Rapport de projet

Louis HATTIGER - Valentin NASONE - Elisa ROUX

November 12, 2023

1 Introduction

Au travers de ce projet, nous avons configuré notre JavaCard afin qu'elle soit capable de signer des données fournies par le biais d'une application terminal mais également de fournir sa clé publique pour que l'utilisateur puisse vérifier leur authenticité. Afin de sécuriser l'accès à la Smart Card nous avons mis en place un système de code PIN à 4 chiffres.

2 Architecture de la solution

Afin de réaliser une quelconque action, l'applet doit être sélectionné par l'envoi des APDU suivants : "00 A4 04 00 0A AID".

2.1 Sécurisation de la carte par un code PIN

Pour ce faire nous avons utilisé la classe OwnerPIN. Dans un premier temps, nous mettons les APDU reçus dans un buffer. Ensuite, nous regardons si le premier bloc en hexa, le CLA est bien égal à 0 puisque nous ne l'avons pas changé dans le code. Après, si le deuxième bloc, celui d'instruction INS, est bien égal à l'instruction pour vérifier un code PIN (0x20 dans notre cas cf. cheefs.....) nous appelons la méthode correspondante. Celle-ci va comparer la taille du code entré (5e bloc) avec la taille du code PIN attendu. Si la taille est correcte alors OwnerPIN va valider ou non le code PIN entré par l'utilisateur et renvoyer le code correspondant (SW1 = 0x90 SW2 = 0x00 en cas de succès).

Format d'APDU pour rentrer le code pin 1234 : "00 20 00 00 04 01 02 03 04"

Nous avons aussi ajouté la possibilité de modifier le code PIN car OwnerPIN permet de garder en mémoire le fait qu'un utilisateur s'était correctement authentifié juste avant et permet de faire cette action.

Format d'APDU pour modifier un code PIN en 1235 : "00 30 00 00 04 01 02 03 05"

Voici les APDU à envoyer pour faire une action complète d'authentification et de modification du code PIN :

"00 A4 04 00 0A AID" puis "00 20 00 00 04 01 02 03 04" puis "00 30 00 00 04 01 02 03 05"

2.2 Mécanisme de génération de clés RSA

Notre génération de clés RSA passe par l'utilisation des fonctions de la librairie javacard.security. Dans le constructeur de la classe nous générons la paire de clés. La clé privée sera utilisée pour signer les données et la clé publique sera à envoyer à l'utilisateur pour qu'il puisse faire la vérification.

2.3 Signature des données

Pour signer les données nous voulions utiliser au début la classe Cipher mais nous avons rencontré des problèmes de compatibilités entre le package "javacardx.crypto" et notre version de java. Nous avons donc opté pour la classe Signature.

Tout d'abord on initialise l'objet "signer" dans le constructeur en spécifiant l'algorithme de signature. Puis on continue de l'initialiser dans la méthode de chiffrement des données en spécifiant que l'opération de signature se fera avec la clé privée. La signature est générée à l'aide de la méthode "sign". Enfin on envoie la signature est renvoyée dans la réponse APDU en utilisant "apdu.setOutgoingSend".

Format d'APDU pour signer des données :

"00 A4 04 00 0A AID" puis "00 40 00 00 04 01 02 03 04"

On est conscient que notre méthode de signature n'est qu'un MVP, en effet pour des gros fichiers ça ne fonctionne pas. De plus nous n'arrivons pas à la tester mais on ne comprend pourquoi ça ne fonctionne pas.

2.4 Envoi de la clé publique

Lorsque nous envoyons la clé publique, nous faisons une extraction de l'exposant et du modulo de celle-ci puis nous le renvoyons sous la forme d'APDU. Cette instruction est caractérisée par le code INS 0x50.

Format d'APDU pour récupérer la clé publique :

"00 A4 04 00 0A AID" puis "00 50 00 00 00"

3 Méthodes de test pour le développement

Pour écrire le code de l'applet de la JavaCard nous avons utilisé le logiciel JCIDE pour Windows qui testait les codes sur un simulateur de JavaCard. Ensuite nous génèrions le fichier .cap en lignes de commandes ainsi que décrit dans le Lab 2. Enfin nous téléchargions l'applet sur la JavaCard grâce à des scripts et gpshell.

Pour tester l'envoie d'APDU sur la carte dans un premier temps sans l'application terminal nous avons utilisé l'outil opensc-tool, par la suite nous nous servions de notre application terminal écrite en Python avec la librairie pyscard.
Notre application Python ne fonctionnement pas totalement, voici une possible commande permettant de tester le programme avec opensc-tool :

opensc-tool -s "00 A4 04 00 0A AID" -s "00 40 00 00 00" -s "00 50 00 00 00"

L'application python que nous avons essayé de mettre en place pour jouer le rôle d'application, permet de se connecter à la smartcard, de rentrer le code pin et de vérifier celui-ci. Il permet également de changer ce code pin. Cependant nous n'avons pas réussi à faire marcher la signature ainsi que la récupération de la clé publique en python. Une erreur de mémoire fût le problème.