le certificat de la signataire et il contient l'un des deux paramètres suivantes :

- *IssuerAndSerialNumber* : identifie un certificat, et de ce fait une entité et une clé publique. Cet identificateur est établi grâce au nom distingué de l'émetteur du certificat et au numéro de série spécifique du certificat.
- *subjectKeyIdentifier* : identifie le certificat du destinataire par la valeur *subjectKeyIdentifier* de l'extension X.509.

Le paramètre *version* est la syntaxe correspondante au numéro de version. Si le champ *SignerIdentifier* correspond à *issuerAndSerialNumber*, alors la version doit être 1. Si le champ *SignerIdentifier* correspond à *subjectKeyIdentifier*, alors la version doit être 3.

[82] Cryptographic Message Syntax

Liste des publications

Internet-Draft (contributions à l'IETF)

[Bad0] P. Eronen (editeur), H. Tschofenig (editeur), M. Badra (auteur), O. Cherkaoui (auteur), I. Hajjeh (auteur), A. Serhrouchni (auteur), "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", IETF Internet Draft, draft-ietf-tls-psk-02.txt, (Proposed Standard) September 2004.

[Bad1] M., Badra, O., Cherkaoui, I., Hajjeh, and A., Serhrouchni, "Pre-Shared-Key Key Exchange Methods for TLS", draft-badra-tls-key-exchange-00.txt, Internet Engineering Task Force Draft, July 2004.

[Bad2] M., Badra, A., Serhrouchni, and P., Urien, "TLS Express", draft-badra-tls-express-00.txt, Internet Engineering Task Force Draft, July 2004.

[Bad3] M., Badra, and P., Urien, "EAP-Double-TLS Authentication Protocol", draft-badra-eap-double-tls-00.txt, Internet Engineering Task Force Draft, May 2004.

Revue

[Bad4] M., Badra, A., Serhrouchni and P., Urien, "A Lightweight Identity Authentication Protocol for Wireless Networks", special issue of Computer Communications on topics in wireless/mobile security and performance. Volume 27/17. Pages: 1738-1745.

[Bad5] M. Badra, I. Hajjeh, and A. Serhrouchni, "Using Pre-Shared-Keys within TLS for Wireless Authentication", Paper submitted to IEEE Transactions on Wireless Communications.

Proceeding

[Bad6] M., Badra, and P., Urien, "Enhancing WLAN Security by Introducing EAP-TLS Smartcards", IADIS WWW/Internet 2004 Conference, Madrid, Spain, 6-9 October 2004. to appear

[Bad7] P., Urien, M., Badra, and M., Dandjinou, "EAP-TLS Smartcards, from Dream to Reality", the fourth IEEE workshop on Applications and Services in Wireless Networks, IEEE ASWN 2004, Boston, Massachusetts, USA.

[Bad8] M., Badra, and P., Urien, "Introducing SmartCards to Remote Authenticate Passwords using Public Key Encryption", 2004 IEEE Sarnoff Symposium on Advances in Wired and Wireless Communications 2004, NJ, USA. Pages: 123-126.

[Bad9] M., Badra, and P., Urien, "Toward SSL integration in SIM SmartCards", IEEE Wireless Communications and Networking Conference, IEEE WCNC 2004, Atlanta-Georgia USA. ISBN 0-7803-8345-1 (CD-ROM).

[Bad10] M., Badra, Ph., Godlewski, A., Serhrouchni and P., Urien, "Une méthode comparative pour établir des solutions de sécurité basées sur SSL-WTLS", IEEE SETIT 2004, International Conference Sciences of Electronic, Technology of Information and Telecommunications, Sousse, Tunis.

[Bad11] M., Badra, and A., Serhrouchni, "Light TLS: Extended Handshake for Wireless Communications", Fifth International Conference on Mobile and Wireless Communications Networks, MWCN 2003, Singapore. pp. 63-67.

[Bad12] M., Badra, and A., Serhrouchni, "A New Secure Session Exchange Key Protocol For Wireless Communications", the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, IEEE PIMRC2003, Chine. pp. 2765 – 2769.

[Bad13] M., Badra, P., Godlewski, and A., Serhrouchni, "Extended Wireless Transport Layer Security", the third IEEE workshop on Applications and Services in Wireless Networks, IEEE ASWN 2003, Switzerland. pp. 35-45.

[Bad14] M., Badra and P., Godlewski. "SSL et WTLS : Etude Analytique et Comparative", 2nd Conference on Security and Network Architectures, SAR02, Marrakech (Maroc), July 2002.

Liste des acronymes

3GPP	3rd Generation Partnership Project
APDU	Application Protocol Data Unit
AES	Advanced Encryption Standard
AH	Authentication Header
BSS	Basic Set Service
CA	Certificate Authority
CHAP	Challenge-Handshake Authentication Protocol
CMS	Cryptographic Message Syntax
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CORBA	Common Object Request Broker Architecture
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DH	Diffie-Hellman
DoS	Denial of Service
EAP	Extensible Authentication Protocol
EAPoL	EAP over LAN
ECC	Elliptic Curve Cryptographic
EFF	Electronic Frontier Foundation
EPIS	Embarquer un Protocole Internet Sécurisé
ESP	Encapsulating Security Payload
ESS	Extended Service Set
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
GSM	Global System for Mobile
GTK	Group Transient Key
HIPERLAN	HIgh PErformance Radio LAN
HMAC-MD5	Hash-based Message Authentication Mode-Message Digest 5
HTML	HyperText Mark-up Protocol
HTTP	HyperText Transport Protocol
IBSS	Independent BSS
IEEE	Institute for Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	IP Security
ISO	International Standard Organization
ISP	Internet Service Provider
IV	Initialization Vector
JFK	Just Fast Keying
KCK	EAPoL-Key Confirmation Key
KEK	EAPoL-Key Encryption Key
L2TP	Layer Two Tunneling Protocol
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LLC	Logical Link Control
MAC	Message Authentication Code
MAC	Medium Access Control
MAN	Metropolitan Area Network

MD5	Message Digest
MIC	Message Integrity Check
MitM	Man in the Middle
MPPE	Microsoft Point-to-Point Encryption
NAS	Network Access Servers
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
OTA	Over The Air
PDA	Personal Digital Assistant
PEAP	Protected EAP
PFS	Perfect Forward Secrecy
PIN	Personal Identification Number
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
PKCS#7	Public Key Cryptography Standard #7
PMK	Pairwise Master Key
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
PRF	Pseudo Random Function
PSK	Pre Shared Key
PTK	Pairwise Transient Key
RADIUS	Remote Authentication dial-In User Service
RC4	Rate Code 4
RSA	Rivest Shamir Adleman
RSN	Robust Security Network
SDU	Service Data Unit
SHA	Secure Hash Standard
SIM	Subscriber Identity Module
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SRP	Secure Remote Password
SSID	Service Set IDentity
SSO	Single Sign On
T-SDU	Transport SDU
TCP	Transport Control Protocol
TK	Temporal Key
TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
TTL	Time To Live
TTLS	Tunnel TLS
UMTS	Universal Mobile Telecommunication Service
URL	Universal Resource Locator
USIM	Universal SIM
VPN	Virtual Private Network
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WBML	Wireless Binary Mark-up Language
WDP	Wireless Datagram Protocol
WEP	Wireless Encryption Privacy
Wi-Fi	Wireless Fidelity
WIM	WAP Identity Module

WML	Wireless Mark-up Language
WPA	Wi-Fi Protected Access
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTP	Wireless Transport Protocol
WTLS	Wireless TLS
WWW	World Wide Web
XDR	eXternal Data Representation