OpenSST : Open Simple Secure Transaction

Une approche de réduction de la complexité pour les transactions électroniques

Alexandre Dulaunoy
Conostix S.A.
66, rue de Luxembourg
L-4221 Esch-sur-Alzette
adulau@conostix.com

Sébastien Stormacq
Aubay Luxembourg
51, rue de Strasbourg
L-2561 Luxembourg
s.stormacq@aubay-si.lu

Abstract

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 définitions et intéractions. 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). Ce document est une introduction à OpenSST (projet réalisé par trois acteurs au Grand-Duché de Luxembourg : Aubay SI, Conostix et le CRP-HT) et surtout, une vue globale de l'ensemble du système.

Keywords Sécurité Proxy Transaction Signature Electronique Cryptographie

Table des matières

1 Introduction

L'industrie informatique possède une panoplie importante de solutions techniques pour la sécurisation des transactions électroniques. Cette sécurisation peut se réaliser sur plusieurs niveaux (cela va du transport au message lui-même). Par exemple : SSL/TLS[1] ou SSH[2] travaillant sur les couches de transport et OpenPGP[3] sur le message lui-même. OpenSST fut créé dans le but de combler l'espace entre ces deux approches. Les standards OpenSST décrivent un format du message[5] uniforme, des formats spécifiques[1] et des échanges[2] suivant le cadre d'utilisation. Lors du développement d'OpenSST, trois contraintes[3] majeures à respecter furent découvertes :

- Sécurité
- Simplicité
- Ouverture

La sécurité est, bien entendu, un prérequis primordial à OpenSST, c'est-à-dire que le système doit satisfaire aux exigences de confidentialité, d'intégrité, d'authentification et de non-répudiation. Il est à noter que OpenSST doit aussi prendre en compte les exigences de sécurité variables suivant les contraintes extérieures. Comme par exemple, l'utilisation d'algorithmes cryptographiques plus légers dans un système limité en processeur et/ou mémoire. Cela permet d'expliquer la présence d'algorithmes cryptographiques plus faibles[4] dans le standard du format de message ainsi que la flexibilité dans le choix des méthodes de signature et de chiffrement.

La simplicité en ingénierie logicielle est aussi un facteur important pour le design d'un système informatique. D'une part, cela se vérifie avec l'approche UNIX[8] où chaque programme doit réaliser une tâche simple, et ce, de façon efficace. Bien entendu, cela entre aussi en jeu dans le cadre de la compréhension[9] d'un système global et surtout, sa réutilisation[5]. Cela permet d'expliquer la structure du

[1] OpenSST message type

[2] OpenSST exchange type

[3] Bien entendu, il existe d'autres contraintes mais celles-ci sont les prioritaires.

[4] Bien entendu, les algorithmes forts sont aussi disponibles. Ce n'est que pour les systèmes avec des contraintes spécifiques qui utilisent les plus faibles.

[5] dans le sens "*reuse*"

message (détaillée plus bas) et sa relative simplicité[6] par rapport à des standards plus complexes comme XML Encryption[11] et XML Signature[10].

L'ouverture est un des facteurs importants à l'acceptation et l'utilisation de standards. Cette ouverture permet d'accéder à un interfaçage aisé avec plusieurs transports et méthodes éprouvées. Cela permet d'expliquer la disponibilité en Logiciel Libre mais aussi l'approche ouverte de la réalisation du standard OpenSST. Le format n'est pas fermé et l'évolution même est créée par les interactions avec les développeurs et les nouvelles utilisations qui pourront être réalisées.

Il est clair qu'il n'existe pas de "silver bullet"[7] pour la sécurité des transactions électroniques. OpenSST se veut comme une solution alternative simple où des solutions complexes ou propriétaires sont trop difficiles à mettre oeuvre.

2 Format du message

La base d'OpenSST est le format de message XML utilisé pour l'encapsulation sécurisée des transactions. Le choix d'XML s'est imposé pour deux raisons majeures : d'une part, l'évolution de XML qui permet d'ajouter des éléments dans les versions supérieures sans remettre en question le parsing des anciens formats et d'une autre part, la portabilité[8] entre les différentes architectures informatiques. Le format se présente sous cette forme[9] :

```
<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 d'encryption 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[5] 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[10] ou non.

L'élément `data` ne peut apparaître qu'une fois dans un même message OpenSST. Il contient les données (souvent) encryptées. 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 (cf. plus bas dans la description des prototypes). Il existe souvent aussi deux sous-éléments à `data` : `tid` et `timestamp`. Ils sont localisés dans cet élément car l'encryption est effective 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 schema XML OpenSST principal. 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.

3 Implémentations logiciels

Dans le cadre du développement du projet OpenSST, plusieurs prototypes ont été réalisés dans le but de valider le format de message ainsi que le fonctionnement du protocole. Nous allons détailler deux cadres d'utilisation :

3.1 Proxy HTTP OpenSST (application e-banking)

Le proxy HTTP se positionne dans le contexte des applications web. Il permet de créer un tunnel fortement sécurisé entre le poste de l'utilisateur et le site web visité. La solution classique pour sécuriser un site web est l'utilisation de HTTP sur SSL, plus couramment appelé HTTP/S. HTTP/S apporte deux

[6] Dans OpenSST, la canonisation (Canonical XML) n'est pas nécessaire et il y a très peu d'élément requis dans le standard lui-même.

[7] No silver bullet : essence and accidents of software engineering, *Frederick P. Brooks, Jr.*

[8] par exemple, la portabilité des formats de fichier binaire à travers des architectures avec "endiannes" différents.

[9] Le forme présentée est un format simplifié. Nous vous invitons à lire le Schema XML de la structure de base : `http://www.opensst.org/protocol/message-format/opensst.xsd`

[10] Utilisation d'une algorithme cryptographique asymétrique pour la clé de session ou l'utilisation d'un système à clés partagées

garanties à l'utilisateur :

- il garantit l'identité du site web visité,
- il garantit la confidentialité des données échangées entre le poste client et le serveur web.

En revanche, HTTP/S ne promet rien en matière d'identification de l'utilisateur[11]. Il ne permet pas non plus de signer des requêtes.

Bien que la plupart des sites de commerces en ligne peuvent s'accommoder de ces absences, d'autres types d'applications, ne peuvent se le permettre : les extranets d'institutions financières, les applications médicales, les applications de banque en ligne[12], etc

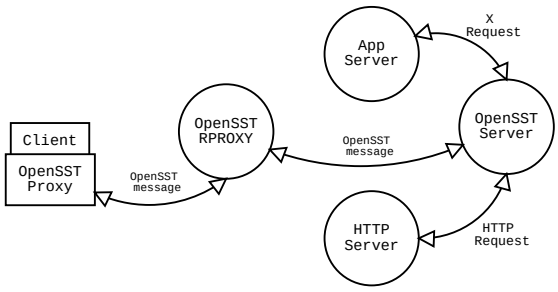
Le proxy HTTP de OpenSST a été développé en pensant aux applications financières mais peut être utile partout où HTTP/S n'apporte pas un niveau de sécurité suffisant.

Dans ce contexte, la sécurisation des applications est réalisée en installant sur le poste de l'utilisateur un programme complémentaire qui adresse les problèmes non résolus par HTTP/S : l'identification forte de l'utilisateur et la signature des transactions. Ce programme est appelé proxy car il intercepte les requêtes HTTP envoyées par le navigateur internet avant de les transférer, de manière sécurisée, vers le serveur de destination.

La solution Proxy HTTP de OpenSST est construite à partir de trois composants :

- le proxy OpenSST lui-même,
- un reverse proxy,
- le serveur OpenSST.

La figure ci-dessous positionne ces trois composants dans un architecture globale :



Le proxy OpenSST intercepte les requêtes HTTP envoyées par le navigateur internet, les encode au format OpenSST, les signe éventuellement et les envoie au serveur OpenSST.

Le serveur OpenSST, quant à lui, reçoit les requêtes envoyées par le proxy, les décode, vérifie la signature si nécessaire et, quand tout est correct, transmet la requête HTTP au serveur web de l'application.

Ce dernier reçoit donc une requête HTTP valide, et y répond. La réponse suit le même chemin dans l'autre sens.

Le processus est parfaitement transparent aux acteurs en place : ni le navigateur internet, ni le serveur web ne "savent" que la requête est encapsulée entre le moment où elle est émise par le premier et le moment où elle est reçue par le second. Cette solution permet donc de sécuriser des applications web sans qu'elles aient été conçues spécifiquement pour cela.

Le reverse proxy joue un 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 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 apparaître dans plusieurs circonstancesă :

- 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 incorrect ou une erreur logicielle,
- un erreur de transfert.

3.1.1 Détail du protocole

Pour réaliser cette encapsulation de requêtes HTTP et échanger les messages de manière sécurisée, le proxy HTTP de OpenSST définit un ensemble de messages, i.e. de contenus de l'élément `data` des messages OpenSST que nous proposons de survoler ci-dessous[13].

Création d'une session La première tâche d'un proxy OpenSST est de créer une session sécurisée avec son serveur. La création de session se fait en deux étapes. Dans un premier temps, le proxy envoie un message de demande de création de session au serveur `publickey-request`. Ce message contient un minimum de données qui permet au serveur d'identifier sommairement le proxy et la future session. Le serveur répond au proxy en lui envoyant sa clé publique

Dans un second temps, le proxy génère une clé de session. Cette clé de session est encodée avec la clé publique du serveur de manière à pouvoir être échangée en toute sécurité dans le message

[11] SSL intègre une solution d'identification forte du client (via X.509) mais son utilisation dans le cadre de HTTP/S est complexe à mettre en oeuvre, différemment supportée par les navigateurs Internet et, en pratique, rarement utilisée pour une application généraliste, destinée au grand public.

[12] e-banking

[13] Notons que seuls les éléments clés sont décrits dans ce papier, la description complète et exhaustive des formats des messages du proxy sont décrits dans [6]

`sessionkey-request`.

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

Paire de clés proxy La toute première fois que le proxy démarre, il doit générer une paire de clés - qui lui serviront à signer certaines requêtes - et communiquer cette paire de clés au serveur. Cette opération est supportée par le message `init-request`.

Le serveur va donc recevoir une clé publique. Il est nécessaire qu'il puisse associer cette clé avec un utilisateur dûment identifié de son côté : c'est le rôle des éléments `token` et `otp`[14].

L'utilisateur qui souhaite entrer en communication avec une organisation reçoit de manière sécurisée[15] de la part de celle-ci un nom d'utilisateur et un code PIN. Ces éléments permettront permettront d'identifier le client de manière sécurisée lors de l'initialisation du proxy.
Le serveur répond avec un message `init-response` ainsi le serveur peut communiquer le résultat de l'opération au proxy.

Authentification de l'utilisateur Si l'acceptation de la clé publique de l'utilisateur ne se fait qu'une seule fois dans la vie du proxy[16], il est en revanche nécessaire d'authentifier l'utilisateur à chaque ouverture de session.

L'authentification de l'utilisateur est faite de manière très simple : en envoyant un message quasiment vide mais signé. Le serveur, en vérifiant la validité de la signature, pourra authentifier l'utilisateur. Pour authentifier l'utilisateur, le proxy envoie un message `auth-request`.
Les éléments `code` et `messsage` permettent au serveur de communiquer au proxy le résultat de l'opération.

Echange des données 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 HTTP. Les messages envoyés du proxy vers le serveur s'appellent `tx-request`. A chaque requête HTTP générée par le navigateur internet, une paire de messages `tx-request`, `tx-response` sont échangés entre le proxy et le serveur. Une requête pour une simple page web peut générer plusieurs requêtes sous-jacentes pour obtenir les images, les feuilles de styles,...

Fermeture d'une session Quand l'utilisateur décide de terminer sa session le proxy envoie un message `logoff-request`. Ce message ne contient aucun élément utile. Il signifie juste au serveur l'intention de l'utilisateur. Les éléments `code` et `message` permettent au serveur de communiquer au proxy le résultat de l'opération.

Signature des transactions Pour des questions de performance, le proxy ne peut s'offrir le luxe de signer tous les messages OpenSST qu'il génère. Il doit donc pouvoir décider, de manière dynamique, quels messages il est nécessaire de signer et lesquels peuvent être envoyés sans signature. Pour ce faire, le proxy utilise un liste des URLs (sous forme d'expressions régulières) qu'il est nécessaire de signer. Un exemple d'un tel fichier pourrait être :

```
http\://localhost\:8080/manager/.*      SHA-1/RSA
http\://localhost\:8080/admin/index.jsp SHA-1/RSA
```

La partie de gauche est une expression régulière, la partie de droite est le nom de l'algorithme qui doit être utilisé pour la signature. Quand le proxy crée un message OpenSST, il vérifie toujours si l'une des expressions régulières correspond à l'URL à laquelle la requête est destinée. Dans l'affirmative, il ajoute une signature au message.

Le proxy ne s'arrête pas à la première correspondance, il vérifie systématiquement toute la liste des expressions régulières. Ceci permet, dans le cas où plusieurs expressions correspondent à l'URL destinatrice, de créer des messages avec multiples signatures. Cette technique est non intrusive, elle permet de modifier la liste des URLs qui nécessitent une signature sans toucher au code du proxy.

3.2 Offline (système embarqué)

Les systèmes embarqués possèdent, par essence, des capacités reduites en terme de capacité mémoire, puissance processeur et de possibilités réseaux. Par exemple, la "stack" TCP/IP de ces systèmes est souvent limitée ou il ne supporte pas complétement TCP. Dans ce cadre, OpenSST apporte des approches complémentaires pour éviter ces limitations et/ou travailler avec des ressources limitées.

3.2.1 Contrôle à distance

Dans le cadre des systèmes embarqués, la gestion à distance est une des fonctionnalités primordiales. Ces systèmes disposent de plus en plus de fonctionnalités d'interconnection réseau et des possibilités de gestion à distance. La gestion distante est souvent très variable suivant le type d'application utilisant le système dédié. Nous avons résolu ce problème en utilisant un prototype "offline" pour les messages OpenSST, c'est-à-dire qu'il est possible d'avoir des solutions avec différentes approches de fonctionnement et utilisant un sous-ensemble minimal à OpenSST.

[14] One Time Password

[15] Par courrier sécurisé, par retrait à un guichet, etc ...

[16] L'acceptation de clé se fait également à l'occasion d'un renouvellement de clé en cas d'expiration, de perte ou de compromission de la première paire de clés.

Encapsulation Comme les systèmes embarqués n'utilisent pas les mêmes systèmes de commande, ils sont souvent incompatibles entre eux. D'une part, les commandes sont souvent des formats et structures variables et d'une autre part les méthodes de transports sont aussi variables. Pour utiliser OpenSST dans ce cadre, un Internet-Draft[7] fut réalisé pour ces extensions.

Ce draft définit deux parties importantes :

- Les requêtes dans la partie `data` avec un type dédié : `embedded-command`,
- un traitement déconnecté[7] des messages OpenSST.

Sécurité La sécurité des messages OpenSST pour le contrôle à distance des systèmes embarqués est un équilibre entre les ressources de ces systèmes et la sécurité de l'ensemble. D'une part pour les systèmes limités en ressource, nous utilisons une signature cryptographique de type HMAC pour la signature des messages. La signature est construite grâce à une clé partagée entre les systèmes et un assemblage contenant la séquence du message[17], le type et le hash de l'ensemble `data`. Cela permet d'obtenir un niveau de sécurité assez élevé sans pour autant utiliser des ressources importantes si l'utilisation de `encryption` avait été complète. D'une autre part, la sécurité peut être complète en utilisant l'ensemble du standard OpenSST sur des systèmes embarqués plus performants. Dans les deux cas, les messages peuvent rester compatibles et partager les mêmes infrastructures et développements logiciels. Ce qui est avantage important pour les infrastructures hétérogènes.

4 Conclusion

Le protocole OpenSST est encore assez jeune mais il a déjà montré sa grande versatilité. Les extensions futures et les approches de recherches complémentaires sont importantes. Nous espérons pouvoir compter sur plusieurs projets de recherche et d'applications industrielles utilisant ce protocole et en faissant évoluer l'approche d'OpenSST et ses différents développements en Logiciel Libre.

5 Informations

Des informations complémentaires (ainsi que le code source disponible des différentes prototypes en Java et Perl (sous licence GNU General Public License)) peuvent être trouvées sur :

`http://www.opensst.org/`

Si vous désirez participer aux développements futurs de OpenSST, n'hésitez pas à contacter les auteurs.

[17] Pour éviter les attaques de rejeu avec des messages UDP, par exemple.

Références

[1] T. Dierks, C. Allen, The TLS Protocol version 1.0, Internet Engineering Task Force, January 1999.

[2] T. Ylonen, T. Kivinen, SSH Transport Layer Protocol version 1.5, Internet Drafts, SECSH Working Group, September 20, 2002.

[3] J. Callas, L. Donnerhacke, OpenPGP Message Format, Internet Engineering Task Force, November 1998.

[4] Jr. Frederick P. Brooks. No silver bullet : essence and accidents of software engineering. Computer, 20(4) :10–19, 1987.

[5] A. Dulaunoy, T. Fruru et S. Stormacq, OpenSST Message Format, Internet Drafts, December 2002.

[6] S. Stormacq, OpenSST Message Type : HTTP proxy, Internet Drafts, December 2002.

[7] A. Dulaunoy, OpenSST Message Type : Offline processing, Internet Drafts, December 2002

[8] D. M. Ritchie, K. Thompson, The UNIX Time-Sharing System, The Bell System Technical Journal, 1978.

[9] N. Bezroukov, Why is simplicity so important for the OSS movement ?, `http://www.softpanorama.org/OSS/Bla_faq/the_value_of_simplicity.shtml`, 2001.

[10] M. Bartel and others, XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002.

[11] Takeshi Imamura and others, XML Encryption Syntax and Processing, W3C Recommendation, 10 December 2002.

[12] R. Hallez, OpenSST : Fiche Technique - HTTP, Publication CRP - AccessPME (FNR), Décembre 2002.