Face Key

Quentin GERARD, Louis L'HARIDON et Matthieu VILAIN
CMI Systèmes Intelligents et Communicants, Université de Cergy-Pontoise
mail@etu.u-cergy.fr

Résumé

abstract of the project

I. Sécurité

A. Introduction au problème

Lors de l'utilisation de l'application, beaucoup de données transitent entre les différents blocs que composent l'architecture du logiciel, et/ou sont stockés dans le Base de Donnée. Parmis elles, se trouvent des informations extrêmement sensibles concernant les utilisateurs. Pour un utilisateur, nous pouvons y trouver ces informations (liste non ordonnée et non exhaustive) :

- Adresses mails
- Mot de passe (de l'application Face Key)
- Combinaisons Identifiants/Mot de passe de plusieurs sites
- Coordonnées Bancaires
- Et d'autres

Si un utilisateur mal intentionné arriverait à intercepter les messages entre le client et le serveur, en se plaçant entre les deux grâce à un attaque basique de type Man in the Middle (MITM), alors il aurait accès à toutes ces données. De même si la base de donnée arrivait un jour à être compromise, et accessible par un utilisateur mal intentionné, alors il aurait également accès à toutes les données. Il nous faut donc trouver une solution qui nous permet à la fois de sécuriser les envoie de données Client/Server, et une solution qui nous permet de rendre illisible les informations dans la base de données à tous les utilisateurs autres que l'utilisateur concerné.

B. RSA

1) Introduction: Le RSA est un algorithme de cryptographie dit "asymétrique" (une clé privé et une clé publique, contrairement à la cryptographie symétrique qui utilise la même clé pour crypter et décrypter). Il a été rendu public par R.L. Rivest, A. Shamir, et L. Adleman dans un papier publié en 1977 appelé "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*". Il a pour but, à la fois de crypter des messages afin qu'ils puissent être transmis sans risquer d'être lu par une personne tierce, mais aussi de signer les messages. Un des principe du RSA est que la clé privé, permettant de décrypter les messages soit impossible à retrouver à partir de la clé publique, qui elle permet d'encrypter les messages. Nous avons donc décidé d'utiliser cette méthode afin de crypter les messages entre le Client et le Server.

2) Génération des clés: La génération des clés appartient au receveur des messages, il doit ensuite communiquer la clé publique, qui n'est pas une information sensible, à l'envoyeur du message. Dans notre cas, le serveur et le client vont chacun générer de leur coté une pair de clés et transmettre à l'autre partie, la clé publique. Ainsi, même si les clés sont interceptés, n'importe qui pourra envoyer des messages au client et au serveur, mais personnes ne pourra lire les messages crypter envoyé, à l'exception du client et du serveur.

Pour créer les clés nous devons d'abord choisir deux nombres premiers p et q et ensuite calculer

$$n = p.q$$

Notons que n sera présent dans la clé publique il est donc important de trouver deux nombres premiers p et q assez grand pour qu'il soit difficile de décomposer n (Il est recommandé d'après les auteurs d'utiliser un nombre premiers composer de 100 chiffres), en effet chaque nombre n à une décomposition **unique** en facteur premiers. On cherche ensuite d tel que :

$$pgcd(d, (p-1).(q-1)) = 1$$

Nous venons donc de créer notre clé privé (d, n), pour trouver notre clé publique nous devons calculer e tel que :

$$e.d \equiv (mod(p-1).(q-1))$$

Notre clé publique est donc le couple (e, n).

3) Cryptage et Décryptage des messages: Pour crypter un message M nous avons besoin de notre clé publique (e, n), notre message crypter C se calcule de la manière suivante :

$$C \equiv M^e (mod\ n)$$

La fonction de décryptage est tout aussi simple, à partir de notre message crypter C , et de notre clé privé (d, n), nous calculons :

$$M \equiv C^d (mod\ n)$$

La méthode décrite içi est démontrée mathématiquement dans le papier d'origine de R.L. Rivest, A. Shamir, et L. Adleman.

4) Implémentation: Pour implémenter l'algorithme nous avons choisis d'utiliser la bibliothèque libre et open source OpenSSL, et plus particulièrement la bibliothèque *libcrypto* qui fourni les algorithmes de cryptographie. OpenSSL est une bibliothèque largement répandu et utilisé, il est donc plutôt aisé de trouver de la documentation dessus (suivant les fonctionnalités utilisées).

a) Génération des clés: Les clés publique et privés du serveur et du client sont généré préalablement, et stocké les les fichiers des deux programmes.

b) Échange des clés: Lors de chaque connexion du client au serveur, les clés publiques les deux parties sont échangé, envoyé sous forme de fichier "*.pem*" et stocké, avec les autres clés pour le client, et dans un dossier nommé par le PID du processus pour le serveur (le serveur étant multi-processus).

c) Cryptage et décryptage: Pour le cryptage et le décryptage, les fonctions de OpenSSL sont utilisées après avoir charger les clef dans la mémoire. Le cryptage et le décryptage est opérationnel pour peu que les deux clés issus de la même pair soit utilisées pour crypter et décrypter.

d) État de l'implémentation: Le système n'est que partiellement implémenté, en effet, malgré le bon fonctionnement de chacune des briques indépendamment des autres, nous pouvons observer un nombre non négligeable de raté lorsque nous essayons d'utiliser l'ensemble dans le contexte Client/Serveur. En effet, la transmission du message crypté ne se fait pas correctement (contrairement aux messages non crypté), il apparait effectivement que la somme de vérification du message crypté de l'une des partie, n'est pas la même que la somme de vérification du même message mais de l'autre côté. Nous pouvons en déduire que le message s'est mal transmis, cependant nous n'avons pas encore trouvé la raison de ce phénomène.