MASTER EN SCIENCES INFORMATIQUES

OpenSST: format de message et protocole d'échange sécurisé de données

Hallez, Régis

Award date:
2003

Facultés Universitaires Notre Dame de la Paix
Institut d'informatique
Rue Grandgagnage, 21
B-5000 Namur

OpenSST : Format de message et protocole d'échange sécurisé de données

par
Régis Hallez

Sous la direction du **Professeur Jean Ramaekers**

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique
Année académique 2002-2003

Résumé

Dans des environnements tels que l'e-business, l'e-banking, l'e-gouvernement,... il devient de plus en plus essentiel de sécuriser les systèmes de transactions électroniques pour éviter les fraudes et les vols d'informations dont les conséquences peuvent se révéler désastreuses. Bien qu'il existe actuellement de nombreux protocoles sécurisés d'échange de données, ces derniers peuvent avoir des failles et il manque cruellement d'un standard qui puisse s'adapter dans tous les cas d'application possibles (messagerie électronique, commerce électronique, systèmes embarqués, systèmes offline,...). C'est le but poursuivi par Aubay, Conostix et le Centre de Recherche Public Henri Tudor (Luxembourg) qui ont décidé de développer un nouveau format de message, OpenSST (Open, Simple, Secure, Transactions). Ce mémoire réintroduit les concepts de sécurité élémentaires avant de passer en revue brièvement les différents protocoles d'échange les plus répandus actuellement. Ensuite, le format de message et protocole OpenSST sont présentés et expliqués de manière plus approfondie.

Mots-clés : sécurité, OpenSST, format de message, transactions électroniques, protocoles d'échange de données sécurisés

In the context of e-business, e-banking or e-government, it is more and more important to secure our systems. Security is necessary to avoid fraud and theft of information as these can have disastrous consequences. Secure data exchange protocols exist but they can suffer from security holes. Furthermore, there is no standard that can cope with every single case (email, e-payment, embedded systems, offline systems, ...). That is why Aubay, Consostix and the Centre de Recherche Public Henri Tudor (Luxembourg) decided to develop a new message format, OpenSST (Open, Simple, Secure, Transactions). This dissertation introduces the different basic security concepts and some of the most commonly used secure data exchange protocols. It then goes on to introduce and thoroughly explain OpenSST (message format and protocol).

Key-words : security, OpenSST, message format, electronic transactions, secure data exchange protocols

Remerciements

Je remercie Monsieur Jean Ramaekers, mon promoteur, pour son aide, ses conseils et son suivi tout au long de ces deux années.

Je tiens ensuite à remercier Christophe Feltus, mon maître de stage, et les membres du CITI de Esch-sur-Alzette, Corine Barella, Jérôme Larré, Jean Larock, Bruno Cornette, Jean-Charles Bernacconi et les autres, pour leur amabilité et leurs conseils tout au long de mon stage à Esch.

Pour leurs conseils, leurs connaissances, leur soutien et leur sympathie, je voudrais remercier Alexandre Dulaunoy de Conostix et Sébastien Stormacq de Aubay ainsi que Yves Depril de Conostix.

Je tiens finalement à remercier ma famille, mes amis et particulièrement ma petite amie, Laura, qui m'ont tous soutenu, encouragé et remonté le moral dans les moments difficiles.

Régis Hallez.

Table des matières et autres tables

1 Table des matières

2 Table des illustrations

Introduction

Depuis l'avènement des réseaux informatiques grand public, la façon de penser la vie quotidienne, autant pour les particuliers que pour les entreprises et autres organismes publics, a énormément évolué. On assiste actuellement à une évolution de la société qui tend de plus en plus à permettre la gestion des affaires courantes en restant assis devant son ordinateur. Le point 2 de la partie *problématique de la sécurité des transferts et transactions, et importance de la sécurité* de ce mémoire retrace cette évolution et met en avant les enjeux, impacts et obstacles de l'informatisation et de l'utilisation des réseaux

Plus particulièrement, ce mémoire a pour but de cibler les échanges qui se réalisent quotidiennement entre les entreprises, les particuliers, les banques, les organismes publics,... et la nécessité de sécuriser ces échanges pour éviter des accidents ou même des désastres. En effet, les informations échangées par l'intermédiaire de réseaux à risques tel l'Internet peuvent s'avérer être extrêmement confidentielles et le vol de ces informations peut causer des problèmes considérables. Il peut s'agir bien sûr de pertes financières mais également de pertes de confidentialité de documents nécessitant le secret. La Citibank, en 1994, a, par exemple, été victime d'un hacker qui a réussi à transférer plus de 10 millions de dollars sur des comptes situés un peu partout dans le monde. De même, une étude réalisée par le MIS Training Institute en 1997 révèle que « 13% des 148 plus grandes compagnies mondiales ont déclaré avoir subi des vols d'informations »[1]. Les points 3 et 4 de la partie suivante introduisent les concepts de transactions et de cryptographie et insistent sur la nécessité de sécuriser les échanges.

Dans le cadre de la sécurisation des échanges de données, de nombreux formats de messages et protocoles sécurisés d'échange de données ont été développé ces dernières années. Les plus connus et plus répandus sont repris et discutés dans la partie *Protocoles Existants*.

Cependant, le Centre de Recherche Henri Tudor (Luxembourg), en partenariat avec Aubay et Conostix, 2 entreprises luxembourgeoises, a décidé, au début de l'année 2002, de développer un nouveau protocole, OpenSST, pour répondre aux besoins du marché luxembourgeois. La présentation de ce protocole en détail et ma participation à ce projet se trouvent reprises dans la partie *OpenSST*.

Mon rôle dans le projet OpenSST a été tout d'abord d'établir une étude comparative du protocole OpenSST par rapport aux différents protocoles

[1] Voir http://www.securite-informatique.com/menaces.htm .

déjà existants. Ensuite, j'ai dû délivrer des documents de formalisation des différentes étapes du projet. Pour finir, après avoir mis en évidence certains cas d'utilisation fortement demandé par le marché luxembourgeois, ma dernière tâche a été de réaliser quelques fiches techniques décrivant la solution apportée par OpenSST ces différents cas d'application.

Le point 5 de cette dernière partie envisage l'avenir d'OpenSST et les prochaines étapes de son développement.

Problématique de la sécurité des transferts et transactions, et importance de la sécurité

1 Introduction

Ce chapitre a pour but de familiariser le lecteur aux différents concepts relatifs aux transactions électroniques et à la sécurisation de celles-ci. Le point 2 retrace l'évolution des mentalités vers un monde dans lequel l'utilisation de l'informatique et des réseaux a acquis un rôle considérable dans les relations entre les différents acteurs de la société contemporaine, c'est-à-dire les entreprises, les banques, le gouvernement, les particuliers,... Les enjeux et impacts de l'informatique et des réseaux y sont également repris, de même que les obstacles rencontrés. Le point 3 consiste en une définition des transactions électroniques. Enfin, le point 4 insiste sur le besoin de sécurisation des transactions électroniques relatives à des domaines sensibles et définit les concepts fondamentaux de la sécurité au sens informatique du terme.

2 Développement de l'informatique et des réseaux

Avant de rentrer dans le vif du sujet et de parler des systèmes de transferts sécurisés de données, commençons par une brève introduction sur l'importance croissante de l'informatique et des réseaux dans notre société.

2.1 Historique

L'utilisation de l'informatique et des réseaux n'est pas nouvelle. En effet, depuis plus de 15 ans, les particuliers pouvaient déjà utiliser les réseaux télématiques grand public (ex : Minitel, vidéotex) pour effectuer des achats divers, réserver des places pour des événements ou réserver leurs vacances. Au niveau professionnel, il existait également l'EDI qui permettait aux entreprises de s'échanger de l'information de manière entièrement automatisée (EDI = « *échange automatisé de messages normalisés et agréés entre applications informatiques, à l'aide d'un moyen téléinformatique.* »[2]). L'EDI avait cependant de nombreux problèmes tels les investissements lourds, l'application de protocoles et de modes de présentation peu accessibles aux non experts, et une mise en œuvre complexe[3].

[2] Professeur Etienne Montero, *syllabus d'informatique et droit*, année académique 2001-2002.
[3] Voir http://www.telepost.fr/edipost/fr_ediqu.htm .

Le développement de l'Internet a permis de résoudre ces problèmes. En effet, l'Internet ne nécessite que peu d'investissements spécifiques. De plus son coût d'utilisation est modeste et il est accessible au grand public.

D'autres différences majeures entre les réseaux télématiques et l'Internet :

Réseaux télématiques et EDI	Internet
Réseau fermé	Réseau ouvert
Protocoles propriétaires	Protocoles non propriétaires (TCP/IP)
Point de contrôle unique	Pas de point de contrôle unique, décentralisé

Figure 1 - Différences entre Internet et les réseaux télématiques et EDI

2.2 Situation actuelle

D'après une enquête effectuée en janvier 2001 par Nua Internet, il y aurait environ 400 millions de personnes qui seraient connectées à l'Internet. Cette étude révèle également que le profil de l'utilisateur a changé. On passe d'une population universitaire à une population résidentielle avec un haut pouvoir d'achat.[4]

Dans le monde professionnel, les PME peuvent maintenant se permettre de participer aux échanges online d'information et ne sont plus exclus du fait d'investissements trop lourds.

Le nombre d'applications online se multiplient et il est impossible de prévoir jusqu'à quel point l'informatisation s'étendra. On retrouve aujourd'hui de nombreuses tâches quotidiennes qu'il est possible de réaliser grâce à l'utilisation d'un ordinateur. Plus besoin de se rendre à la banque pour gérer ses comptes, il suffit de posséder le logiciel d'e-banking de sa banque. Faire ses courses ? Quelques clics suffisent pour que la commande soit passée et il ne reste plus qu'à attendre la livraison à domicile. Remplir sa déclaration d'impôts sera bientôt également possible via un ordinateur. Une entreprise veut passer des commandes auprès de ses fournisseurs ? Rien de plus simple grâce au développement de l'e-business,... La liste est encore longue et continue de s'allonger.

Nous pouvons donner quelques chiffres relatifs à l'e-commerce. Pour ce qui est du B2C e-commerce (Business to Consumer e-commerce : vendre des biens et services aux consommateurs), aux USA, les dépenses des

[4] Voir D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.

consommateurs ont été de $7,7 milliard en 1998, $17,3 milliard en 1999 et $28 milliard en 2000. Les chiffres pour le B2B e-commerce (Business to Business e-commerce : transactions entre les entreprises) seraient de $226 milliard en 2000 et les études projettent d'atteindre les $2700 milliard en 2004.[5]

2.3 Enjeux et impacts

Le point précédent nous a permis de nous rendre compte de l'importance prise par l'informatique et les réseaux dans la société actuelle. Passons maintenant en revue les enjeux et impacts pour les différents acteurs de cette société.

> *« Les NTIC [...] modifient profondément les modes de consommation, les processus de production et l'ensemble des relations de l'entreprise avec ses partenaires. Un nouveau bond technologique est en train de se produire. »* (Dominique Strauss-Kahn)

Côté Professionnels

✓ Simplification des contacts et des échanges d'informations interentreprises et au sein même de l'entreprise.

✓ Bouleversement des règles qui gouvernent l'environnement concurrentiel grâce à l'e-commerce.

- Transparence des marchés par une meilleure information sur l'offre.
- Pression sur les prix.
- Personnalisation de l'offre. Il faut chercher d'autres moyens de se distinguer des concurrents (interactivité, one-to-one marketing, profilage des utilisateurs).

✓ Simplification des procédures qui coûtent traditionnellement très cher aux entreprises par la mise en réseau interne (entre les services opérationnels et les services achats) et externe (avec les fournisseurs).

Côté Etat

✓ Simplification de la gestion des documents et de l'administration.

---

[5] Voir D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.

Côté Particuliers

✓ E-commerce :
- Transfert de pouvoir vers les clients : accès à la totalité de l'offre disponible dans le monde.
- Accroissement de l'économie grâce au changement des règles de concurrence. (baisse des prix, transparence)

✓ E-gouvernement : gain de temps et rapprochement des particuliers et de l'Etat.

✓ E-banking : gain de temps et gestion quotidienne facilitée.

2.4 Obstacles

Il y a bien sûr des obstacles à surmonter pour pouvoir bénéficier au mieux des enjeux de l'informatisation. En voici quelques exemples :

✓ Sécurité des échanges électroniques (authentification, non-répudiation, intégrité, incertitude juridique).

✓ Apprentissage nécessaire.

✓ Domination de la langue anglaise.

✓ Coûts des équipements de base.

C'est autour du premier obstacle cité que s'oriente ce mémoire.

3 Définition d'une transaction

Prenons la définition proposée par l'Agence Wallonne des Télécommunications, l'AWT:

« une transaction électronique est un ensemble cohérent d'échanges d'informations relatifs à une même idée ou à un même acte, entre deux ordinateurs au travers d'un réseau informatique »[6]

Le schéma ci-dessous représente l'environnement d'une transaction électronique.

[6] Voir les fiches techniques de l'AWT, http://www.awt.be/cgi/fic/fic_menu.asp .

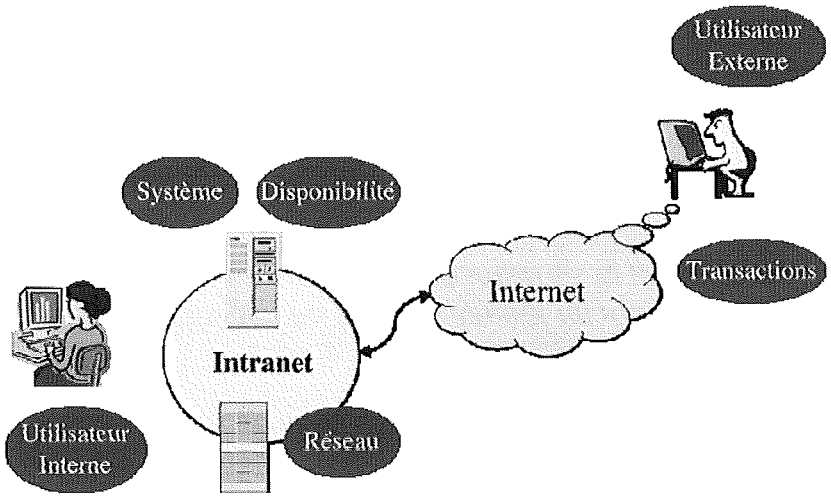

Figure 2 - Transaction électronique

Des transactions électroniques interviennent dans variétés d'applications, telles que l'e-business, l'e-gouvernement, l'e-banking, les systèmes informationnels médicaux ou encore les systèmes embarqués.

Exemple : e-gouvernement

- Transmission de documents officiels.
  - Appels d'offres / cahiers des charges
  - Formulaires
  - Offres
- Transmission de données financières.
  - Déclaration TVA
  - Déclaration d'impôts
  - Matrice cadastrale

4 Sécurisation des transactions

Les transactions électroniques interviennent dans des domaines aussi nombreux que variés. Certaines applications nécessitant des transactions électroniques peuvent s'avérer très sensibles. Dans le point précédent, des exemples de transactions relatives à l'e-gouvernement sont cités. On comprend que Monsieur X remplissant sa déclaration d'impôt en ligne désire que les informations qu'il communique soient protégées et ne tombent pas dans les mains d'un quelconque pirate informatique qui pourrait utiliser ces informations à ses dépends. Il en va de même pour de nombreuses applications. Le concept de sécurisation des transactions découle directement de ce besoin de protection.

Avant d'expliquer les propriétés fondamentales nécessaires à la sécurisation des transactions électroniques, définissons la sécurité informatique et les dangers potentiels.

De nouveau, prenons comme base la définition de la sécurité informatique donnée par l'AWT:

« La sécurité est la situation dans laquelle un système informatique, connecté ou non à un réseau externe de télécommunications, est protégé des dangers internes ou externes »[7]

L'AWT donne aussi un schéma représentant les dangers potentiels:

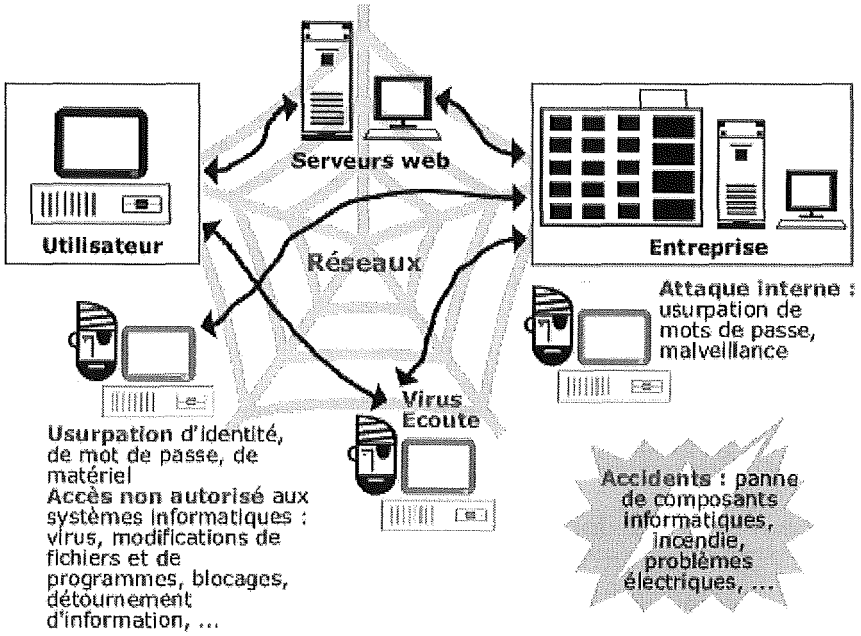

Figure 3 - Dangers potentiels

Les 4 propriétés fondamentales nécessaires à la sécurisation des transactions électroniques sont les suivantes :

- La confidentialité.
- L'authentification.
- L'intégrité.
- La non-répudiation.

Ces propriétés, qui seront définies plus bas dans cette partie, peuvent être assurées par le recours à divers procédés et méthodes de cryptographie existants. Ceux-ci sont les suivants :

[7] Idem que note 6 .

La cryptographie symétrique ou à clé secrète.

Les algorithmes de chiffrement symétrique présupposent l'existence d'une clé partagée, et tenue secrète, entre les deux parties d'une transaction. Les données sont chiffrées à l'origine à l'aide de cette clé et déchiffrées à la destination au moyen de cette même clé. Quiconque ne possède pas la clé utilisée est incapable de lire les données en clair.

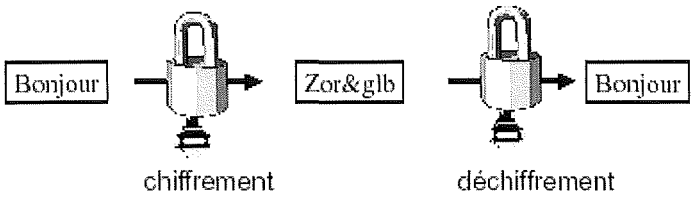

Figure 4 - Chiffrement symétrique

Des exemples de tels algorithmes sont DES, AES, RC2, IDEA,...

DES

Algorithme de chiffrement basé sur la technique de cryptographie symétrique publié par IBM et adopté par le département de la défense américaine en 1977. DES repose sur une clé symétrique de 56 bits et effectue un chiffrement par blocs de 64 bits.

Du fait de sa vulnérabilité aux attaques de type force brute, l'usage de DES décline au profit d'autres algorithmes, notamment 3-DES et AES (Rijndael).

AES

Algorithme de chiffrement basé sur la cryptographie symétrique. AES est le remplaçant de DES. Face aux vulnérabilités du DES vis-à-vis des attaques par force brute, l'institut NIST a lancé en janvier 1997 un appel d'offre pour le remplacement du DES par un nouvel algorithme de chiffrement par blocs de 128 bits, supportant des clés de chiffrement de 128, 192 et 256 bits au minimum. Parmi les 5 algorithmes retenus (MARS, RC6, Rijndael, Serpent, Twofish), l'algorithme belge Rijndael a été choisi.

RC2

Algorithme de chiffrement par blocs (64 bits), à clé de longueur variable. Il est créé par Ron Rivest pour RSA Data Security. Il est plus rapide que DES et est conçu comme une simplification de DES en vue de le

remplacer. Il peut être plus ou moins sûr que DES en fonction de la longueur de clé utilisée. RC2 est confidentiel et propriétaire.

IDEA

Algorithme de chiffrement symétrique né en Europe dans les années 1990. IDEA chiffre par blocs de 64 bits utilisant des clés de 128 bits, basé sur des opérations de mélange dans divers groupes algébriques. Considéré comme l'un des algorithmes les plus forts. Malgré cela, il ne fait pas l'unanimité chez les spécialistes en cryptographie et son adoption par le monde Internet reste limitée.

La cryptographie asymétrique ou à clé publique.

Les algorithmes de chiffrement asymétrique repose sur l'utilisation d'une paire de clés complémentaires. L'une rendue publique et l'autre, privée, gardée secrète et connue uniquement de son propriétaire. Chaque clé permet de chiffrer le message que seule l'autre clé peut déchiffrer. Il n'est pas possible de chiffrer et déchiffrer à l'aide de la même clé car le processus est irréversible. Il est également impossible de dériver une clé à partir de l'autre.

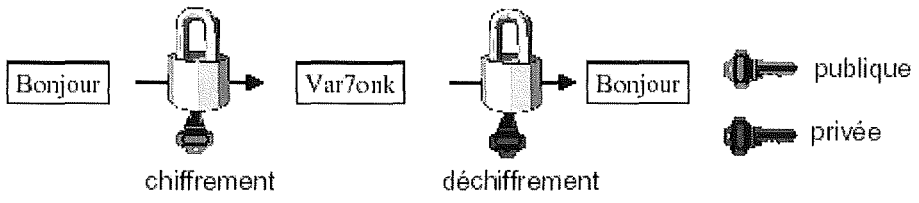

Figure 5 - Chiffrement asymétrique

Exemples : RSA, DSS,...

RSA

Algorithme créé par Rivest, Shamir et Adleman (RSA) en 1977, et basé sur la cryptographie asymétrique, RSA est l'algorithme le plus utilisé dans le monde et offre les services de sécurité essentiels tels que l'authentification, la confidentialité, l'intégrité et la signature. RSA connaît un essor important avec le développement des infrastructures de gestion de clés (PKI). Anciennement propriété de la société RSA Security, mais ouvert au domaine public depuis la fin 2000, l'algorithme RSA repose sur la difficulté de factoriser un grand nombre suivant ses facteurs premiers.

Les clés asymétriques utilisées dans le cadre de RSA sont classiquement de taille 512, 1024, 2048, 4096 bits. On considère à l'heure

actuelle qu'une clé de 512 bits à une durée de résistance trop faible pour être utilisée.

DSS

Algorithme de signature électronique (DSA) développé par la *U.S. National Security Agency* (NSA) pour l'authentification de documents électroniques. DSS a été mis en avant par la *National Institute of Standards and Technology* (NIST) en 1994 et est devenu le standard pour l'authentification de documents électroniques du gouvernement des Etats-Unis.

Les fonctions de hashage.

Le procédé de hashage consiste à convertir un texte original de longueur quelconque en un message de longueur fixe (en général de longueur inférieure). Le but est d'utiliser le message hashé comme empreinte digitale du message original qu'il identifie de manière univoque. Le destinataire peut alors être raisonnablement certain de l'intégrité du message.

Les propriétés d'un bon hash sont :

- Il n'y a pas moyen de retrouver le message original à partir du hash.
- Deux messages ne différant que d'un seul bit ont un hash résultant complètement différent.
- Il est impossible, à partir d'un hash, de déduire le hash que donnerait telle ou telle modification du message.

Exemples : SHA-1, MD5, RIPEMD-160,...

SHA-1

SHA (Secure Hash Algorithm) est l'algorithme spécifié dans le Secure Hash Standard (SHS) et a été développé par NIST et publié comme un standard FIPS. SHA-1 est une révision de SHA et a été publiée en 1994. Sa conception est très similaire à la famille d'algorithmes développés par Rivest (MD2, MD4, MD5). SHA-1 prend un message de moins de 2^{64} bits de longueur et produit un output de 160 bits. L'algorithme est légèrement plus lent que MD5, mais, le message en sortie étant plus grand, il est plus sûr contre les attaques. Il est très populaire et largement utilisé.

MD2, MD4 et MD5

Algorithmes de hachage développés par Rivest. Ils sont destinés aux applications nécessitant une signature électronique pour lesquelles des messages de taille importante doivent être "compressés" d'une façon sécurisée avant d'être signé avec la clé privée. Ces trois algorithmes prennent en entrée un message d'une longueur arbitraire et produisent en sortie un "message digest" de 128 bits. Si la structure de ces algorithmes est globalement similaire, MD2 est différent de MD4 et MD5 en cela qu'il a été optimisé pour des machines 8 bits alors que les deux autres sont prévus pour des machines 32 bits.

RIPEMD-160

Fonction de hachage de 160 bits, développée par Hans Dobbertin, Antoon Bosselaers et Bart Preneel dans le cadre du projet européen RIPE (RACE Integrity Primitives Evaluation, 1988-1992). Il est prévu pour être utilisé comme remplacement sécurisé des fonctions MD4 et MD5 (128 bits) et RIPEMD.

La signature électronique.

La signature électronique de données permet à un destinataire de s'assurer de l'identité de l'expéditeur. Il est nécessaire à celui qui souhaite signer électroniquement ses données de disposer d'une paire de clés (la signature électronique se base sur la cryptographie asymétrique).

Pour signer électroniquement un message, l'expéditeur utilise sa clé privée, et, à la réception, le destinataire utilise la clé publique de l'expéditeur. Ce procédé lui permet de vérifier la validité de la signature car seul le détenteur de la clé privée a pu créer la signature. Il n'est pas nécessaire de chiffrer tout le message pour le signer, il suffit de chiffrer son hash.

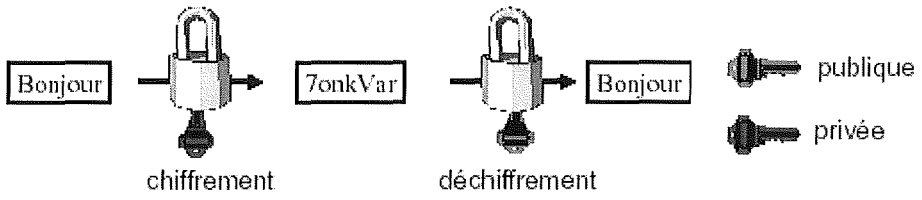

Figure 6 - Signature électronique

4.1 Confidentialité

Le glossaire de sécurité de e-SecurIT définit la confidentialité en ces termes :

« Qualité de l'information qui n'est pas disponible ou accessible par des individus, entités ou processus non autorisés. »[8]

La confidentialité au sens informatique du terme est le fait de préserver le contenu d'un message des personnes non autorisées. La confidentialité des données se base sur les algorithmes de chiffrement symétrique ou asymétrique (Cfr. supra).

Dans le cas de transactions, l'émetteur d'un message peut souhaiter la confidentialité des données qu'il transmet au destinataire. En effet, Internet est un réseau ouvert et à risques et donc peu sûr. Par exemple, un consommateur utilisant un système de paiement online veut pouvoir rendre confidentielles les données relatives à sa carte de crédit pour éviter qu'un tiers mal intentionné puisse y accéder et réutiliser ces données pour effectuer des achats.

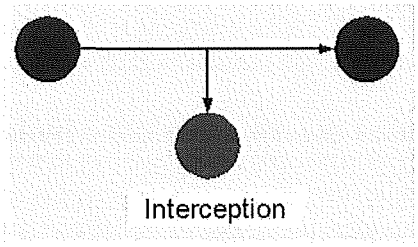

Figure 7 - Confidentialité

4.2 Authentification

Toujours selon e-SecurIT, l'authentification est un

« Service de sécurité dont l'objectif est de valider l'identité d'une entité (utilisateur ou équipement). Il existe classiquement trois méthodes d'authentification permettant de prouver l'identité d'une entité :

- Authentification basée sur la connaissance d'un secret (ex. : mot de passe).
- Authentification basée sur la possession d'un objet (ex. : carte à puce, jeton).
- Authentification basée sur la biométrie. »[9]

Ce service de sécurité est assuré au moyen de la signature électronique (Cfr. supra).

---

[8] voir glossaire de e-SecurIT, http://securit.free.fr/glossaire.htm .
[9] Idem que note 8.

La signature électronique est une contrainte souvent exigée pour les transactions dans l'environnement de l'e-business. Par exemple, son utilité pour le cas du paiement électronique online est évidente. Un serveur dans un système d'achat online souhaite être certain de l'identité du client pour assurer la non-répudiation de la transaction, c'est-à-dire que le client ne puisse pas nier avoir réalisé la transaction.

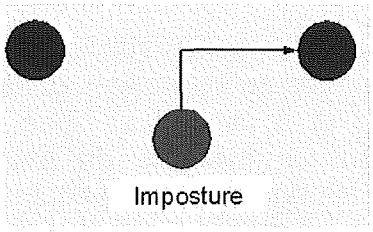

Figure 8 - Authentification

4.3 Intégrité

« Qualité de données qui n'ont pas été altérées ou détruites de manière frauduleuse. »[10]

L'intégrité d'un message assure que celui-ci n'a pas été altéré lors de son passage sur le réseau.

C'est le procédé de hachage qui permet de garantir l'intégrité d'un message. A l'origine, le message est couplé avec son hash et transmis au destinataire. Celui-ci recalcule le hash du message et le compare avec le hash qui lui a été envoyé. Si les deux sont identiques, le destinataire peut être certain de l'intégrité des données.

Dans le cas du paiement électronique, lorsqu'un utilisateur envoie un message avec le numéro de sa carte et le montant à débiter, il veut s'assurer que ce montant ne soit pas modifié lors du passage sur le réseau.

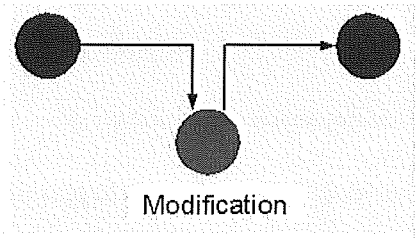

Figure 9 - Intégrité

[10] Idem que note 8.

4.4 Non-répudiation

« Service de sécurité dont l'objectif est de générer, récolter, maintenir, rendre disponible et valider l'évidence (information utilisée pour établir une preuve) concernant un événement ou une action revendiquée afin de résoudre les possibles disputes sur l'occurrence ou non de l'événement ou de l'action. »[11]

La non-répudiation d'un message est le fait que ce message est lié avec certitude à celui qui l'envoie. Il s'agit d'éviter que l'émetteur d'un message puisse nier l'avoir envoyé. Cette propriété se base sur la signature électronique et sur les fonctions de hashage. En effet, si l'émetteur signe le hash du message, il lie son identité avec le message et ne peut plus, par la suite, nier avoir envoyé ce message.

L'utilité dans le cas du paiement électronique est indéniable car la non-répudiation assure au commerçant que le consommateur ne pourra pas nier avoir émis la transaction.

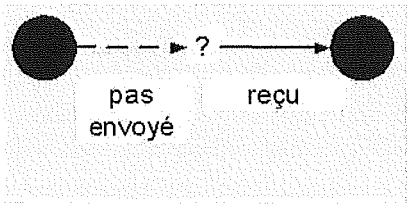

Figure 10 - Non-répudiation

5 Références

- Professeur Etienne Montero, *syllabus d'informatique et droit*, année académique 2001-2002.
- Télépost, *Qu'est-ce que l'EDI ?*, http://www.telepost.fr/edipost/fr_ediqu.htm
- D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.
- Ph. Oechslin, Quelques notions de cryptographie, http://lasecwww.epfl.ch/securitereseaux/files/sl_07.pdf .
- Aubay, Conostix, CRP Henri Tudor, *OpenSST Open Simple Secure Transactions*, SI Expo 2002

[11] Idem que note 8.

- Fiches techniques de l'AWT, http://www.awt.be .
- Glossaire Whatis.com, http://whatis.techtarget.com .
- Glossaire de cryptographie de X5, http://www.x5.net/faqs/crypto .
- Glossaire de Hoo-la, http://www.hoo-la.com/avenirmicro/glossaire_a.php .
- Glossaire de e-SecurIT, http://securit.free.fr/glossaire.htm .
- Glossaire de cryptographie, http://security.rbaumann.net/convenc.php?sel=2 .
- Norbert Vidon, Jean-François Denis, Yves De Pril, Christophe Feltus, *OpenSST, Open, Simple, Secure Transactions*, GT Spiral, Décembre 2002.

Protocoles existants

1 Introduction

Le chapitre précédent a permis de faire ressortir le besoin en sécurisation des transactions et a défini les exigences fondamentales en matière de sécurité informatique. Ce chapitre a pour but de passer en revue de manière non-exhaustive et superficielle les principaux formats de messages et protocoles disponibles permettant la sécurisation des transactions électroniques. Mais avant tout, il convient d'éclairer le lecteur sur le concept de protocole.

En informatique, un protocole est un ensemble de règles à suivre par deux parties lorsqu'elles communiquent entre elles. Un protocole définit une suite de messages (requêtes-réponses) que les parties doivent respecter afin de se comprendre et de permettre l'échange d'information.

Le site Web www.commentcamarche.net donne la définition suivante:

« Un protocole est une méthode standard qui permet la communication entre des processus (s'exécutant éventuellement sur différentes machines), c'est-à-dire un ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau. Il en existe plusieurs selon ce que l'on attend de la communication. Certains protocoles seront par exemple spécialisés dans l'échange de fichiers (le FTP), d'autres pourront servir à gérer simplement l'état de la transmission et des erreurs (c'est le cas du protocole ICMP), ... »[12]

Les points 2 à 9 décrivent 8 protocoles ou formats de messages actuellement disponibles pour sécuriser des transactions électroniques. Le schéma suivant positionne ces protocoles dans le modèle OSI relativement aux différentes couches réseaux.

[12] Voir le site de vulgarisation informatique http://www.commentcamarche.net .

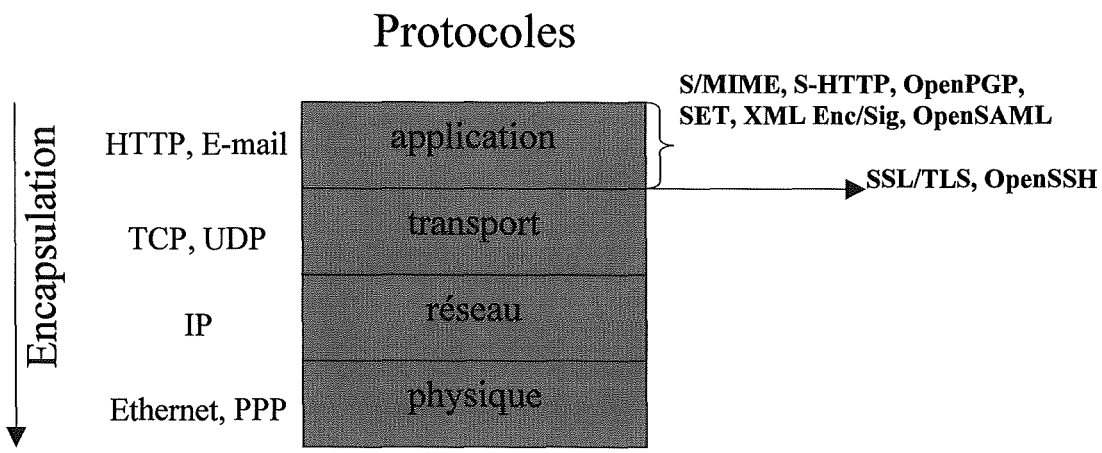

Figure 11 - Situation des protocoles dans le modèle OSI

2 S/MIME

S/MIME est un protocole de chiffrement de messages électroniques reposant sur la technique de cryptographie asymétrique RSA. C'est un standard proposé par Deming Software et RSA Data Security pour chiffrer et/ou authentifier des données MIME. S/MIME est une extension sécurisée de MIME (S/MIME définit un format pour les données MIME) intégrant des services d'authentification par signature électronique (MD5, SHA-1) et confidentialité par chiffrement (RSA, RC2, DES). S/MIME règle également les problèmes opérationnels supplémentaires comme les certificats ANSI X. 509 et le transport via Internet. Le format des messages S/MIME est défini dans le standard PKCS #7 (Public Key Cryptography System). RSA a proposé S/MIME comme standard auprès de l'Internet Engineering Task Force (l'IETF est une communauté internationale de développeurs spécialisés dans les réseaux, de vendeurs et de chercheurs qui, organisés en groupes autour de certains sujets, se concentrent sur l'évolution de l'architecture de l'Internet[13]). S/MIME est inclus dans les dernières versions des browsers Web de Microsoft et de Netscape et est également utilisé dans des produits d'autres vendeurs.[14]

Forces

- Sécurité forte.
- Popularité.
- Gratuité.

[13] Voir http://www.ietf.org/overview.html .
[14] Voir glossaires http://whatis.techtarget.com, http://www.x5.net/faqs/crypto , http://www.geocities.com/openpgp/glossaire.htm , http://searchsecurity.techtarget.com .

- Pérennité.

Faiblesses

- Limitation à la messagerie électronique.
- Coût des PKI (Public Key Infrastructure).

3 OpenPGP

OpenPGP est dérivé de "Pretty Good Privacy" ou PGP, un logiciel de cryptographie forte écrit en 1991-1992 par Philip Zimmermann. PGP est particulièrement bien adapté à l'utilisation sur Internet et est très sûr.

Le standard OpenPGP est libre et développé par tous les utilisateurs autour du groupe de travail OpenPGP de l'IETF. Le RFC 2440 en décrit le format de message.

OpenPGP utilise divers algorithmes comme IDEA, RSA, DSA, MD5, SHA- 1 pour le chiffrement, l'authentification, l'intégrité des messages et la gestion de clés[15].

Forces

- Sécurité forte.
- Performance.
- Popularité.
- Gratuité.
- Pérennité.

Faiblesses

- Non-évolutivité du format de message (binaire).
- Incompatibilité des versions successives.

4 SSL/TLS

SSL est un protocole développé par Netscape en 1994 pour fournir sécurité et confidentialité sur Internet. SSL supporte l'authentification du serveur et assure la sécurité et l'intégrité du canal de transmission. Il opère au niveau de la couche de transport, entre TCP et HTTP, et est donc indépendant des applications. SSL chiffre complètement le canal de

[15] Idem que note 14 et OpenPGP Alliance Website, http://www.openpgp.org et RFC 2440 sur OpenPGP, http://www.ietf.org/rfc/rfc2440.txt.

communication mais ne supporte ni les signatures électroniques au niveau du message, ni la non-répudiation des messages. SSL v3 a été standardisé par l'IETF sous le nom de TLS (Transport Layer Security). SSL et TLS sont intégrés dans la plupart des browsers Web. Les deux ne sont pas interopérables entre eux mais un browser supportant SSL est capable de traiter les messages envoyés avec TLS.

C'est un protocole général de sécurisation des communications via le réseau quoiqu'il ait été conçu au départ pour permettre la sécurisation des transactions par cartes de crédit sur le Web. Les informations importantes à chiffrer sont, par exemple, les détails bancaires et le numéro de carte de crédit.

SSL est notamment utilisé pour le paiement sécurisé et est soutenu par Bank of America, Mastercard, MCI et Silicon Graphics[16].

Forces

- Confidentialité, intégrité.
- Simplicité, performance, popularité.
- Transparence au niveau des applications.

Faiblesses

- Chiffrement de la session, mais pas des messages.
- Pas de non-répudiation du message.
- Authentification souvent unilatérale et pas bilatérale.

5 S-HTTP

S-HTTP (Secure HTTP), à ne pas confondre avec HTTPS (HTTP over SSL), a été développé par E. Rescorla et A. Schiffman de EIT (Enterprise Integration Technology) pour sécuriser les connexions HTTP. Il permet d'assurer les propriétés fondamentales de la sécurité (confidentialité, authentification, intégrité et non-répudiation) en utilisant divers mécanismes de cryptographie.

S-HTTP permet d'encapsuler les messages de différentes manières. Chaque message peut être chiffré et signé électroniquement. Pour un document donné, S-HTTP est une alternative au protocole analysé précédemment, SSL. Une différence majeure est que S-HTTP permet au client d'envoyer un certificat pour s'authentifier auprès du serveur alors que dans SSL, seul le serveur pouvait être authentifié. S-HTTP est plus

[16] Idem que note 14 et RFC 2246 sur TLS v 1.0. http://www.ietf.org/rfc/rfc2246.txt .

susceptible d'être utilisé dans des situations où le serveur est une banque et requiert l'authentification du client. Cette technique est plus sure que l'utilisation d'un userid et d'un mot de passe.

Une autre propriété du format de message S-HTTP est qu'il est récursif, c'est-à-dire qu'un message S-HTTP peut encapsuler un autre message S-HTTP.

Si SSL se situe juste au-dessus de TCP, S-HTTP travaille, lui, juste au-dessus de HTTP. Les deux protocoles peuvent être utilisés par l'utilisateur d'un browser, mais celui-ci ne peut pas combiner les deux pour un document donné[17].

Forces

- Sécurité forte.
- Récursivité du format de message.

Faiblesses

- Peu ou pas d'implémentation du protocole.
- Pas de standardisation et risque d'incompatibilités.

6 OpenSSH

OpenSSH, version non-propriétaire de SSH (Secure Shell), est un protocole de communication sécurisée permettant l'accès distant à des machines Unix (en remplacement de commandes telles que rlogin, rsh et rcp qui ne sont pas sécurisées). Il est utilisé énormément par des administrateurs de réseaux Web ou autres. OpenSSH permet de pallier les faiblesses de sécurité des accès distants aux systèmes Unix (ex. : telnet, X11) en fournissant les services de sécurité .

Comme SSL/TLS, OpenSSH se situe au niveau de la couche transport et ne peut donc assurer l'authentification et la non-répudiation au niveau du message mais seulement au niveau de la session.

[17] Idem que note 14 et Adam Shostack, *An overview of SHTTP*, http://www.homeport.org/~adam/shttp.html, mai 1995 et RFC 2660 sur S-HTTP, http://www.ietf.org/rfc/rfc2660.txt .

OpenSSH repose sur la technique de cryptographie asymétrique RSA. Ce protocole utilise les algorithmes symétriques Blowfish et Triple DES pour la confidentialité des données[18].

Forces

- Transparence et facilité d'utilisation.
- Alternative à plusieurs programmes (telnet, ftp, rlogin, rsh, rcp).
- Popularité.

Faiblesses

- Sécurité de la session, mais non des messages eux-mêmes.

7 SET

SET est le produit de l'association des deux plus gros organismes mondiaux qui délivrent des cartes de crédit, c'est-à-dire Visa et MasterCard. D'autres entreprises ont également collaboré au projet telles que Netscape Coporation, Microsoft, IBM, ... Le projet a débuté en 1996 et, depuis, de nombreuses révisions (8) ont été apportées au protocoles. C'est la compagnie indépendante SETCo, formée par Visa et MasterCard en 1997, qui gère ces révisions du standard[19].

SET est un protocole qui ne s'occupe que de la partie paiement des transactions.

Pour garantir la sécurité, de nombreuses techniques et algorithmes de cryptographie sont utilisées (telles que MD5, SHA, Dual signatures, RSA). Notamment, il est fait en sorte que le marchand ne puisse nullement avoir accès aux données bancaires du client et de même, la banque du marchand ne peut voir les détails de la transaction commerciale qui se déroule.

Pour permettre l'interopérabilité, tous les messages sont définis dans un format indépendant de la machine ou de l'OS.

---

[18] Idem que note 14 et Rajan Bhaktar, *How SSH protects your remote access sessions*, http://www.cas.mcmaster.ca/~wmfarmer/SE-4C03-02/projects/student_work/bhaktar.html, 23 mars 2002.

[19] Voir D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.

SET repose sur SSL, STT (Secure Transaction Technology de Microsoft, protocole développé pour le cas spécifique du paiement électronique sur l'Internet) et sur S-HTTP.

Dans ce système, toutes les parties doivent détenir un certificat et la paire de clés correspondante. Il doit donc exister une hiérarchie d'autorités de certification dont la racine est SETCo (depuis sa création).

3 composants principaux sont nécessaires pour permettre l'utilisation du protocole SET :

- Un logiciel portefeuille chez le client.
- Le support de SET doit être installé sur le serveur du marchand.
- L'interface de paiement doit pouvoir traiter les transactions SET et pouvoir interagir avec les réseaux financiers.

Forces

- Tous les messages entre les différentes parties sont entièrement sécurisés (Confidentialité, authentification, intégrité et non-répudiation des messages).
- L'interopérabilité est prise en compte.
- Les différentes parties sont parfaitement authentifiées.

Faiblesses

- SET est décrit complètement en 3 volumes fastidieux. Cela demande donc beaucoup d'efforts aux développeurs pour produire des applications répondant aux normes de SET.
- L'interopérabilité demande également des efforts très lourds.
- Le besoin d'une hiérarchie de certification pouvant supporter le système rend le processus assez complexe. De plus, les associations de cartes de crédit qui veulent pouvoir participer au système doivent d'abord mettre en œuvre l'infrastructure nécessaire mais également passer par une procédure stricte de certification avec le niveau supérieur dans la hiérarchie.
- SET est un logiciel propriétaire et nécessite donc un investissement non-négligeable.

8 XML Enc/Sig

XML Encryption procure la sécurité pour les applications qui requièrent l'échange sécurisé de données structurées. Il spécifie le processus de chiffrement des données et de représentation du résultat sous forme de document XML. Les données peuvent être des éléments XML, le contenu

d'éléments XML ou n'importe quelle données (y compris un document XML).

XML Digital Signature procure les services d'intégrité et d'authentification des messages ou des signataires pour des données de n'importe quel type, qu'elles soient situées à l'intérieur même du document XML qui inclut la signature ou ailleurs[20].

Forces

- Format auditable.
- Sécurité forte au niveau du message lui-même.
- Récursivité du format de message.

Faiblesses

- Complexité
- Propriétaire

9 OpenSAML

OpenSAML est la version non propriétaire de SAML (Security Assertion Markup Language). SAML est un standard XML qui permet à un utilisateur de s'enregistrer une seule fois pour pouvoir utiliser différents sites Web (affiliés). SAML est conçu pour les transactions B2B (Business-to-Business) et B2C (Business-to-Consumer).

SAML spécifie 3 composants : *assertions*, *protocol* et *binding*.

Il y a trois *assertions* : *authentification*, *attribute* et *authorization*. *Authentification assertion* valide l'identité de l'utilisateur. *Attribute assertion* contient l'information spécifique à propos de l'utilisateur. Et *authorization assertion* identifie ce que l'utilisateur est autorisé à faire.

Protocol définit comment SAML demande et reçoit les *assertions*. *Binding* définit comment les échanges de messages SAML sont convertis en échanges SOAP (Simple Object Access Protocol). SAML fonctionne avec de nombreux protocoles tels que HTTP, SMTP, FTP et supporte aussi SOAP, BizTalk et ebXML (Electronic Business XML).

[20] Voir Bilal Siddiqui, *Exploring XML Encryption*, http://www-106.ibm.com/developerworks/xml/library/x-encrypt, Août 2002.

Forces

- Sécurité forte par l'utilisation du format de message XML Enc/Sig.

Faiblesses

- Propriétaire.

10 Tableaux comparatifs

Les tableaux suivants reprennent les différents protocoles analysés précédemment. On peut toutefois remarquer la présence d'un protocole en plus, pas encore évoqué, OpenSST. Le but est de situer ce protocole, que nous analyserons dans la partie suivante, par rapport aux autres protocoles existants.

10.1 Tableau 1

Remarques:

- OpenSAML utilise XML Enc/Sig, ce qui lui permet de garantir certaines propriétés fondamentales.
- Le nombre de croix utilisées pour certains critères permet d'insister sur l'importance de ce critère pour le protocole particulier.

	S/MIME	OpenPGP	SSL/ TLS	S-HTTP	OpenSSH	SET	XML Enc/Sig	OpenSAML	OpenSST
Cadre courant	**email**	**email**	**Web**	**Web**	**Accès distant**	**paiements**	**transactions**	**transactions**	**transactions**
Transparence/auditabilité	X	X	-	-	X	-	X	X	X
Simplicité d'intégration	X	XX	XX	XX	XX	X	X	XX	XX
Indépendance/applications	-	-	X	-	X	-	-	-	-
Indépendance/plate-formes	X	X	X	-	X	-	X	X	X
Généricité/récursivité	X	X	-	-	-	-	X	-	X
Complexité	XXX	XX	X	XXX	XX	XXXX	XXX	XX	XX
Confidentialité	X	X	X	X	X	X	X	X	X
Authentification	X	X	-	X	-	X	X	X	X
Intégrité	X	X	X	X	X	X	X	X	X
Non-répudiation	X	X	-	X	-	X	X	X	X
Popularité	XX	XXX	XXX	-	XXX	X	XX	XX	-
Pérennité	X	X	X	-	X	X	X	X	X
Coût	X	X	X	XX	X	XXX	XX	X	X

Figure 12 - Tableau comparatif des protocoles 1

OpenSST, comme la plupart des protocoles repris dans ce tableau, répond bien aux 4 exigences fondamentales de la sécurité des transferts que sont :

- La confidentialité des données : Les spécifications du protocole OpenSST laissent la possibilité de l'implémenter en utilisant différents algorithmes de chiffrement. Les deux implémentations réalisées utilisent l'algorithme AES (Rijndael).

- L'authentification : Les messages OpenSST sont signés électriquement par l'émetteur de sorte que le destinataire peut être raisonnablement certain de son identité. Les spécifications laissent le choix entre les algorithmes RSA et DSS pour réaliser cette signature électronique. Les deux prototypes réalisés implémentent la version RSA. SSL et SSH, contrairement aux autres protocoles, ne garantissent pas l'authentification du message mais bien celle de la session.

- L'intégrité : L'utilisation d'algorithmes de hachage tels MD5 et SHA-1 permettent de vérifier que le message n'a pas été altéré pendant son transport.

- La non-répudiation : Ce principe est garantit également par l'utilisation d'une signature électronique. La remarque faite pour SSL et SSH pour l'authentification est la même pour la non-répudiation, c'est à dire que la session est non-répudiable, mais pas le message.

Indépendance vis-à-vis des applications : Une même et unique implémentation du protocole suffit quelle que soit l'application qui l'utilise. Ce n'est bien sûr une propriété que ne possède que SSL et SSH qui se situent au niveau de la couche de transport alors que les autres protocoles tel OpenSST se situent, eux, au niveau de la couche applicative (modèle OSI).

L'indépendance vis-à-vis des plate-formes signifie également qu'il ne faut pas modifier l'implémentation du protocole pour qu'il puisse tourner sur différents types de machine. OpenSST respecte ce critère par l'utilisation de langages tels Java et Perl.

Généricité/Récursivité : Il s'agit de la propriété reconnue à un format de message de pouvoir s'encapsuler lui-même. Par exemple, pour le protocole OpenSST, la partie data du message peut contenir un message OpenSST entier.

Exemple pratique : le cas où plusieurs serveurs OpenSST se relayent un message, chacun d'eux le signant.

Popularité : OpenSST n'étant pas encore distribué sur le marché, il ne connaît pas de popularité auprès du public.

Complexité : En comparaison avec les autres protocoles, OpenSST est relativement simple à comprendre, à implémenter et à intégrer.

Coût : OpenSST est un logiciel libre et par conséquent, le prix d'achat est nul. Il reste cependant les coûts d'intégration et de maintenance.

10.2 Tableau 2

	S/MIME	OpenPGP	SSL/TLS	S-HTTP	OpenSSH	SET	XML EncSig	OpenSAML	OpenSST
Cadre courant	email	email	Web	Web	Acces distant	paiements	transactions	transactions	transactions
Couche OSI	Application	Application	Transport	Application	Transport	Application	Application	Application	Application
Format de message	MIME	PGP	-	PEM/PGP	-	PKCS#7 + DER	XML	XML	XML
Types de certificats	X509	OpenPGP	X509	X509	Variable	X509	variable	X509	variable

Figure 13 - Tableau comparatif des protocoles 2

Dans ce tableau sont reprises les caractéristiques générales des différents protocoles. OpenSST, comme on peut s'en apercevoir, est très proche de XML Enc/Sig. Il est en effet plus général que S/MIME, limité au domaine du courrier électronique, et que SET, lui s'occupant uniquement de la partie paiement comprise dans une transaction.

OpenSST se situe, comme nous l'avons également vu précédemment, au niveau de la couche applicative. Cela signifie que son implémentation est différente d'une application à une autre. Cette caractéristique est commune à la plupart des protocoles, excepté SSL et OpenSSH qui se situent au niveau de la couche de transport et dont le rôle est de sécuriser une session plutôt que les messages eux-mêmes.

Le format de message repose sur XML, qui est un standard de par sa clarté, sa compréhension et sa facilité de traitement. Les protocoles plus anciens tels PGP ou S/MIME ont leur propre format de message. Le désavantage que l'on peut dès lors souligner est le fait que ces formats ne sont pas modifiables sans devoir également modifier l'implémentation des protocoles. Ce n'est pas le cas pour les protocoles se basant sur un format de message XML. En effet, le problème ne se pose pas pour XML. On peut rajouter des éléments sans problème de compatibilité avec les versions existantes du protocole. Ces nouveaux éléments sont tout simplement ignorés.

Les certificats utilisés pourront être différents d'une implémentation à une autre. Le prototype déjà développé en Java utilisera les certificats X509 mais le prototype en Perl utilise, lui, le format custom de la librairie crypto.

10.3 Tableau 3

Remarque :

- Pour les algorithmes de chiffrement, les nombres du tableau représentent la longueur (en bit) de la clé de chiffrement qui peut être utilisée.

	S/MIME	OpenPGP	SSL/TLS	S-HTTP	OpenSSH	SET	XML Enc/Sig	OpenSAML	OpenSST
Echange de clés									
pre-shared key		X			X	X			X
RSA	X	X	X	X	X	X	X	X	X
DSS	X	X	X		X	X			X
Algorithme de chiffrement									
DES	56	56	56	56		56			56
IDEA	128	128		128					128
3DES	168	168	168	168	168		168	168	168
RC2	40/64/128			Variable					
RC4			40/128						
RC5	X								variable
CAST5	40/64/80/128	128							variable
BLOWFISH		128			128				variable
TWOFISH		256							variable
AES		128/192/256					128/256	128/256	variable
CDMF				40					
SAFER		128							
Signature									
RSA	X	X	X	X	X	X	X	X	X
DSS		X	X	X	X		X	X	X
Digest algorithm									
MD5	128	128	128	128					128
SHA1	160	160	160	160			160	160	160
MD2	128	128		128					
RIPEMD-160		160						128	
TIGER192		192							
HAVAL-5-160		160							

Figure 14 - Tableau comparatif des protocoles 3

Il faut insister sur le fait qu'un algorithme n'est pas meilleur qu'un autre s'il supporte plus d'algorithmes de chiffrement. Il est cependant intéressant de constater que OpenSST prend en compte, dans ses spécifications, la possibilité d'implémenter toute une série d'algorithmes qui assurent un niveau différent de sécurité et qui affecteront les performances du protocole. Les prototypes actuellement implémentés utilisent RSA, AES et SHA-1.

11 Constat

On peut donc tirer quelques conclusions quant aux différentes solutions proposées pour la sécurisation des transactions.

Soit ce sont des solutions propriétaires avec toutes les contraintes et désavantages que cela entraîne. Il y a une dépendance certaine vis-à-vis du fournisseur quant à la résolution des problèmes, la mise à jour des versions, les ajouts de fonctionnalité,... Le code de la solution n'étant pas auditable, il est impossible de connaître avec certitude le procédé de sécurisation. Les acquéreurs de la solution sont donc contraints de faire entièrement confiance au fournisseur. L'investissement est également plus élevé dans le cas d'une solution propriétaire. L'équivalent libre n'est pas nécessairement gratuit mais son coût d'acquisition est plus modeste. Ce n'est pas le seul coût à prendre en compte. Les coûts d'entrée et d'exploitation sont, en général, bien inférieurs dans le logiciel libre. Le coût de migration d'une solution propriétaire vers une solution libre peut toutefois se révéler non négligeable. Il convient donc de bien l'évaluer. Le dernier désavantage des solutions propriétaires, et non le moindre est la pérennité. Pour pouvoir adapter la solution à l'évolution constante des architectures matérielles, il est utile de disposer des sources du logiciel. Ce n'est pas le cas pour le logiciel propriétaire qui, acheté généralement très cher, s'avère inutilisable après un certain temps. Le gros risque est également la faillite du fournisseur. Dans ce cas, l'acquéreur se retrouve dans une impasse et doit alors changer de solution.

Soit ce sont des solutions libres mais pour lesquelles on rencontre divers problèmes tels l'incomplétude ou la complexité.

En bref, il n'existe pas, à l'heure actuelle, de standard de transaction simple, sécurisé et complet. En effet, on trouve soit des solutions bon marché, faciles à mettre en œuvre mais incomplètes (SSL/TLS, OpenPGP,...), soit des solutions complètes mais lourdes à installer, complexes et chères (SET). De plus, les solutions sont souvent limitées en terme de plate-forme d'application.

12 Références

- Glossaire Whatis.com, http://whatis.techtarget.com .
- Glossaire de cryptographie de X5, http://www.x5.net/faqs/crypto .
- Glossaire de cryptographie, http://www.geocities.com/openpgp/glossaire.htm .

- RFC 2311 sur S/MIME, http://www.ietf.org/rfc/rfc2311.txt .
- Site de Vulgarisation Informatique, http://www.commentcamarche.net .
- OpenPGP Alliance Website, http://www.openpgp.org .
- RFC 2440 sur OpenPGP, http://www.ietf.org/rfc/rfc2440.txt .
- Glossaire de Searchsecurity, http://searchsecurity.techtarget.com .
- RFC 2246 sur TLS v 1.0. http://www.ietf.org/rfc/rfc2246.txt .
- Adam Shostack, *An overview of SHTTP,* http://www.homeport.org/~adam/shttp.html, mai 1995.
- RFC 2660 sur S-HTTP, http://www.ietf.org/rfc/rfc2660.txt.
- Rajan Bhaktar, *How SSH protects your remote access sessions*, http://www.cas.mcmaster.ca/~wmfarmer/SE-4C03-02/projects/student_work/bhaktar.html, 23 mars 2002.
- D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.
- Bilal Siddiqui, *Exploring XML Encryption*, http://www-106.ibm.com/developerworks/xml/library/x-encrypt, Août 2002.
- Neil J. Rubenking, *Securing Web Services*, PC Magazine, http://www.pcmag.com/article2/0,4149,519023,00.asp, 1er Octobre 2002.
- Paul Desmond, *SAML Just The Start For Web Services Security*, eSecurity planet.com, http://www.esecurityplanet.com/views/article.php/10752_1498491, 11 novembre 2002.
- Oasis-open consortium Website, http://www.oasis-open.org .
- OpenSST Website, http://opensst.org .

OpenSST

1 Généralités

Après avoir défini le concept de transaction électronique et les besoins en sécurisation de ces transactions et après avoir présenté un certain nombre de protocoles et formats de messages qui essayent d'apporter cette sécurisation indispensable, abordons maintenant un autre protocole qui n'en est actuellement qu'à ses balbutiements: OpenSST (pour Open, Simple, Secure, Transactions).

C'est dans l'optique d'un protocole fédérateur qu'a été développé le protocole OpenSST.

L'idée du format de message et du protocole OpenSST vient d'une société luxembourgeoise spécialisée dans le domaine des développements d'applications transactionnelles et sécurisées. L'idée de départ d'Aubay consistait à créer une solution de sécurité en matière d'application e-banking. (Le prototype du proxy HTTP OpenSST (Cf. infra) est orienté selon cette idée de départ). Aubay a ensuite demandé la collaboration de Conostix, start-up spécialisée dans le développement Open Source d'applications sécurisées. Le dernier acteur est le Centre de Recherche Public Henri Tudor et plus particulièrement le CITI, Centre d'Innovation par les Technologies de l'Information. Créé en 1987, le Centre de Recherche Public Henri Tudor a comme première finalité de renforcer le tissu économique luxembourgeois par le développement technologique et l'innovation, entre autres dans les TIC, Technologies de l'information et de la communication[21]. De par sa neutralité sur le marché et la logistique qu'il pouvait apporter à Aubay et Conostix, le CITI était le candidat idéal pour une coopération active. Le projet OpenSST entre dans le cadre d'un projet FNR (Fond National de Recherche) appelé ACCES-PME.

Le projet a débuté en avril 2002 et depuis, beaucoup de chemin a été accompli. Le point 2 de ce chapitre reprend les contraintes et exigences établies suite à des entretiens réalisés auprès de 12 sociétés luxembourgeoises faisant partie de différents secteurs : Construction, secteur bancaire, professions libérales, ... Le point 3 détaille le format du message OpenSST. Dans le point 4, différents cas d'utilisation sont repris et la solution OpenSST envisagée est expliquée en détail. Enfin, l'avenir du protocole est expliqué au point 5.

[21] Voir http://www.tudor.lu .

« OpenSST est un format standardisé de message sécurisé pour les transactions électroniques. Ce standard est décomposé en plusieurs définitions : d'une part la structure des messages et d'une autre part les différentes interactions. L'approche majeure d'OpenSST est d'avoir une structure simple pour les messages qui peut servir à plusieurs cadres d'utilisation (des applications bancaires à la gestion des systèmes embarqués). »[22]

2 Contraintes et exigences

Avant de définir le protocole OpenSST, il était indispensable de passer en revue le marché luxembourgeois pour connaître leurs exigences en matière de sécurité des transactions électroniques. Un panel de sociétés couvrant plusieurs secteurs a donc été interrogé pour pouvoir définir un certain nombre de contraintes et exigences auxquelles le protocole OpenSST doit se plier. C'est le but de ce point 2. Les exigences majeures sont reprises et sont classées par type d'application que le protocole OpenSST doit être capable de sécuriser.

Par souci d'anonymat des sociétés interrogées, les noms ne seront pas cités et seront remplacés par le nom du secteur dans lequel elles se situent et un numéro.

2.1 HTTP

Fonctions normales :

- Transmission de pages Web.
- Echanges de fichiers (textes, images, sons, vidéos ou autres fichiers multimédias).

Mécanismes de sécurités possibles :

- Authentification du client et du serveur HTTP.
- Chiffrement des pages Web ou des fichiers.
- Intégrité des pages Web ou des fichiers.
- Signature des pages Web ou des fichiers.

Solutions possibles :

- Proxy HTTP sur le poste client.
- Transfert via HTTP (par ex: monitoring de systèmes).

---

[22] Voir A. Dulaunoy, S. Stormacq, *OpenSST : Open Simple Secure Transaction : Une approche de réduction de la complexité pour les transactions électroniques*, Janvier 2003.

Secteurs cibles :

- e-banking : possibilité de réaliser ses transactions bancaires online.
- e-paiement : faire des achats, effectuer des paiements online.
- e-gouvernement : accéder aux services administratifs online.
- Construction : consultation de plans, de projets et téléchargements online.
- Professions libérales.

exemples concrets :

- Assurance-1 : Application Web pour la consultation de portefeuilles. Il faudrait toutefois prévoir des modules complémentaires pour intégrer OpenSST à l'architecture actuelle:
  - Module de gestion des autorisations d'accès aux pages ASP.
  - Fichier XML pour configurer la politique de sécurité du serveur.
  - Module d'administration des certificats (vérification de la validité).

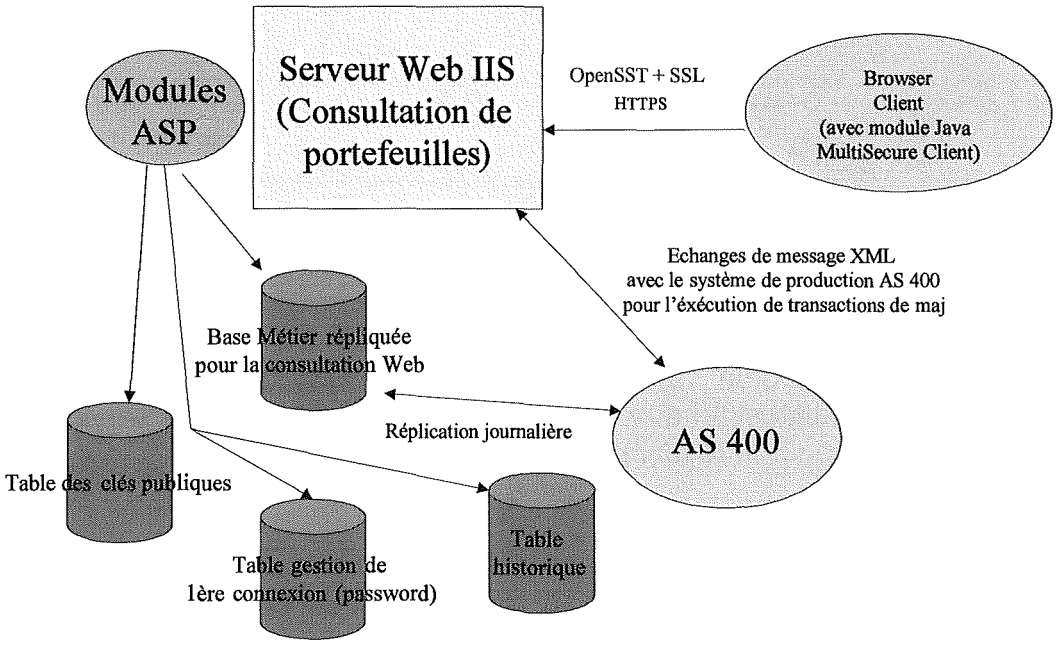

Figure 15 - Exemple 1 : Consultation de portefeuilles online

- Construction-1.
  - Consultation de pages Web et téléchargement de documents.

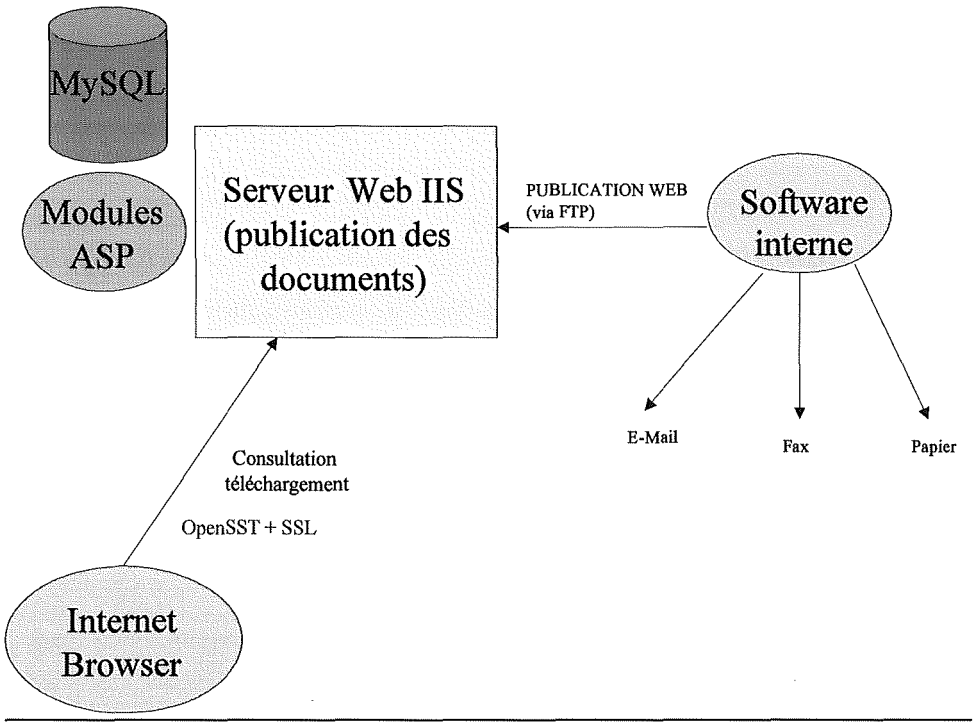

Figure 16 - Exemple 2 : Consultation de pages Web et téléchargement

- Avocat-1:
  - Intégration d'un système d'authentification fort sur des sites d'intermédiation tel que « escrow.com »(consignation de services avec des tiers pour la réservation de noms de domaines).

- Construction-2 :
  - Sécurisation du site Web accessible à certains clients de construction-2 et reposant actuellement sur une solution Silverstream.

- SANTEC – CRP Henri Tudor :
  - HealthNet : développement de systèmes d'échange de résultats de laboratoires par la technologie OpenSSL avec la gestion de la distribution des clés asymétriques et d'un certificat « HealthNet ».

- Construction-3 / Construction-4. :
  - Chiffrement et signature électronique des plans échangés via le service Forum « SiteScape » fourni par Forum Network. Nécessité de s'intégrer dans une infrastructure type PKI.

- Développement possible au niveau des modules Tcl de SiteScape afin de répercuter les signatures émises dans l'interface.

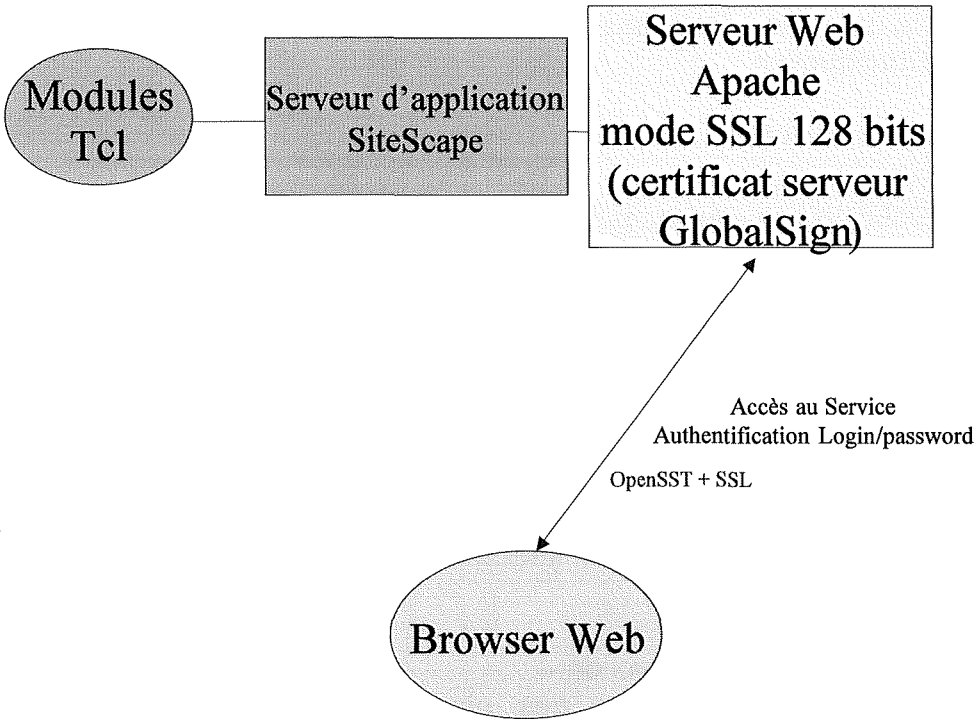

Figure 17 - Exemple 3 : Accès au serveur avec authentification

2.2 Messagerie électronique (SMTP)

Fonctions normales :

- Envoyer et recevoir du courrier électronique. (Mail Transfer Agent)

Mécanismes de sécurités possibles :

- Authentification de l'émetteur.
- Chiffrement des messages et fichiers envoyés.
- Intégrité des messages et fichiers envoyés.
- Signature de l'émetteur et du message.

Solutions possibles :

- Client OpenSST utilisant infrastructure SMTP.
- Simple application permettant d'encapsuler un message écrit en clair dans le format OpenSST. Ensuite il suffit de faire un copier/coller dans la fenêtre réservée au message dans n'importe quelle application de messagerie électronique. Le destinataire effectue l'opération dans l'autre sens lors de la réception.

Secteurs cibles :

- Industrie de l'acier.
- Professions libérales.
- Assurances – Banques.
- Construction.

Exemples concrets :

- Construction-2 :
  - Solution actuelle reposant sur MS Exchange 2000.

- SANTEC – CRP Henri Tudor :
  - Transmission de dossiers et de résultats entre médecins et laboratoires (fiches XML), entre hôpitaux.

- Avocat-1 :
  - Partage et diffusion des contrats ou des mandats avec le client (besoin de garantie quant à la signature des documents transmis, problème d'authentification, de chiffrement des données et également de timestamping).

- Construction-3 / Construction-4 :
  - Signature électronique des plans échangés par le biais de la messagerie électronique (besoin d'infrastructure type PKI).

- Construction1.
  - Distribution des documents par email.

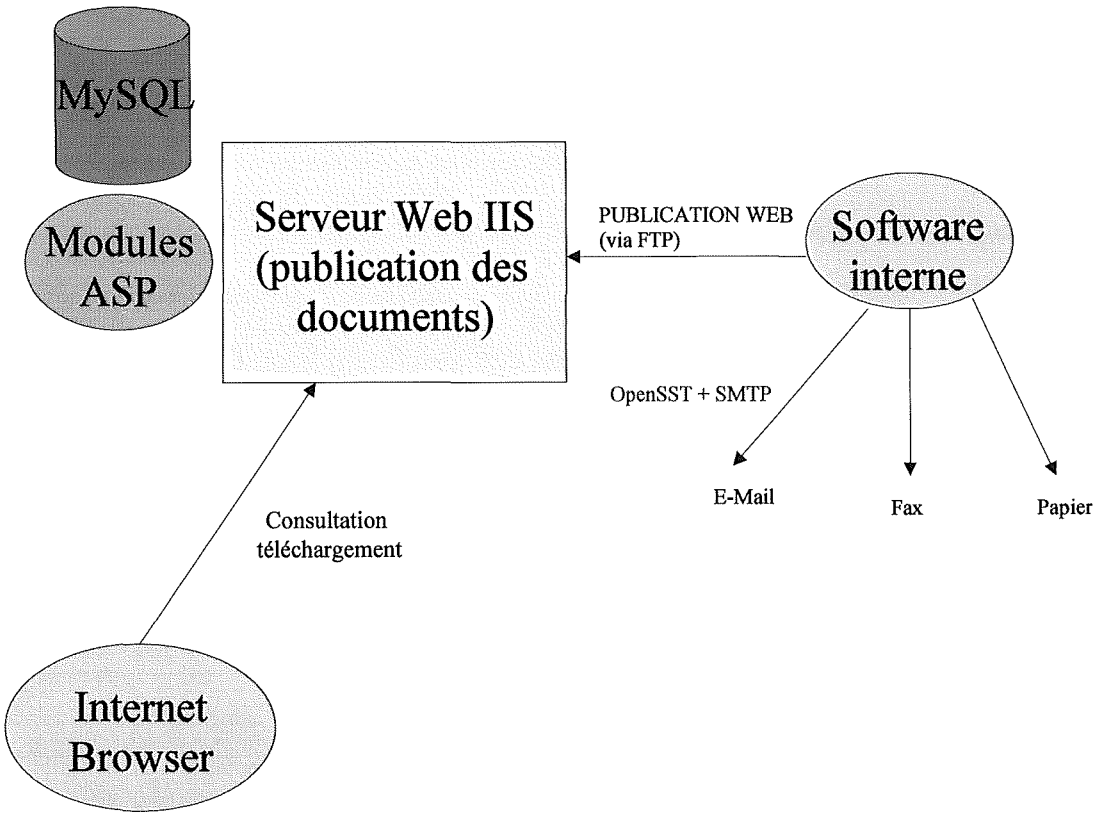

Figure 18 - Exemple 4 : Distribution de documents par email

2.3 FTP

Fonctions normales :

- Echange de fichiers.

Mécanismes de sécurités possibles :

- Authentification des parties
- Chiffrement des fichiers envoyés.
- Intégrité des fichiers envoyés.
- Signature des fichiers envoyés.

Solutions possibles :

- Application qui prend en entrée le fichier à transférer et qui donne en sortie le fichier chiffré et signé. Ensuite, il ne reste plus qu'à transférer le fichier normalement.

- Application avec une petite interface graphique qui encapsule le fichier dans le format OpenSST et qui le transfère automatiquement en FTP.

Secteurs cibles :

- Industrie de l'acier.
- Assurances – Banques.
- Construction.

Exemples concrets :

- Construction-2 :
  - Solution de transmission de certificats qualité aux clients de construction-2 (chiffrer et signer ces certificats avec OpenSST).

- Construction3 / Construction-4 :
  - Signature électronique des plans échangés par le biais de FTP (besoin d'infrastructure type PKI).

- Assurance1 :
  - Réplication journalière de la base de données Métier.

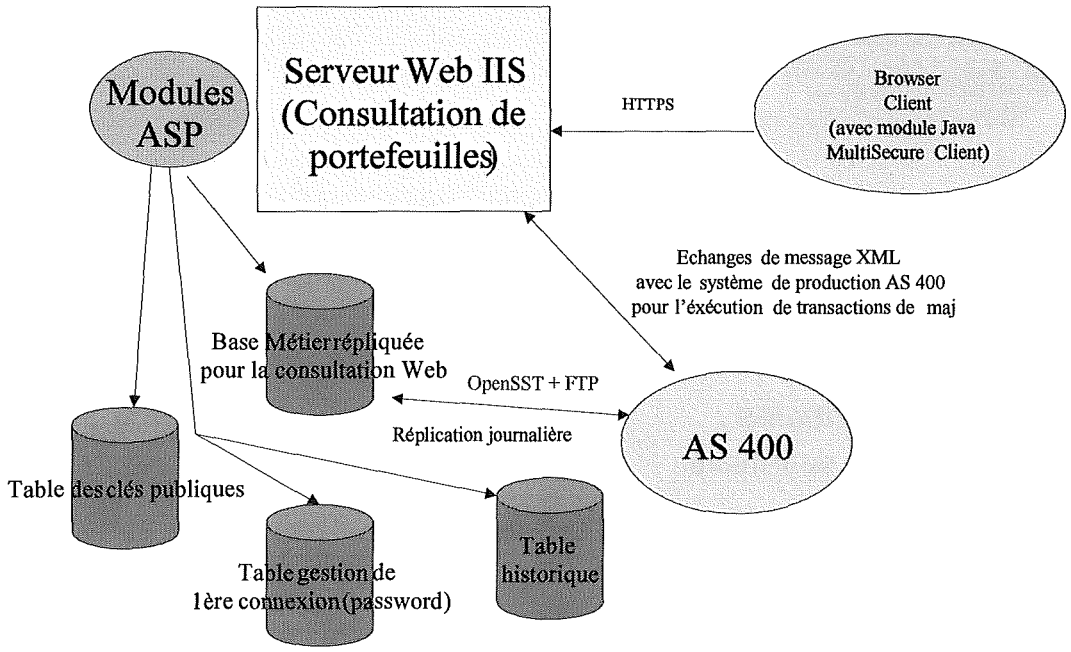

Figure 19 - Exemple 5 : Réplication journalière d'une base de donnée

- Construction-1.
  - Export de fichiers vers un serveur Web.

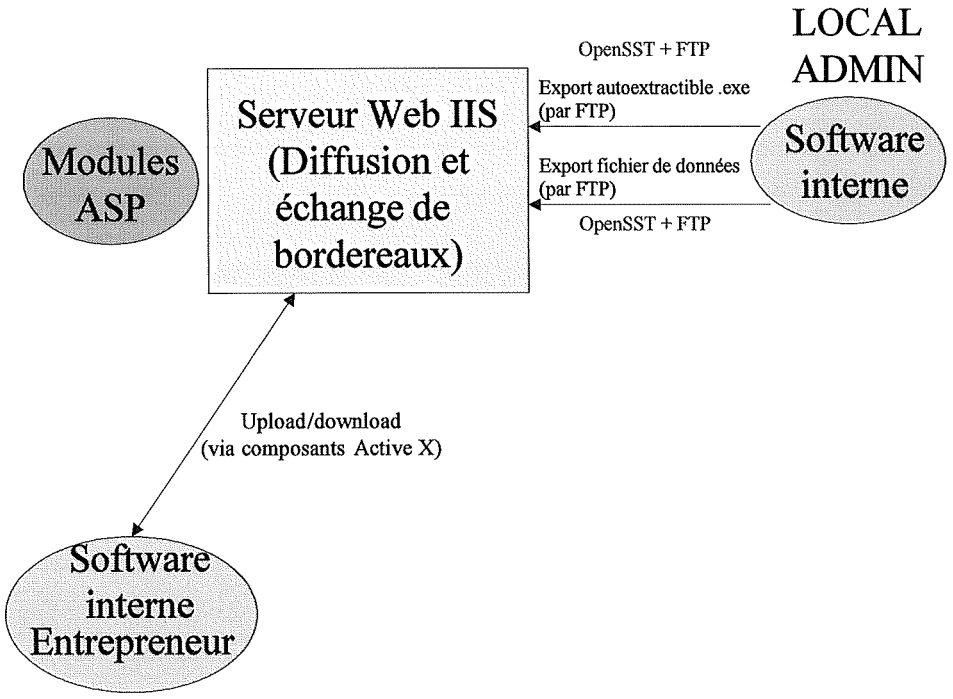

Figure 20 - Exemple 6 : Export de fichiers vers un serveur Web

- Construction-1.
  - Transfert de documents signés et chiffrés vers le serveur Web.

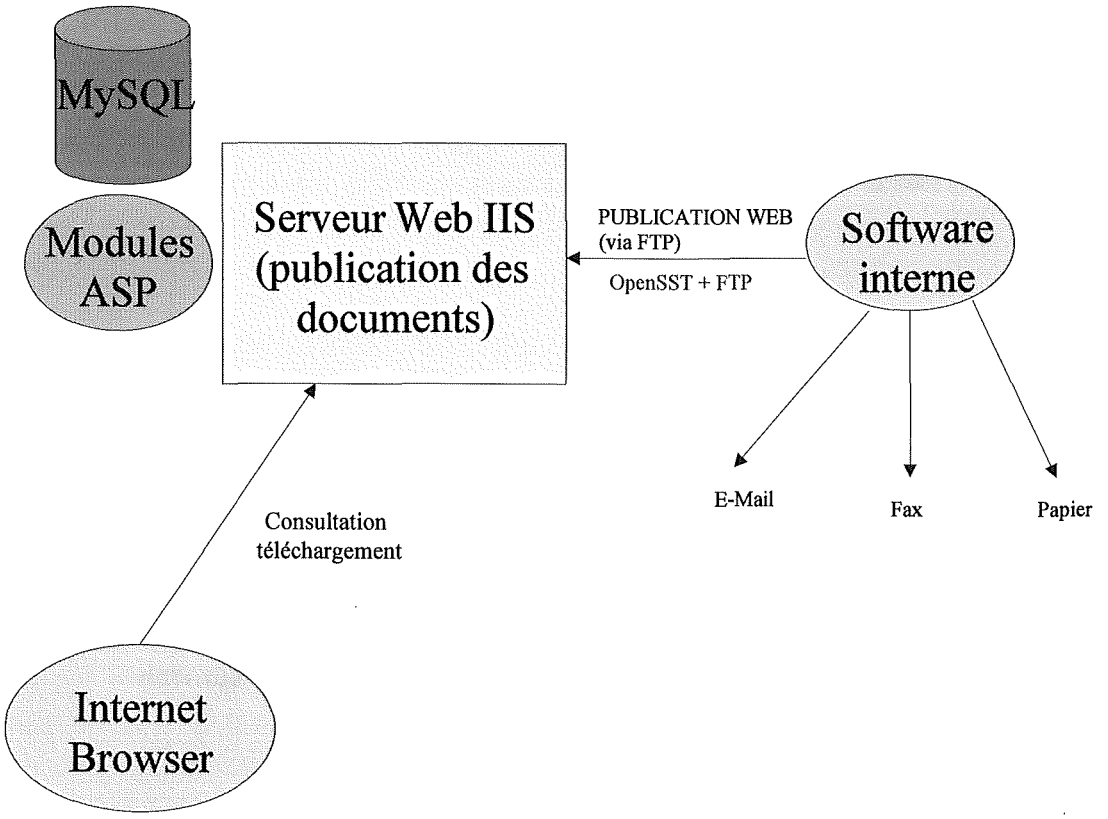

Figure 21 - Exemple 7 : Transfert de documents signés et chiffrés vers un serveur Web

2.4 Stockage

Fonctions normales :

- Simple stockage en local de documents de tous types possibles (PDF, DOC, TXT, ...).

Mécanismes de sécurités possibles :

- Chiffrement des documents stockés.
- Signature des documents stockés.

Solutions possibles et composants nécessaires :

- Simple application qui prend en input un fichier quelconque et sort en output le fichier chiffré et signé au format OpenSST.

Secteurs cibles :

- Tout type de secteur.

Exemples concrets :

- Construction-2 :
  - Application de gestion des archives.

- Construction-1.
  - Stockage en local de bordereaux signés.

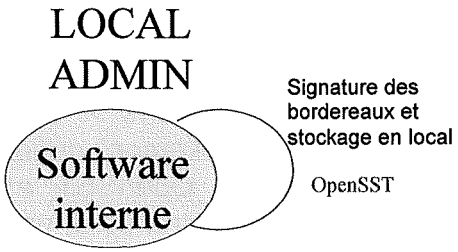

Figure 22 - Exemple 8 : Stockage en local de bordereaux signés

2.5 Commentaires supplémentaires

D'autres exigences ont été émises par les différentes sociétés. Ces exigences sont reprises ci-dessous avec quelques commentaires.

- Mise en place d'une structure d'accréditation autour du projet : L'organisme cité pour cette structure est l'OLAS (Office Luxembourgeois d'Accréditation et de Surveillance).

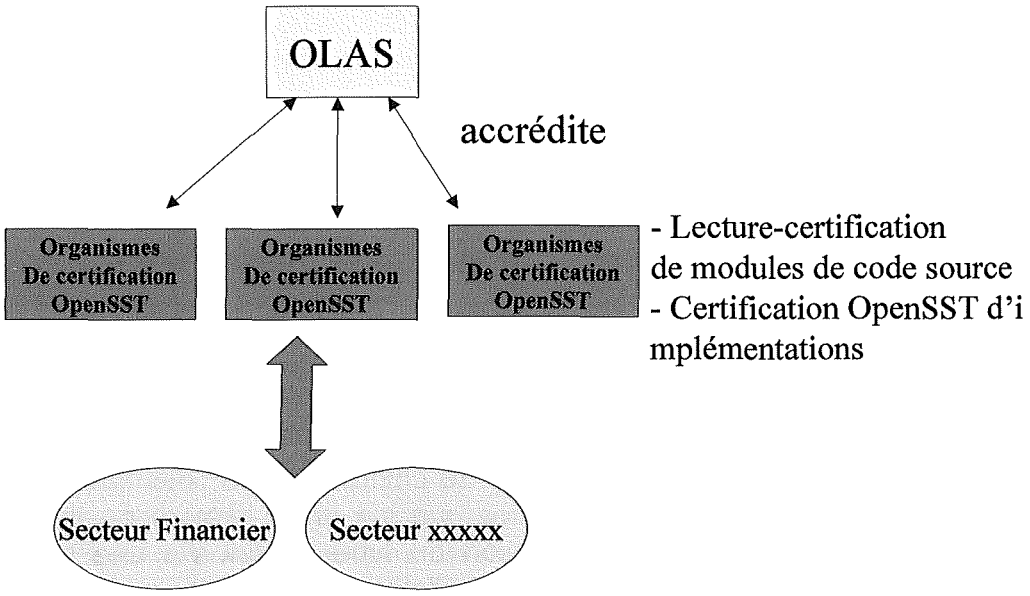

Figure 23 - Exemple de structure d'accréditation

- Capacité d'OpenSST de chiffrer toutes formes de données.
- Authentification des différents acteurs de la transaction.
- Fonctionnement sous le protocole SSH.

3 Format du message

Une fois les contraintes et exigences spécifiées, il est possible de déterminer le protocole qui doit les satisfaire. Au point 1, il est dit que OpenSST est avant tout un format de message. Les différentes interactions dépendant des applications à sécuriser ne peuvent être définies qu'une fois le format formellement défini. Cette partie décrira et définira le format des messages OpenSST et au point suivant, les différents types d'applications seront traités et une solution sera proposée pour chacun d'eux.

Le message OpenSST se présente comme un document XML (eXtensible Markup Language). En effet, XML est évolutif et portable. La forme d'un message OpenSST se présente comme suit :

```
<opensst xmlns...>

<encryption type="0-2-0-8" id="example" .../>

<data type=.../>
...
</data>

<signature type="1-3-1" id="example" ...>
dNGQnaD...
</signature>

</opensst>
```

Il y a trois grandes parties dans la structure même du message OpenSST :

– Encryption
– Data
– Signature

L'élément *encryption* peut apparaître de 0 à n fois dans un même message OpenSST. Il définit le type de chiffrement de l'élément *data* via l'attribut *type* ainsi que la clé symétrique via un id (*keyId*) ou la clé de session (*symmetricKey*). La définition du type se fait via une table de correspondance concernant le choix de l'algorithme cryptographique, ainsi

que les méthodes de "padding" mais aussi le mode de fonctionnement en tant que système cryptographique hybride ou non.

L'élément *data* ne peut apparaître qu'une fois dans un même message OpenSST. Il contient les données (souvent) chiffrées. L'attribut *encoding* permet de définir le type d'encodage de cette partie (par exemple : Base64). L'attribut *type* définit le type du message. Les types ne sont pas décrits dans le standard de base mais dans des bases complémentaires. Ce type peut être un autre message OpenSST mais aussi des données spécifiques. Il existe souvent aussi deux sous-éléments à data : *tid* et *timestamp*. Ils sont localisés dans cet élément car le chiffrement est effectif à l'intérieur de l'élément *data*. D'autres sous-éléments peuvent apparaître suivant le type du message lui-même.

L'élément *signature* peut apparaître de 0 à n fois dans un même message OpenSST. Il contient la signature de la partie data. Le type et la méthode de signature sont contenus dans l'attribut *type* qui est décrit dans une table correspondance équivalente à la partie *encryption*.

L'ensemble du format de message est décrit dans le schéma XML OpenSST basique. Ce format de message exprime le strict minimum de la transaction, Ce sont les utilisations propres des types qui étendent la syntaxe et les échanges[23].

4 Cas d'application et fiches techniques

Après avoir défini le format des messages OpenSST, il est temps maintenant de traiter les interactions du protocole OpenSST avec les différentes applications qu'il devra sécuriser. Trois types d'applications à sécuriser sont ressortis de l'analyse des contraintes et exigences exprimées par les sociétés luxembourgeoises:

- Les transferts en ligne par le biais du protocole HTTP.
- Les transferts offline ou hors connexion et le stockage.
- La messagerie électronique (protocole SMTP).

4.1 Proxy HTTP et paiement électronique

Le protocole HTTP est un protocole pour la transmission de pages Web et l'échange de fichiers (textes, images, sons, vidéos,...). Il est utilisé dans le cadre de l'e-business et pour des applications telles que l'e-banking, l'e-

[23] Voir A. Dulaunoy, S. Stormacq, *OpenSST : Open Simple Secure Transaction : Une approche de réduction de la complexité pour les transactions électroniques*, Janvier 2003 et Dulaunoy, T. Fruru et S. Stormacq, *OpenSST Message Format*, Internet Drafts, December 2002.

paiement pour les sites d'achats online,... Il est donc capital de pouvoir sécuriser les transactions effectuées pour mettre en confiance les utilisateurs potentiels et éviter les attaques qui pourraient causer de pertes financières importantes.

La solution OpenSST se présente comme suit :

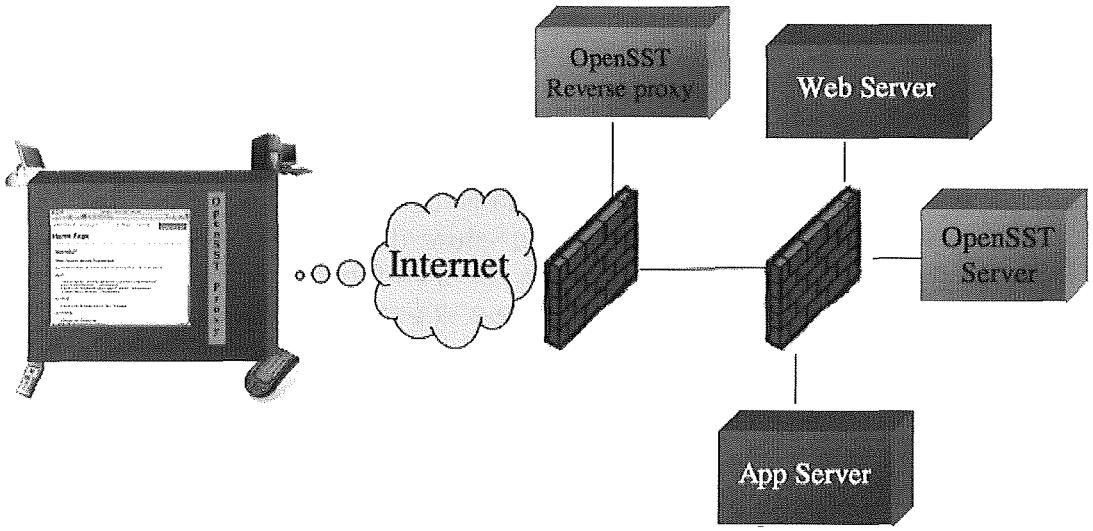

Figure 24 - Architecture du proxy HTTP OpenSST

La solution choisie est en fait l'utilisation d'un proxy OpenSST chez le client qui intercepte toutes les requêtes effectuées par le browser, qui les traite en les convertissant au format OpenSST et les transmet ensuite au serveur OpenSST. Celui-ci, lorsqu'il reçoit ces messages OpenSST, décapsule la requête HTTP initiale et peut alors les transmettre au serveur Web qui ne s'aperçoit pas des mécanismes de sécurité mis en place. L'opération est donc complètement transparente pour le client et pour le serveur Web. Dans le sens inverse, le procédé s'effectue de manière identique[24].

Le troisième composant que l'on remarque sur ce schéma est un *reverse proxy* OpenSST. Celui-ci joue en fait le rôle d'intermédiaire entre le proxy et le serveur OpenSST. Il intercepte les messages OpenSST provenant du proxy, en vérifie la syntaxe, mais pas la sémantique, pour s'assurer qu'il s'agit bien d'un message OpenSST bien formé. Quand il a

[24] Voir S. Stormacq, *OpenSST Message Type : HTTP proxy*, Internet Drafts, December 2002 et C. Feltus, D. Khadraoui, R. Hallez, F. Costa Pinto, *Expérimentation du protocole OpenSST pour les transactions électroniques*, CITI (Centre d'Innovation par les Technologies de l'Information) du CRPHT (Centre de Recherche Public Henri Tudor), Mars 2003.

effectué ces vérifications, il retransmet le message au serveur. S'il détecte une erreur, il renvoie au proxy un message HTTP avec un code d'erreur. L'erreur de syntaxe peut être apparaître dans plusieurs circonstances :

- Une modification par un tiers mal intentionné lorsque le message est sur le réseau.
- Un message formé de toute pièce qui ne respecte pas le format.
- Un programme buggé.
- Une attaque de type buffer overflow.
- ...

Le protocole se déroule en plusieurs étapes. Voici un schéma du protocole (laissons de côté le reverse proxy qui ne fait que de la vérification de syntaxe) :

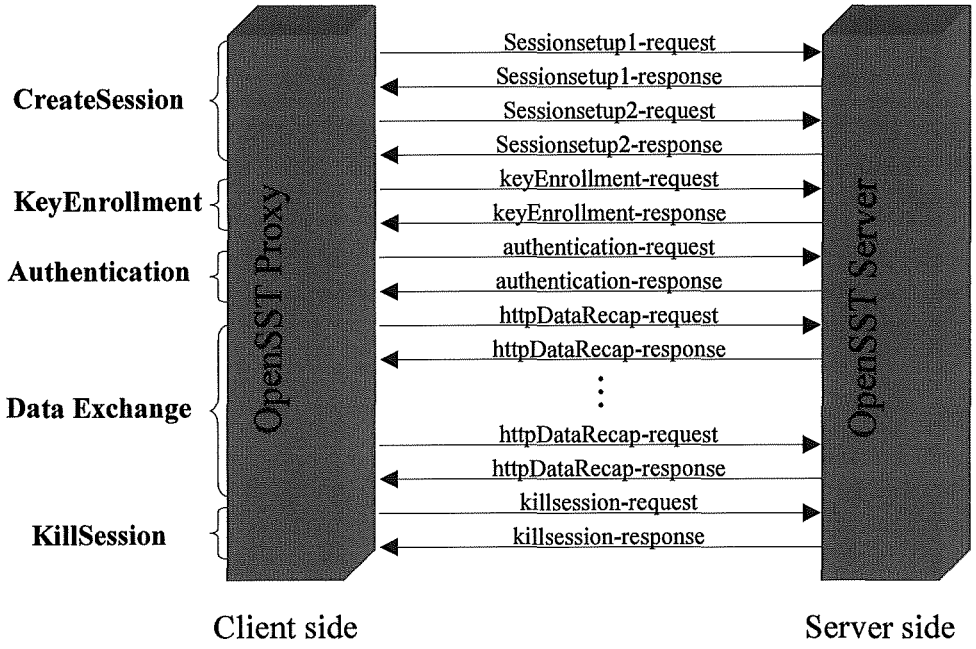

Figure 25 - Messages échangés dans le cas du proxy OpenSST

Tous les types de messages qui seront cités par la suite sont repris dans un RFC relatif à la solution proxy HTTP de OpenSST.

Les exemples de partie *data* sont restreints, dans ce document, aux éléments clés de chaque message. Les tags XML obligatoires ou optionnels dans la partie *data* sont spécifiés dans un schéma XML spécifique à chaque type de message.

CreateSession

Cette étape a pour but que le proxy obtienne la clé publique du serveur, qu'une clé de session soit créée et échangée et que le proxy identifie clairement le serveur.

Phase 1 :

- Le proxy envoie un message (pas de chiffrement, pas de signature) au serveur pour obtenir sa clé publique.

  Techniquement, l'attribut *type* de l'élément *data* est mis à *sessionSetup1*.
  L'élément *data* contient :

```
<data>
  <request>
    <ip>openSST proxy's ip address</ip>
  </request>
</data>
```

  L'élément *ip* n'est là que pour éviter d'avoir un élément *data* vide mais une autre information aurait très bien pu convenir.

- Le serveur répond au proxy par un message dont le *type* est mis à *sessionSetup1*.

```
<data>
  <response>
    <publickey format="base64">AB5F32 ....</publickey>

    <session-id>32AE6BC ... </session-id>

    <salt format="base64">43MkQ ... </salt>
  </response>
</data>
```

  Pour le moment le proxy demande la clé publique du serveur. A très court terme, il faudra plutôt demander le certificat du serveur. Le proxy devra vérifier la validité du certificat

  Le *session-id* est généré par le serveur et devrait être utilisé par le proxy pour les messages ultérieurs.

  Le salt (nombre aléatoire) généré par le serveur peut être utilisé par le proxy pour générer des clés.

En effet, pour la génération de clés, il est nécessaire de disposer d'un nombre aléatoire comme point de départ (cela permet de déterminer les nombres premiers qui seront utilisés). La qualité de ces nombres aléatoires est très importante : si on peut prévoir quel nombre aléatoire va être utilisé, on peut retrouver la clé. Comme les postes clients ne disposent pas nécessairement d'une source aléatoire de qualité, le serveur envoie un nombre aléatoire qui permettra au proxy de générer les clés. Comme l'environnement serveur est contrôlé, ce nombre sera de bonne qualité (source hardware ou autre).
En revanche, si le proxy dispose d'une bonne source aléatoire, il n'est pas obligé d'utiliser le salt qu'on lui envoie.

Phase 2 :

- Le proxy génère une clé de session (chiffrée avec la clé publique du serveur) et l'envoie au serveur. Le *type* de data est mis à *sessionSetup2*.

```
<data>
  <request>
    <sessionkey format="base64" encrypt="algo">AB5F32 ....</sessionkey>
  </request>
</data>
```

L'attribut *encrypt* de l'élément *sessionkey* correspond à l'algorithme utilisé pour encoder la clé.
L'attribut *format* désigne l'algorithme utilisé pour encoder les données en ASCII.

- Le serveur lui répond (*type* à *sessionSetup2*)

```
<data>
  <response>
    <code>XXX</code>

    <message>Success</message>

    <magic-value>AJHe52S...</magic-value>
  </response>
</data>
```

La *magic-value* correspond au hash du *session-id* et de la clé de session, le tout exprimé en base 64 :
base64(SHA-1(session-id + session-key))

où l'opérateur « + » représente la concaténation

Cette valeur est également calculée par le proxy. Si les deux valeurs sont identiques, le proxy est assuré de l'identité du serveur.
Il faut remarquer que, faute de clé commune entre le proxy et le serveur, les 4 messages précédents sont échangés en clair sur le réseau.

A partir de ce point, tous les messages entre le proxy et le serveur sont chiffrés avec la clé de session.

KeyEnrollment

Cette séquence de message ne s'effectue qu'une seule fois pour chaque utilisateur. Son objectif est que le serveur reçoive la clé publique du client.

On suppose que le client a reçu de manière sécurisée (par courrier postal, au guichet,...) un userid et un pincode. En effet, les échanges physiques existent toujours. Ils resteront les échanges les plus sécurisés quel que soit le développement des technologies.

Le proxy génère une paire de clés (privée et publique). La clé privée, chiffrée grâce à une passphrase fournie par le client, est stockée dans un fichier local avec le userid et un identificateur (profile) choisi par le client.

- Le proxy envoie au serveur un message (*type* à *keyEnrollment*) qui contient le userid (*token*), la clé publique et un otp (one-time password) qui est un HMAC du pincode et de la clé publique (otp = HMAC (pincode, PuK)).

```
<data>
  <request>
    <token>user id</token>

    <otp> ... </otp>

    <publickey format="base64">AB5F32 ....</publickey>
  </request>
</data>
```

  L'attribut *format* de *publickey* est le nom de l'algorithme utilisé pour convertir la donnée en ASCII.

- Comme le serveur possède une base de données associant les userid et les pincodes, il peut vérifier l'identité du client (en recalculant le HMAC) et donc avoir confiance dans la clé publique qu'il a reçue. Il répond comme suit (*type* à *keyEnrollment*) :

```
<data>
  <response>
      <code>XXX</code>

      <message>Success</message>
  </response>
</data>
```

Les éléments *code* et *message* permettent au serveur de communiquer au proxy le résultat de l'opération.

Authentication

Cette étape est toute simple et permet au serveur de s'assurer de l'identité du client.

- Le proxy envoie juste au serveur son userid (que le serveur possède dans une base de données). Il signe le message (*type* à *authentication*).

```
<data>
  <request>
    <token>user id</token>
  </request>
</data>
```

- Le serveur doit juste rechercher dans sa base de données la clé publique correspondant à ce userid. Il peut ensuite vérifier la signature. Il rèpond alors au client (*type* à *authentication*).

```
<data>
  <response>
      <code>XXX</code>

      <message>Success</result>
  </response>
</data>
```

Les éléments *code* et *message* permettent au serveur de communiquer au proxy le résultat de l'opération.

Data Exchange

Une fois l'utilisateur dûment authentifié, les messages échangés entre le proxy et le serveur ne contiennent plus que des requêtes HTTP et des réponses HTML.

- La requête HTTP qui provient du browser du client est interceptée par le proxy qui l'encapsule au format OpenSST comme suit (*type* de *data* à *httpDataDecap*) :

```
<data>
  <request>
    <!-- the session id is assigned by the server at initialization time -->
    <session-id>....</session-id>

    <url type="post/get/delete/">
    http://www.foo.com/foo
    </url>

    <params>
      <!-- the url params as sent by the client -->
      <param name="name">value</param>
    </params>

    <http-headers>
      <!-- the http header as sent by the client -->
      <http-header name="name">value</http-header>
    </http-headers>
  </request>
</data>
```

L'élément *session-id* contient le numéro de session donné par le serveur lors de l'établissement de la session ;
L'élément *url* contient l'URL vers laquelle le serveur doit faire suivre la requête ;
L'élément *params* contient les paramètres de l'URL, tels que passés par le navigateur ;
L'élément *http-headers* contient les en-têtes du protocole HTTP telles que passées par le navigateur internet.

- Le serveur répond (*type* à *httpDataDecap*) :

```
<data>
  <response>
    <!-- the session id is assigned by the server at initialization time -->
    <session-id>....</session-id>

    <http-code> <!-- 200, 404, ... --></http-code>

    <answer>
      <!-- the raw data as received from the remote host
        ** always Base64-encoded ** -->
      <!-- empty (or absent ?) when the http-code != 200 -->
    </answer>

    <http-headers>
      <!-- the http header as sent by the server -->
      <http-header name="name">value</http-header>
    </http-headers>
```

```
  </response>
</data>
```

L'élément *session-id* contient le numéro de session donné par le serveur lors de l'établissement de la session ;
L'élément *http-code* contient le code HTTP issu par le serveur web cible ;
L'élément *answer* contient la réponse du serveur web cible ; cette réponse peut contenir de l'HTML, des feuilles de styles CSS, des images JPG, etc .. Comme la réponse peut contenir des éléments binaires, elle est toujours encodée en base 64 ;
L'élément *http-headers* contient les en-têtes du protocole HTTP telles que reçues du serveur web cible (y inclus les cookies, etc ...).

Remarque : Une requête pour une simple page Web peut générer plusieurs requêtes sous-jacentes pour obtenir les images, les feuilles de styles etc ...

KillSession

Cette étape sert, comme son nom l'indique, à mettre fin à la session.

- Le proxy envoie un message vide (*type* à *killsession*).

```
<data>
  <request>
  </request>
</data>
```

- Le serveur répond(*type* à *killsession*) :

```
<data>
  <response>
    <code>XXX</code>
    <message>Success</message>
  </response>
</data>
```

Les éléments *code* et *message* permettent au serveur de communiquer au proxy le résultat de l'opération.

Signature

Pour des raisons de performance, le proxy ne signe pas tous les messages OpenSST générés. Le serveur dispose d'une liste reprenant les

pages nécessitant une signature électronique. Cette liste est contenue dans un fichier qui est téléchargé par le proxy au moment où il démarre.

Cette liste se présente comme une suite:

URL, CODE

où URL est une expression régulière (utilisation de *? etc ...) et CODE est l'algorithme à utiliser pour la signature.

Idéalement, ce fichier devrait être signé.

A chaque requête HTTP interceptée, le proxy vérifie s'il y a ou non une correspondance entre l'URL demandée et une (ou plusieurs) des expressions régulières de la liste. Si la page demandée requiert une signature, le proxy demande au client (par l'intermédiaire d'un petit GUI) son profile et sa passphrase (qu'il a donnés à l'étape de génération des clés) pour pouvoir retrouver la clé privée et signer le message. Il envoie alors sa requête signée au serveur qui vérifie la signature et renvoie la page demandée. Cette technique permet de modifier la liste des URL's sans devoir toucher au code du proxy.

4.2 Transferts offline et stockage

Les systèmes de stockage consistent à stocker en local ou sur un serveur distant des fichiers.

Dans les systèmes de transferts de fichiers offline, les fichiers à envoyer sont placés dans une file d'attente et effectivement envoyés en différé (asynchrone).

Il est donc intéressant de pouvoir sécuriser ces applications qui font partie intégrante de l'e-business et du télétravail.

Le schéma ci-dessous explique les différents états d'un système offline sécurisé:

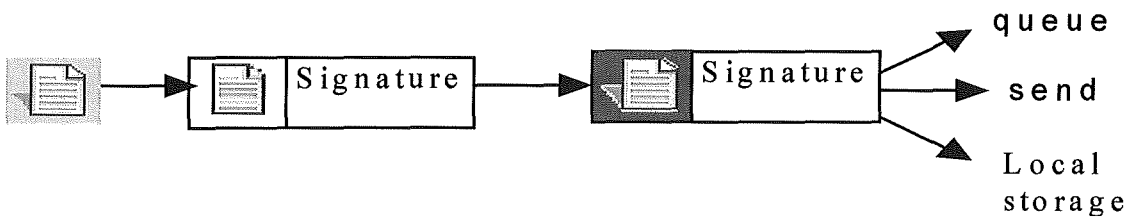

Figure 26 - Etats d'un document dans un système offline sécurisé

Le fichier peut être signé puis chiffré et, ensuite, placé dans une file d'attente, envoyé directement ou stocké localement (cas du stockage).

Le seul cas particulier est celui du queueing :

Lorsque le fichier est placé dans une file d'attente, il porte un numéro identifiant.

Ex : 62416.opensst

Un fichier 62416.ok est créé dans cette file d'attente lorsque le fichier est prêt à être envoyé.

Plus tard, une personne ou une application va déclencher l'envoi du fichier vers le destinataire (par exemple, un serveur FTP).
Si le fichier 62416.ok n'est pas présent dans la file d'attente, l'envoi n'est pas réalisé.

Une solution apportée se présente comme suit :

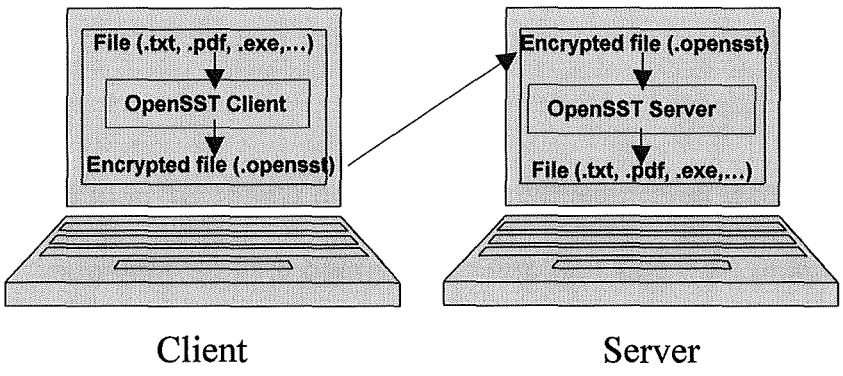

Figure 27 - Solution OpenSST pour les systèmes offline ou le stockage

Pour la simplicité, n'abordons pas le cas du queueing.

Lorsqu'un client veut envoyer un fichier au serveur, il encapsule le fichier à envoyer au format OpenSST. Ensuite, le transfert du fichier se fait, par exemple, par l'intermédiaire d'une session FTP traditionnelle. Le serveur décapsule alors le fichier, le déchiffre et vérifie la signature.

Pour le stockage, s'il se fait en local, la partie serveur peut être ignorée car tout se fait chez le client. L'opération de déchiffrement se fait

également chez le client. Si le stockage se fait sur un serveur distant, cela se déroule de la même manière qu'un système offline.

Voici un exemple de fichier chiffré et encapsulé au format OpenSST :

```
<opensst xmlns="http://www.opensst.org/protocol/message-format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opensst.org/protocol/message-
format/opensst.xsd">

<encryption type="0-2-0-8" id="dupond"
symmetricKey="key encrypted with dupond's public key"/>

<encryption type="0-2-0-8" id="smith"
symmetricKey="key encrypted with smith's public key"/>

<data type="file" encoding="base64" destination="gateway.opensst.org">
  < !-- here is the filename element and the file that are both encrypted
  -->
</data>

<signature type="1-3-1" id="dupond" encoding="base64">
  < !—signature encrypted with dupond's private key -->
</signature>

</opensst>
```

Chiffrement

Offline :

Il est possible d'implémenter la solution OpenSST pour qu'elle supporte les algorithmes désirés. Actuellement, il semble indispensable que l'algorithme AES soit supporté comme algorithme symétrique.

La clé symétrique utilisée pour le chiffrement du fichier à envoyer est générée par le client lors de chaque conversion. Elle est ensuite chiffrée avec la clé publique du serveur (chiffrement asymétrique RSA ou DSA). La clé ainsi chiffrée apparaît dans l'attribut *symmetrickey* de l'élément *encryption* du message OpenSST et l'identifiant de la clé publique utilisée se trouve dans l'attribut *id*.

Il est donc indispensable que le client dispose du certificat (ou au moins de la clé publique) du serveur. On suppose donc qu'il l'a reçu préalablement (par exemple, sur le CD d'installation du programme de conversion OpenSST).

Le serveur déchiffre la clé symétrique avec sa clé privée et peut alors déchiffrer le fichier.

Comme dans l'exemple ci-dessus, l'élément *encryption* peut apparaître plusieurs fois. En effet, il apparaîtra autant de fois que le nombre de personnes qui pourront y avoir accès en clair. La seule chose qui changera sera le chiffrement de la clé de session et donc également l'attribut *id*. Elle sera en effet chiffrée avec la clé publique de chacune de ces personnes.

Stockage :

Le client génère, comme pour le cas FTP, une clé de session avec laquelle il chiffre le fichier. Il peut ensuite faire apparaître l'élément *encryption* autant de fois qu'il y a de personnes qui pourront avoir accès au fichier en clair.

Données

Le contenu est le même, qu'il s'agisse du cas FTP ou du stockage.

La partie *data* du message contient le fichier à envoyer (chiffré ou pas) en ASCII (utilisation de l'algorithme base64 avant de chiffrer le fichier). L'attribut *type* de l'élément *data* est mis à *file*. Le nom de fichier est lui indiqué dans un élément à l'intérieur de *data*. Cet élément est défini dans un schéma XML spécifique au type du message.

S'il a été décidé de chiffrer le fichier, tout ce qui sera dans l'élément *data* sera chiffré, y compris l'élément qui contient le nom du fichier.

Signature

Pour pouvoir vérifier la signature électronique de l'expéditeur, le serveur doit posséder la clé publique de celui-ci (ou son certificat).

L'élément *signature* peut apparaître plusieurs fois dans un message OpenSST. En effet, un message peut être signé plusieurs fois par des personnes différentes (ex : messages signés successivement par plusieurs serveurs intermédiaires).

4.3 Messagerie électronique : SMTP

Le protocole SMTP (Simple Mail Transfer Protocol) est un protocole utilisé pour la messagerie électronique. La sécurisation du courrier électronique est capitale pour l'emploi de celui-ci dans des applications critiques (envoi d'informations confidentielles ainsi que de documents attachés).

OpenSST peut apporter une solution dans plusieurs configurations possibles.

1ère solution

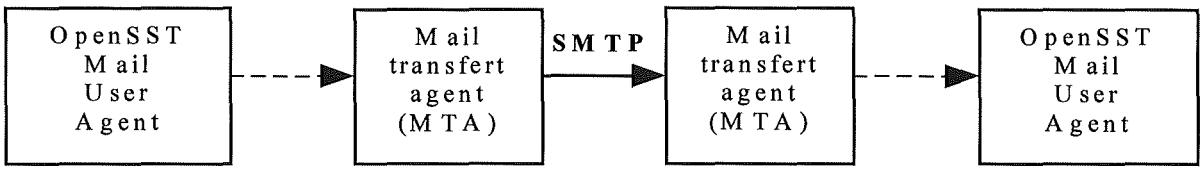

Figure 28 - Solution 1 pour la sécurisation de la messagerie électronique

Cette solution consiste à implémenter complètement un Mail User Agent OpenSST. C'est évidemment la solution qui requiert le plus de travail de développement car il faut manipuler SMTP (envoi) et IMAP/POP3 (réception). Elle a cependant l'avantage d'intégrer parfaitement la sécurité apportée par OpenSST.

2ème solution

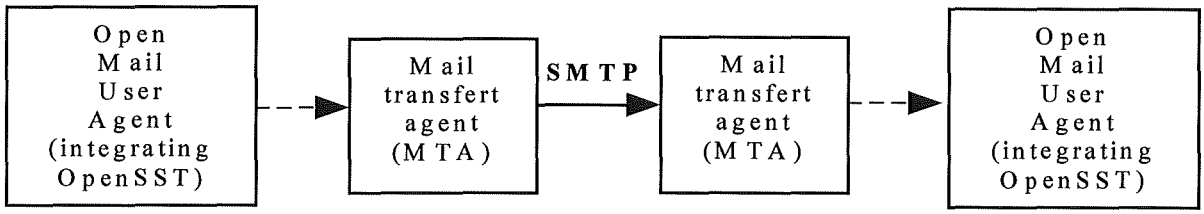

Figure 29 - Solution 2 pour la sécurisation de la messagerie électronique

Dans le cas où l'expéditeur et le destinataire utilisent un Mail User Agent non propriétaire, il est possible d'y intégrer des modules permettant de traiter les messages OpenSST. En effet, le code du logiciel est disponible et il y a donc moyen de rajouter des fonctionnalités pour prendre en charge le format de messages OpenSST. C'est la plus simple des solutions et celle qui demande le moins de développement.

3ème solution

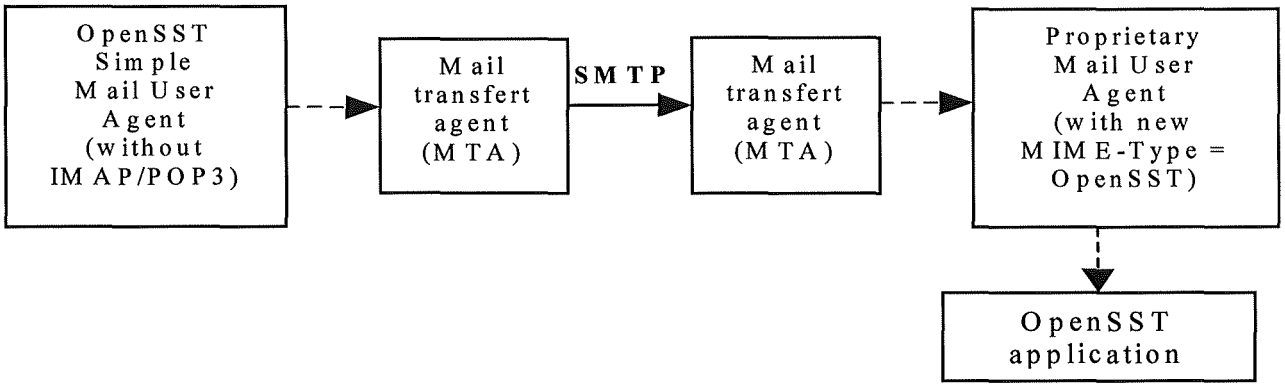

Figure 30 - Solution 3 pour la sécurisation de la messagerie électronique

Dans ce troisième cas, l'expéditeur et le destinataire disposent tous les deux d'un Mail User Agent propriétaire. Le code source n'est pas disponible, il n'est donc pas permis de le modifier pour intégrer la solution OpenSST. Il est nécessaire de contourner la solution propriétaire pour l'envoi. Pour la réception, le logiciel propriétaire peut être configuré pour qu'à la réception d'un message dont le MIME-type est *opensst*, il lance une application OpenSST qui déchiffrera le message et le fera apparaître en clair.

Remarque préliminaire

Le but n'est pas ici d'expliquer comment développer un Mail User Agent. Nous nous contenterons de reprendre les mécanismes propres à la prise en charge du format de message OpenSST.

Une étape indispensable pour les trois solutions est l'ajout d'un MIME-type particulier à OpenSST, c'est-à-dire établir la correspondance entre les fichiers dont l'extension est « *.opensst* » avec un content-type « *application/opensst* ».

Pour ce faire, sous Windows™, le registre du système d'exploitation doit être modifié. Une clé doit être créée sous HKEY_CLASSES_ROOT. Cette clé doit porter le nom « *.opensst* ». Dans cette clé, il faut ajouter une valeur (*string value*). Le nom de cette entrée est « *Content Type* » et la valeur est, par exemple, « *application/opensst* »

Déroulement

L'envoi se fait soit avec un MUA OpenSST, soit avec un MUA non propriétaire modifié pour prendre en charge l'envoi de messages OpenSST.

- L'expéditeur écrit son message de la manière traditionnelle et y attache ou non des fichiers. Il déclenche la procédure d'envoi. Un document MIME traditionnel est alors créé.

- Le document MIME est convertit au format OpenSST.

- Le fichier opensst résultant est alors placé en *attachment* d'un document MIME traditionnel qui ne comporte pas de message.

- Ensuite, le tout est envoyé avec SMTP au MTA. Voici un exemple de la partie data du protocole SMTP :

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="0-738653779-1044366904=:66087"
Content-Transfer-Encoding: 8bit

--0-738653779-1044366904=:66087
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset="iso-8859-1"
Content-Disposition: inline


--0-738653779-1044366904=:66087
Content-Type: application/opensst;
        name="10403247955410835.opensst"
Content-Disposition: attachment;
        filename="10403247955410835.opensst"
Content-Transfer-Encoding: base64

PG9wZW5zc3QgeG1sbnM9Imh0dHA6Ly93d3cub3BlbnNzdC5vcmcvcHJvdG9jb2wv
bWVzc2FnZS1m
b3JtYXQiIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hlb
WEtaW5zdGFu
Y2UiIHhzaTpzY2hlbWFMb2NhdGlvbj0iaHR0cDovL3d3dy5vcGVuc3N0Lm9yZy9wcm
90b2NvbC9t
ZXNzYWdlLWZvcm1hdC9vcGVuc3N0LnhzZCI+PGVuY3J5cHRpb24gdHlwZT0iMC0
yLTAtOCIgaWQ9
ImFkdWxhdSIgc3ltbWV0cmljS2V5PSJwYzVFMUN2Q1VZdVNGQjFzUlByQ3JYNW1
ZdFMzbHFuWUZo[.....]cCtyMzhlWUZTK0JWTGxoY1RUOEFMczZteFF5eXM4VEd
nZE9BbThWd2FpWG9DY0FXbU80R1c0T2tV
VlFzdGUySGZjMFAzRThrUEx5RS84UEZHa2s9PC9zaWduYXR1cmU+PC9vcGVuc
3N0Pgo=

--0-738653779-1044366904=:66087—
```

Pour la réception, il y a trois cas possible. Soit un MUA OpenSST, soit un MUA non-propriétaire intégrant OpenSST, soit un MUA propriétaire.

Les deux premiers cas sont semblables car on peut y apporter les modifications que l'on souhaite. Lorsque le message arrive, le content type du fichier attaché est *application/opensst*. Le MUA décapsule alors le message OpenSST et retrouve un message MIME traditionnel. Celui-ci peut donc être traité de la même manière qu'un message ordinaire.

Dans le cas de la solution propriétaire, Le message va apparaître à l'utilisateur comme étant sans message propre mais avec un fichier OpenSST attaché. Si le système a été configuré correctement (correspondance entre les fichiers qui ont une extension « *.opensst* » et l'application OpenSST), le simple fait d'ouvrir le fichier va lancer l'application OpenSST qui s'occupera alors de décapsuler le fichier pour récupérer le message électronique et l'afficher dans une petite interface très simple.

Chiffrement et signature

Le chiffrement et la signature se déroulent de la même manière que pour les systèmes de transferts offline.

Données

La partie *data* du message OpenSST contient le message MIME d'origine et son attribut *type* est mis à *mail*.

5 Avenir du protocole

La dernière date importante pour le projet était la rencontre du 18 décembre 2002. Lors de cette rencontre, le projet OpenSST a été présenté devant des représentants d'entreprises diverses couvrant un large panel de secteurs. (banques, construction, assurances, organismes publiques, privés,...)

Le but de cette rencontre était de susciter l'intérêt des personnes présentes et, par elles, de l'entreprise qu'elles représentaient, en vue de pouvoir créer des opportunités de développement du protocole OpenSST dans un environnement réel. Un autre but était également d'attirer des développeurs à rejoindre le groupe de travail OpenSST pour continuer les améliorations et le développement du format de message et du protocole. Il s'agit en effet maintenant de créer des implémentations du protocole permettant de sécuriser un plus grand nombre d'applications (à ce jour, il

n'existe que deux prototypes implémentés comme nous l'avons vu plus haut : le proxy OpenSST et l'implémentation offline). C'est maintenant que l'aspect libre du protocole OpenSST peut jouer son rôle. Des développeurs testent les implémentations existantes, proposent des améliorations,...

L'information et la prospection constituent donc l'activité principale de Aubay, Conostix et du CRP Henri Tudor dans le cadre du projet OpenSST et continueront en vue de pouvoir trouver un projet de développement et d'intégration de OpenSST dans un cas réel.

Il y a également plusieurs stages en cours ou prévus autour du développement d'OpenSST. L'un deux est l'intégration de la solution OpenSST dans une application relative à l'e-commerce développée par le CRP Henri Tudor. Un nouveau projet va également voir le jour qui traitera de la recherche, du développement, de la vérification et de la validation d'architectures open source qui procurent des services sécurisés dans le contexte de "signing server" centraux utilisés dans des environnements de collaboration mobile.

6 Références

- CRP Henri Tudor, http://www.tudor.lu .
- T. Dierks, C. Allen, *The TLS Protocol version 1.0*, Internet Engineering Task Force, January 1999.
- Dulaunoy, T. Fruru et S. Stormacq, *OpenSST Message Format*, Internet Drafts, December 2002.
- S. Stormacq, *OpenSST Message Type : HTTP proxy*, Internet Drafts, December 2002.
- A. Dulaunoy, S. Stormacq, *OpenSST : Open Simple Secure Transaction : Une approche de réduction de la complexité pour les transactions électroniques*, Janvier 2003.
- M. Pablos Martin, T. Pinxteren, P. Robert, *Sécurité et commerce électronique*, http://www.tele.ucl.ac.be/ELEC2920/2000/E-Commerce/secu_et_e-commerce.html .
- D. O'Mahony, M. Peirce, H. Tewari, *Electronic Payment Systems for E-Commerce, second edition*, Artech House, 2001.

- C. Feltus, D. Khadraoui, R. Hallez, F. Costa Pinto, *Expérimentation du protocole OpenSST pour les transactions électroniques*, CITI (Centre d'Innovation par les Technologies de l'Information) du CRPHT (Centre de Recherche Public Henri Tudor), Mars 2003.

Conclusion

Tout au long de ce mémoire, nous avons insisté sur l'importance de la sécurisation des transactions électroniques qui s'effectuent dans les environnements professionnels et privés aujourd'hui. L'exemple de la Citibank en 1995 évoqué dans l'introduction est dramatique et il n'est pas le seul, bien au contraire.

Nous avons fait remarquer qu'il existait un certain nombre de protocoles de sécurisation de ces transactions qui sont plus ou moins complets, plus ou moins coûteux et avec une complexité variable. Ces protocoles sont efficaces dans certains cas d'application et certains sont d'ailleurs considérés comme des standards de par le niveau de sécurité qu'ils apportent.

Par la suite, nous vous avons présenté le format de message et protocole OpenSST. Ce protocole est encore dans une phase de développement et nécessite encore d'être travaillé et amélioré pour qu'il puisse répondre aux exigences du marché luxembourgeois auquel il est d'abord destiné. Il lui faut acquérir de l'expérience pour bénéficier de la confiance de entreprises. En effet, installer une nouvelle solution de sécurisation est une étape très importante et très risquée et rares sont encore les entreprises qui prennent cette décision à la légère.

Le but de ce mémoire n'est bien sûr pas d'affirmer que le protocole OpenSST est meilleur que tel ou tel autre mais bien de montrer l'importance de continuer à rechercher de nouvelles solutions pour sécuriser les transactions électroniques qui deviennent de plus en plus courantes et de plus en plus nombreuses et qui touchent à des domaines ne pouvant souffrir de risques de perte de confidentialité ou d'intégrité. Nous pouvons cependant noter le manque d'un standard de sécurisation des transactions simple, peu coûteux et qui puisse s'adapter à un maximum de cas d'application (messagerie électronique, systèmes offline et de stockage, systèmes de transferts online, systèmes embarqués,...). OpenSST essaie d'être assez général pour pouvoir être adapté à ces différents cas.

Ma collaboration au projet OpenSST aura donc contribué à la phase d'information par la formalisation et l'approfondissement des contraintes et exigences et des cas d'application, pour permettre de susciter l'intérêt des sociétés luxembourgeoises.