Programmation cartes à puces

Loïc Demange
loic.demange@etud.univ-paris8.fr

avec les notes de cours de Philippe Guillot

7/18 mars 2021

Banque

Dans la réalité, les commandes de débit et de crédit ne sont produites que pour être réalisées une unique fois, avec une vérification d'intégrité et d'authenticité.

Comment faire en sorte qu'une commande ne puisse être éditée que par la banque ?
Comment éviter qu'une commande puisse être réalisée plusieurs fois ?
Comment éviter qu'on puisse modifier la somme dans le commande ?

De manière générale, comment faire en sorte que la commande ne soit exécutée uniquement si elle est authentique, intègre, et unique ?

Entête, compteur et chiffrement

Réponse : en rajoutant une entête et compteur, le tout chiffré.

Notre nouvelle commande contiendra donc, en données :

- Une entête fixe de 4 octets
- Un compteur de 2 octets qui s'incrémente à chaque opération de la banque
- Un solde de 2 octets

le tout chiffré par un algorithme de chiffrement symétrique.

Entête, compteur et chiffrement

Concrètement,

- La somme ne peut pas être falsifiée directement, car chiffrée.
- Si la commande a un souci d'intégrité ou qu'on cherche à falsifier, l'entête sera fausse.
- Si la commande a déjà été exécutée, la vérification du compteur permettra d'invalider la commande.

TEA

Pour le chiffrement, nous allons utiliser l'algorithme symétrique TEA , qui prend des clés de taille 128 bits.

```
void tea_chiffre(uint32_t * clair,uint32_t * crypto, uint32_t * k);
void tea_dechiffre(uint32_t * crypto,uint32_t * clair, uint32_t * k);
```

clair et **crypto** prennent des tailles de 64 bits, **k** est la clé de 128 bits.

Remarque Il faudra donc convertir des 8x8 uint8_t en 2x32 uint32_t et inversement.

Commandes

- **81 10 00 00 10 x x x x x x x x x x x x x x x x** : introduction clé (16 octets)
- **81 11 00 00 08 x x x x x x x x** : test chiffrement clair
- **81 12 00 00 08 x x x x x x x x** : test déchiffrement chiffré
- **81 C0 00 00 n x x x x x x x x** : get_response()

get_response() fait partie de la norme, et permet de renvoyer des éléments en attente, qui n'ont pas pu être renvoyé par des commandes entrantes.

Les commandes 11 et 12 permettent donc un **get_response()** pour obtenir le chiffré/déchiffré.

Commandes

get_response() doit donc renvoyer le contenu d'un tableau alloué dynamiquement (par **malloc** par ex.) d'une certaine taille, le tout stocké dans les variables suivantes.

```
// Get_response
uint8_t* reponse = 0;                    // réponse
uint8_t tailleReponse = 0;                // taille de la réponse

// Fonction
void get_response();
```

Remarque get_response() doit désallouer le tableau après son envoi, avec **free**.

Commandes

Les commandes 11 et 12 ne sont là que pour vérifier le bon fonctionnement de l'algorithme.
Pour que ce soit fonctionnel, il faut modifier les commandes débit et crédit pour qu'elles prennent en entrée 8 octets, qu'elles déchiffrent les données, qu'elles vérifient l'entête et le compteur, et uniquement dans le cas d'un succès qu'elles débitent/créditent.

Remarque L'entête peut être "codée en dur" (vérifiée directement dans une condition), le compteur sera modifié et stocké dans l'EEPROM.

Remarque 2 La commande chiffrée devra prendre en compte l'endianness sur 64 bits, que ce soit pour le chiffrement/déchiffrement ou pour l'introduction de la clé. Pour le débit/crédit, l'entête, le compteur et le solde seront aussi influencés.

Projet porte monnaie

Implémenter les protections dans votre porte monnaie.