

Smart Social Network

Rapport de projet SSI

27 février 2013

Table des matières

1	Introduction	1
2	Fonctionnement global	2
2.1	Première utilisation	2
2.2	Chiffrement	2
2.3	Déchiffrement	2
3	SmartCard	2
3.1	Génération de nombres aléatoires	2
3.2	Chiffrement/Déchiffrement	3
3.3	Signature/Vérification	3
3.4	Code PIN/PUK	3
3.5	SoftCard	4
3.6	Difficultés rencontrées	4
3.7	Améliorations possibles	4
4	Secure Social Network	6
4.1	FaceCrypt	6
4.1.1	Cryptographie	6
4.1.2	Communications sécurisées	6
4.1.3	Base de données	6
4.2	SSNExt	7
4.3	Difficultés rencontrées	7
4.4	Améliorations possibles	7
A	Fonctionnement détaillé du tunnel entre la carte	8
A.1	Objectifs	8
A.2	Choix techniques	8
A.3	Etablissement du tunnel	8
A.4	Communications dans le tunnel	9

1 Introduction

De nos jours, les réseaux sociaux ont pris une grande ampleur sur internet et accueillent chaque jour de plus en plus d'adhérents. Le principe consiste à y créer un profil, y insérer des données que l'utilisateur désire partagées, telles que des photos, des vidéos, des messages, etc. Sur ce réseau, des « amis » seront ajoutés : ils pourront alors accéder à ces informations.

La problématique soulevée était de limiter la diffusion des informations à certaines personnes dans nos « amis », mais surtout limiter la diffusion d'informations vis à vis du réseau social lui-même. En effet, lorsque nous partageons une donnée, celui-ci détient cette information qu'elle soit définie comme privée ou non.

L'idée de ce projet était alors de limiter cette fuite d'information, afin de garantir la confidentialité des données des utilisateurs et ce, renforcé par une authentification forte. Nous nous concentrerons sur le réseau social Facebook puisqu'il est le plus utilisé.

La concrétisation du projet s'est traduite par le développement d'une extension pour le logiciel Mozilla Firefox permettant à l'utilisateur de gérer le chiffrement et le déchiffrement de ses données sur le réseau social Facebook, les traitements lourds étant confiés à une application Java.

Concernant l'authentification forte, nous avons utilisé des cartes à puce de type Java Card J3A (marque NXP) avec 40 Kilo-octets (Ko) d'EEPROM, via des lecteurs Omnikey 3121.

L'intérêt du projet était également d'analyser la sécurité de ces cartes à puce, à savoir la génération de nombres aléatoires, de clefs (symétriques et asymétriques), chiffrement, déchiffrement et signature. C'est cette même carte à puce qui contiendra à posteriori les données sensibles de l'utilisateur comme son identifiant, son mot de passe, sa clef privée... Le dialogue avec la carte se fait par l'intermédiaire d'un client Java : « SoftCard ».

Initialement prévu comme deux projets différents, un par groupe, il s'est avéré que nous travaillerions conjointement pour se concentrer sur un unique projet regroupant :

- l'étude et la mise en œuvre de solutions d'authentifications et de signatures par cartes à puce, proposé par Magali BARDET ;
- les solutions cryptographiques pour les réseaux sociaux, proposé par Ayoub OTMANI ;

2 Fonctionnement global

Nous allons dans cette partie décrire, dans les grandes lignes, le fonctionnement global du projet.

2.1 Première utilisation

Lorsque l'utilisateur installe l'extension et désire se connecter par la suite à Facebook, une « pop-up » sera affichée, lui demandant ainsi ses identifiants. A cet instant, il pourra décider de générer un nouveau mot de passe – aléatoire – par la carte. Toutes ces informations seront alors stockées sur la carte et il pourra chiffrer des messages destinés à une liste d'amis qu'il aura remplie.

2.2 Chiffrement

Les chiffrements s'effectuent en plusieurs temps. Tout d'abord, l'utilisateur entre son message, clique sur le bouton « chiffrer », sélectionne la ou les listes d'amis concernées et choisit deux modes : l'un garantit l'anonymat mais nécessite plus de ressources, l'autre a les propriétés inverses.

2.3 Déchiffrement

Pour le déchiffrement, deux cas se présentent. Si la personne ayant chiffré le message avait choisi le mode « anonyme », un application tentera de déchiffrer chacune des lignes composant le contenu chiffré avec la clef privée de l'utilisateur, au préalable obtenue en interrogeant la carte.

Ces tentatives s'arrêteront dès qu'aucune erreur n'aura été déclenchée, c'est-à-dire lorsque la clef privée utilisée correspondra à la clef publique ayant servi à chiffrer la clef symétrique. Cette dernière sera alors utilisée pour déchiffrer le message posté sur Facebook.

Dans le cas où le mode « non-anonyme » avait été sélectionné lors du chiffrement, l'algorithme se contentera de déchiffrer la partie chiffrée associée à l'identifiant de l'utilisateur courant.

3 SmartCard

Aujourd'hui nous sommes tous menés à utiliser les cartes à puce comme les cartes bancaires, les cartes vitales. Elles sont notamment utilisées pour effectuer de l'authentification forte et pour contenir des informations confidentielles.

Dans cette partie du projet, nous détaillerons notre étude des solutions cryptographiques pouvant permettre l'authentification ou la signature, puis de mettre à profit ces caractéristiques pour la confidentialité liée à Facebook.

3.1 Génération de nombres aléatoires

La carte à puce permet de générer des nombres aléatoires, utiles dans la création d'IV (Initialization Vector), de mots de passe, de clefs, etc. Ce générateur peut donc être considéré comme un point crucial dans la sécurité de l'application. C'est pour cette raison qu'il a fallu nous assurer que les résultats suivent une distribution uniforme.

La librairie Javacard d'Oracle met à notre disposition deux moteurs de génération de nombres aléatoires : l'un est un algorithme pseudo aléatoire, l'autre cryptographiquement sûr.

Bien entendu, pour notre projet, nous avons utilisé l'algorithme décrit comme « cryptographiquement sûr ». Cependant, par acquis de conscience, nous avons voulu vérifier le niveau de l'aléatoire du générateur dit sûr à l'aide d'un outil permettant de réaliser une analyse statistique dont le compte rendu apparaît ci-dessous :

```
*****
*                                     *
*                               CR de Yicheng                               *
*                                     *
*****
```

3.2 Chiffrement/Déchiffrement

Dans l'étude des cartes à puce, nous avons également utilisé des méthodes de chiffrement et de déchiffrement symétriques et asymétriques. Pour notre cas d'utilisation, nous avons choisi des clefs RSA de 1024 bits pour l'algorithme de chiffrement asymétrique RSA-PKCS1. Quant aux opérations cryptographiques symétriques, des clefs AES de 128 bits ont été générées et utilisées. Ces clefs et algorithmes ont été choisis pour leur sûreté.

Nous avons testé ces algorithmes en chiffrant et en déchiffrant, à l'aide de clefs préalablement générées par la carte, divers messages.

Ici, nous avons utilisé ces algorithmes dans différents cas. Concernant le chiffrement par clef publique, bien qu'implanté, nous n'en avons pas eu besoin : c'est en effet Facecrypt, de la seconde partie du projet, qui s'en chargera, la clef publique lui ayant été fournie. Quant au cryptosystème symétrique, nous l'avons utilisé lors de la communication avec SoftCard via un « tunnel » sécurisé par AES-128. Ce dernier a pour objectif d'apporter confidentialité, intégrité et authentification aux données échangées entre la carte et SoftCard.

3.3 Signature/Vérification

Nous avons étudié un autre cas où le cryptosystème asymétrique peut être utilisé via la carte à puce : la signature et la vérification de données. Grâce à la signature, nous pouvons obtenir authentification et non-répudiation, puisque la signature se fait via la clef privée de l'émetteur. Une méthode de vérification existe également permettant de vérifier l'authenticité des données via la clef publique du destinataire. Un booléen nous est alors retourné selon la réponse.

3.4 Code PIN/PUK

Une carte à puce étant associée à un utilisateur, il va de soi que ce dernier dispose d'un secret pour pouvoir la carte protéger. Il existe pour cela un code PIN, affecté à chaque carte, et dont seul l'utilisateur a connaissance.

Le code PIN est défini sur deux octets, ce qui représente $2^{16} = 65536$ solutions. Ceci est relativement faible, notamment contre une attaque de type « bruteforce », mais elle est contrée ici via à un nombre d'essais limité. Dans notre cas nous limitons le nombre d'essais à trois. En cas de blocage de la carte, dû à un nombre de tentatives trop élevé, seul le code PUK pourra débloquent la carte. Tout comme le code PIN, le nombre d'essais pour entrer le code PUK est de trois. Si ce nombre est dépassé la carte devient inutilisable et la réinstallation des applets est alors la seule option pour la rendre de nouveau opérationnelle.

Durant l'exécution de l'application, dès qu'une fonctionnalité sensible de la carte sera sollicitée, le code PIN de l'utilisateur lui sera demandé, bloquant ainsi temporairement l'accès à la carte. Au final, un déchiffrement, une signature et une modification des identifiants engendreront une telle interrogation.

3.5 SoftCard

Pour permettre le dialogue avec la carte à puce, il a été nécessaire de développer une application tierce : SoftCard. Comme nous l'avons mentionné précédemment, la partie SSN avait besoin de certaines opérations ou de certaines données. C'est pourquoi cette application devait aussi servir d'intermédiaire entre FaceCrypt, détaillé plus loin, et Smartcard.

Etant donné que nous disposions déjà d'un environnement de développement en Java pour les applets de SmartCard, SoftCard a été développé dans ce même langage. Il permettait en outre de disposer des mêmes API que FaceCrypt.

Ainsi, SoftCard a été pensé comme un serveur : pour chaque requête reçue de FaceCrypt, une action est déclenchée. Celle-ci est traitée puis transmise à SmartCard. Comme nous l'avons expliqué dans la partie 3.3, s'il s'agit d'une opération « sensible », le code PIN est demandé à l'utilisateur.

Au final, les opérations supportées par SoftCard sont les suivantes :

- génération d'un nombre aléatoire ;
- obtention de la clef publique
- déchiffrement de données ;
- signature de données ;
- enregistrement, modification et récupération des identifiants Facebook.

Afin de garantir la sécurité des communications entre SoftCard et SmartCard, nous avons aussi implanté un tunnel entre ces deux entités. Pour plus d'informations, l'annexe A décrit notre raisonnement et son fonctionnement.

3.6 Difficultés rencontrées

Une première catégorie de difficulté rencontrée était l'utilisation des fonctions cryptographiques sur SoftCard et SmartCard.

Tout d'abord, malgré une bonne documentation JavaCard, certaines cartes n'implémentent pas toutes les méthodes. Il nous a donc fallu nous assurer qu'elles existaient réellement, en testant parfois « à l'aveuglette ». En outre, nous avons dû faire de telle sorte que FaceCrypt et SoftCard utilisent les mêmes algorithmes cryptographiques, notamment au niveau des cryptosystèmes asymétriques.

Une seconde source de problèmes est provenue du fait que les informations que nous ne pouvions pas envoyer à la carte sont de taille limitée.

3.7 Améliorations possibles

Actuellement certaines améliorations sont possibles, notamment au niveau de la gestion du code PIN et de celle de l'arrachage de la carte. En effet, le code PIN n'est demandé que lorsque des informations sensibles sont réquisitionnées auprès de la carte. On pourrait ainsi imaginer un déblocage de la carte dès sa connexion au terminal. En outre, si aucune erreur n'est gérée, lorsqu'une requête est envoyée à la carte après qu'elle ait subi une reconnexion physique « à chaud », une erreur sera générée. Dans notre cas, étant donné le temps qu'il nous restait lorsque nous nous sommes penchés sur le problème, nous avons ajouté une surveillance d'une levée d'exception, en forçant une reconnexion à la carte. Si à ce moment la carte n'est toujours pas insérée, une erreur est soulevée, terminant ainsi le processus auquel s'est connecté FaceCrypt. Une amélioration serait donc de se mettre en attente d'une connection correcte avec la carte.

4 Secure Social Network

Ce projet avait initialement pour objectif d'étudier les différents procédés cryptographiques que nous pouvions utiliser pour sécuriser la vie privée des utilisateurs vis-à-vis d'un réseau social. C'est Facebook qui a été choisi car plus pertinent étant donné son ampleur.

Après la fusion des deux projets, il a été jugé intéressant d'utiliser la carte à puce comme outils d'authentification forte pour réaliser des opérations comme la génération de nombres aléatoires, le déchiffrement avec la clef privée par exemple.

Pour les autres besoins de chiffrement moins sensibles – comme chiffrer en utilisant un algorithme symétrique et une clef générée par la carte, ou chiffrer avec une clef publique – nous avons développé une application Java « FaceCrypt » qui permet de réaliser les opérations de chiffrement plus rapidement que la carte.

Afin de proposer une solution de chiffrement au sein de Facebook, nous avons aussi conçu une extension pour *Mozilla Firefox*, SSNExt, servant d'interface entre ce réseau social et l'utilisateur.

4.1 FaceCrypt

Comme mentionné précédemment, FaceCrypt est une application gérant une partie du chiffrement du projet et faisant le relais entre l'extension Firefox et le composant SoftCard.

Elle agit comme un serveur pour l'extension et comme un client pour SoftCard, c'est à dire que SSNExt va envoyer des requêtes à FaceCrypt qui va pour sa part traiter ces requêtes et les envoyer à SoftCard si besoin.

FaceCrypt fonctionne donc comme un démon et ne dispose donc pas d'interface graphique, jugée inutile. Ce composant a été écrit en Java dans un premier temps afin de maximiser l'interopérabilité avec SoftCard (lui aussi écrit Java). En effet, nous pensions pouvoir disposer des mêmes algorithmes que sur la carte. De plus, ce langage est le seul, avec le C, que tout le groupe maîtrisait. Nous l'avons préféré au C pour les raisons citées ci-dessus mais aussi pour le fait qu'il soit plus haut niveau et orienté objet.

FaceCrypt est divisé en trois parties :

4.1.1 Cryptographie

Une partie majeure de FaceCrypt consiste en un certain nombre de modules permettant d'utiliser la cryptographie symétrique, asymétrique et des fonctions de hachage. Plusieurs algorithmes peuvent être utilisés mais pour ce projet, notre choix s'est porté sur AES (CBC avec une clef de 256) et RSA, avec des clefs de 1024 bits.

4.1.2 Communications sécurisées

4.1.3 Base de données

Pour gérer ses amis pouvant déchiffrer les messages qu'il poste, un utilisateur peut les placer dans des listes. Celles-ci sont gérées par FaceCrypt et l'extension *via* une base de données *sqlite*¹.

FaceCrypt gère une base (donc un fichier) pour chaque utilisateur. Une base contient trois tables permettant de lier des amis à une liste, la table des amis contient aussi leur clef publique.

1. Il s'agit d'une base de données contenue dans un seul fichier pouvant être utilisé dans des programmes sans avoir à embarquer une base de données traditionnelle suivant un modèle client-serveur

4.2 SSNExt

4.3 Difficultés rencontrées

Lors du développement de ce sous-projet, nous avons rencontré des difficultés à de multiples endroits :

- les manipulations de la page Facebook : un certain nombre d'éléments que nous pensions triviaux à réaliser nous a finalement posé beaucoup de problèmes, ceci dû au fait de la minutie des développeurs de Facebook à empêcher les utilisateurs de « scripter » leurs actions sur le réseau social.
- les communications sécurisées entre FaceCrypt et SSNExt.

4.4 Améliorations possibles

A Fonctionnement détaillé du tunnel entre la carte

A.1 Objectifs

Le tunnel entre la carte et le logiciel qui contrôle le lecteur sert à protéger les communications avec la carte. Contrairement aux autres liens entre logiciels, il n'a pas été possible d'utiliser un protocole de sécurisation tel que TLS car la carte ne dispose pas de pile TCP/IP. En implanter une étant hors de notre domaine de compétences, nous avons utilisé les connaissances acquises en cours en cryptographie pour ajouter une couche de sécurité sur la liaison de données déjà présente.

Les objectifs cryptographiques réalisés par le tunnel sont les suivants :

Confidentialité : Les données ne peuvent pas être lues par une personne non autorisée.

Intégrité : Les données n'ont pas été modifiées durant leur transport.

Authentification : Les données ont été envoyées par une entité qui connaît le secret.

Les deux derniers objectifs sont réalisables conjointement sans diminuer la sécurité du système alors que le premier nécessite une clé séparée [RGS ANSSI, section 2.5.1]. Dans la suite, le second objectif désigne à la fois l'intégrité et l'authentification.

A.2 Choix techniques

Le choix d'AES nous a paru être le plus pertinent car c'est l'algorithme de référence. Il a remporté le concours lancé par le NIST après de nombreuses études par la communauté. Il a été approuvé par la NSA avec des clés de 128 bits pour protéger des données classifiées au niveau SECRET dans [NSA]. L'annexe B1 du RGS publié par l'ANSSI affirme également que cette longueur est satisfaisante [RGS ANSSI].

Nous avons utilisé AES avec des clés (différentes) de 128 bits pour les deux objectifs. Le mode CBC permettant de faire du chiffrement et de l'authentification-intégrité, nous l'avons utilisé pour les deux. Pour conjuguer performance et sécurité, nous avons implanté une version modifiée de CBC-MAC qui intègre la taille du message en début de message. Cette modification garantit la sécurité lorsque les messages ont une taille variable, comme indiqué dans [CBC-MAC].

L'établissement du tunnel se fait selon un protocole challenge-réponse réciproque pour assurer l'authentification mutuelle de la carte et du logiciel avec laquelle elle communique. Une clé de session est envoyée chiffrée par la carte pour assurer la confidentialité des données.

A.3 Etablissement du tunnel

L'authentification mutuelle peut être résumée comme ceci :

- génération d'un nonce client ;
- envoi du nonce client à la carte ;
- la carte récupère le nonce client, génère un nonce carte, un IV et une clé de session ;
- la carte envoie la clé, le nonce client et le nonce carte chiffrés avec la clé partagée et l'IV ;
- le client extrait l'IV et déchiffre le message avec la clé partagée ;
- le client vérifie si le nonce client est présent pour authentifier la carte, extrait la clé de session et récupère le nonce carte ;
- le client génère un IV et renvoie le nonce carte avec la clé de session établie ;
- la carte récupère le nonce carte en déchiffrant le message reçu avec la clé de session et vérifie s'il est identique à celui envoyé pour authentifier le client.

C'est le constructeur de l'objet Java « Tunnel » qui s'occupe de tout, côté client.

A.4 Communications dans le tunnel

Une fois les entités mutuellement authentifiées et la clé de session échangée, les communications dans le tunnel peuvent se faire.

L'objet « Tunnel » présente une fonction d'envoi semblable à celle présente pour les envois sans tunnel. Il se charge de l'envoi vers la carte en découpant préalablement le message en fragments de taille inférieure à la capacité du lien.

Cet objet présente également une fonction d'exécution qui permet de lancer les fonctions présentes sur la carte selon les paramètres présents dans les données.

Le format général à l'entrée du tunnel est celui-ci :

AID	Instruction	Paramètre 1	Paramètre 2	Longueur des données	Données
-----	-------------	-------------	-------------	----------------------	---------

Ce format est identique à celui des APDU Javacard, la seule différence étant le rôle du premier octet qui nous sert à différencier les applets. Ce fonctionnement est inspiré du modèle TCP/IP dans lequel le numéro de port indique vers quel programme envoyer les données.

Afin de ne pas pouvoir prévoir le contenu du dernier bloc, la taille des fragments varie d'un message à l'autre. En effet, découper le message en fragments de taille fixe à chaque fois fait que le padding sera toujours identique et ceci nous a semblé une divulgation inutile d'information, le pire des cas étant le dernier bloc ne contenant que des 16. Nous avons décidé d'envoyer entre 4 et 5 blocs AES, i.e. entre 64 et 80 octets pour chaque fragment, la valeur exacte étant définie aléatoirement pour chaque envoi de message. Ainsi, le même message envoyé successivement pourra être découpé en blocs de 68 octets la première fois et en blocs de 75 octets la seconde fois. C'est un compromis entre performance, sécurité et facilité d'implémentation.

Le message est donc découpé en plusieurs fragments de même taille et en un dernier de taille inférieure pour le reste. Chaque fragment est ensuite complété par un padding selon la norme PKCS7.

L'envoi dans le tunnel d'un fragment se fait comme ceci :

- génération d'un IV de transmission ;
- chiffrement du fragment avec la clé de session et l'IV ;
- concaténation de l'IV et des données chiffrées ;
- calcul du CBC-MAC avec la taille totale ajoutée au début ;
- concaténation de l'IV, des données chiffrées et du MAC ;
- envoi physique vers la carte.

Références

- [CBC-MAC] Mihir Bellare and Joe Kilian and Phillip Rogaway, *The Security of the Cipher Block Chaining Message Authentication Code*, 2000
- [NSA] CNSS Secretariat, National Security Agency, *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*, Juin 2003
- [RGS ANSSI] Agence nationale de la sécurité des systèmes d'information, *Annexe B1 - Référentiel Général de Sécurité*, 26 janvier 2010