Projet JavaCard

RAPPORT

Mohamed BOUCHENGUOUR
Hamadi DAGHAR
MASTER 2 CYBERSÉCURITÉ
POLYTECH UNICE

Table des matières

Initialisation du système

On commence par charger l'applet dans la carte. Lorsque l'applet est chargé, le constructeur initialise les paramètres de sécurité : un PIN avec une limite de 3 essais et une longueur de 4 chiffres, ainsi qu'une paire de clés RSA de 512 bits. Ces actions ne sont réalisées qu'une seule fois à l'initialisation et ne pourront pas être exécutées à nouveau, à l'image d'une carte bancaire, car la configuration initiale est conçue pour être permanente et garantir l'intégrité et la sécurité des données sensibles. De plus, sachant que l'applet appartient à une banque, il y aura dans l'applet directement l'adresse IP de la banque.

Pour l'initialisation, il faudra tout d'abord lancer le serveur Flask via le script server.py. Ce dernier génère une paire de clés RSA de 512 bits, initialise une base de données vide pour stocker les clés publiques des cartes, ainsi que des structures pour gérer les cartes volées et enregistrer les transactions.

Ensuite, il faut lancer le script init.py. Il commence par demander un code PIN à l'utilisateur afin de l'initialiser dans la carte via l'APDU 0x02. Une fois ce code défini, il devient permanent pour garantir la sécurité et éviter toute modification non autorisée, comme sur une carte bancaire. Une fois initialisé, le script demande à l'utilisateur le code PIN via l'APDU 0x03, qu'il vient de définir, pour vérifier son authenticité et valider l'accès à la carte.

Une fois l'authentification réussie, la clé publique générée sur la carte est récupérée via l'APDU 0x04, puis envoyée au serveur pour enregistrement. Au niveau du serveur, cette clé est vérifiée pour s'assurer qu'elle n'est pas marquée comme volée. Si elle est valide et absente de la base de données, elle est ajoutée à celle-ci. Si elle est déjà enregistrée, une confirmation est simplement renvoyée.

Ensuite, le script récupère la clé publique du serveur et la déploie sur la carte via l'APDU 0x05. Cette clé permettra à la carte de vérifier les signatures générées par le serveur, assurant l'authenticité des communications futures.

Fonctionnalités runtime

Une fois le tout initialisé, il faut lancer le script vending (tout en gardant le script du serveur actif, sans l'arrêter car à chaque démarrage, la clé RSA du serveur, les cartes et transactions enregistrées sont réinitialisées. Évidemment, en situation réelle, il faudrait enregistrer ces données de manière sécurisée afin d'éviter de les perdre à chaque redémarrage.). Celui-ci commence par demander à l'utilisateur de saisir son code PIN pour vérification via la carte. Une fois le PIN validé, le script utilise la fonction pay pour traiter les transactions.

Le script propose plusieurs façons d'envoyer une transaction :

- **Saisie manuelle** : L'utilisateur entre les données de la transaction directement dans la console.
- **Transactions prédéfinies** :
    - Une transaction simple (testTransaction).

    - Une transaction contenant une chaîne répétée (testTransaction2) pour tester les transmissions dépassant 255 octets.
- **Fichier externe** : L'utilisateur peut charger un fichier contenant les données de la transaction (par exemple, transaction.txt dans le dossier).

Pour chaque transaction :

- **Récupération de la date et de l'heure du serveur** : Le script envoie une requête au serveur (/time) pour obtenir une horodatation signée. Le serveur retourne la date, l'heure, et une signature associée qui garantit l'authenticité de l'horodatage.
- **Préparation des données** : La chaîne contenant la date et sa signature est concaténée à la description de la transaction.
- **Transmission à la carte** : La chaîne complète est envoyée à la carte via l'APDU 0x09.

Gestion des données longues de vending vers la carte

Pour transmettre des données trop longues à la carte (plus de 255 octets), on utilise un mécanisme basé sur des fragments envoyés en plusieurs APDU (via la fonction handleMultiAPDU en java et handle_multi_apdu en python) en utilisant le paramètre P1 pour indiquer l'état de la transmission :

1. **Premier fragment (P1 = 0)** : Ce fragment contient les deux premiers octets qui indiquent la taille totale des données que la carte doit recevoir, suivis par la première portion des données. La carte initialise un tampon pour stocker les données reçues et commence à assembler les fragments.
2. **Fragments intermédiaires (P1 = 1)** : Chaque fragment intermédiaire contient une portion des données. La carte les ajoute au tampon jusqu'à atteindre la longueur totale attendue.
3. **Dernier fragment (P1 = 2)** : Ce fragment final complète la transmission. La carte vérifie que la longueur totale des données correspond bien à celle spécifiée dans le premier fragment. Une erreur est levée si les données sont incomplètes ou invalides.

Si P1 = 0 et que les données envoyées sont inférieures à 255 octets, elles sont finalisées directement sans fragments supplémentaires.

Chiffrement et signature dans la carte

Une fois les données reçues, la carte vérifie la validité de la signature du serveur à l'aide de la clé publique du serveur. Si la signature est invalide ou si les données sont insuffisantes (moins de 64 octets pour la signature et 19 pour la date), une erreur est levée. Si la signature est valide, l'horodatage est extrait et concaténé à la description de la transaction avant d'être chiffré.

Les données de transaction sont ensuite divisées en blocs de 53 octets, une limite imposée par les contraintes de chiffrement RSA 512 bits (la taille maximale des données à chiffrer est

légèrement inférieure à la taille de la clé, en raison des exigences de padding PKCS#1). Chaque bloc de 53 octets génère un bloc chiffré de 64 octets, correspondant à la taille de sortie pour RSA 512 bits.

Chaque bloc chiffré est ensuite signé individuellement avec la clé privée de la carte, produisant une signature de 64 octets pour chaque bloc. Les signatures et les blocs chiffrés correspondants sont concaténés dans un tampon de transmission, dans l'ordre suivant:

[Signature1, Chiffré1, Signature2, Chiffré2, Signature3, Chiffré3…]

Ainsi, si on a 3 blocs de 53 octets en entrée, le résultat final sera composé de 3 blocs de 128 octets concaténés (64 octets de signature + 64 octets de données chiffrées pour chaque bloc).

Gestion des données longues de la carte vers vending

Pour transmettre des données trop longues de la carte vers la machine de vending (plus de 256 octets), un mécanisme basé sur des fragments est également utilisé, et la gestion se fait à l'aide de P1 ainsi que globalOffset, qui agit comme un curseur indiquant combien d'octets ont déjà été envoyés. La transmission utilise toujours l'APDU 0x09 (fonction pay), comme pour l'envoi de données de la machine de vending vers la carte. Voici le fonctionnement :

1. **Premier fragment (P1=3 avec globalOffset=0) :**
    - La carte ajoute les 2 premiers octets pour indiquer la taille totale des données que la machine de vending doit recevoir.
    - Le reste du premier fragment contient les premières données à transmettre.
    - La machine de vending extrait ces 2 octets pour connaître la longueur totale et conserve le reste comme le premier fragment reçu.
2. **Fragments suivants (P1 = 3 avec globalOffset > 0)** :
    - Tant que toutes les données n'ont pas été reçues, la machine de vending envoie des APDU avec P1 = 3 pour demander les fragments restants.
    - La carte utilise globalOffset pour déterminer la portion des données à envoyer dans chaque fragment.
3. **Finalisation** :
    - Le transfert se termine une fois que la machine de vending a reçu toutes les données (vérifié en comparant la longueur totale reçue avec la longueur spécifiée dans le premier fragment). Si une erreur survient, elle est signalée.

Vérification signature des données depuis la machine vending

Une fois la transaction complète reçue, la machine de vending utilise la clé publique RSA de la carte pour vérifier l'authenticité de chaque bloc de données. On rappelle que chaque bloc est composé de 128 octets, divisés en deux parties : les 64 premiers octets représentent la signature, tandis que les 64 octets suivants correspondent au contenu chiffré. Pour chaque bloc, la machine vérifie que la signature correspond bien au contenu chiffré en utilisant la clé publique de la carte. Si une signature est invalide, la vérification échoue immédiatement, et la

communication est interrompue. En revanche, si toutes les signatures sont valides, la transaction est acceptée comme authentique.

Récupération de l'adresse IP (signée) du serveur sur la carte

Ensuite, la machine de vending récupère l'adresse IP du serveur de vérification. Cette adresse, envoyée par la carte, est accompagnée d'une signature pour en garantir l'authenticité. La machine vérifie la validité de la signature à l'aide de la clé publique de la carte. Si la signature est invalide, la communication est immédiatement interrompue.

La signature de l'IP garantit que l'adresse provient bien de la carte et qu'elle n'a pas été modifiée ou falsifiée pendant la transmission. Cela renforce la sécurité en évitant tout risque de redirection vers un serveur non autorisé ou compromis.

Envoie de la transaction au serveur

Une fois l'adresse IP du serveur validée, la machine de vending encode la transaction en Base64 et construit une requête HTTP pour l'envoyer au serveur de vérification. La requête inclut deux éléments :

- La transaction signée et chiffrée (encodée en Base64).
- La clé publique de la carte, utilisée uniquement pour identifier à quelle carte appartient la transaction.

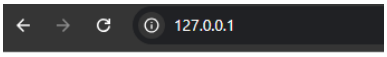
Lorsque le serveur reçoit la requête via le point d'entrée /transaction, il :

- **Vérifie si la clé est enregistrée** : Si la clé fournie ne correspond à aucune clé connue par le serveur, la transaction est rejetée.
- **Vérifie si la carte est marquée comme volée** : Si la clé publique correspond à une carte signalée comme volée, la transaction est immédiatement rejetée.
- **Vérifie si la signature correspond aux données chiffrées :** Si la signature est valide et correspond aux données, la transaction est acceptée, sinon elle est rejetée.
- **Enregistre la transaction :** La transaction et la clé publique correspondant à cette transaction sont enregistrées.

Visualisation des données décryptées

Pour obtenir les transactions décryptées, le serveur suit les étapes suivantes :

1. **Connexion au serveur** : L'utilisateur se connecte à l'URL du serveur (http://127.0.0.1/ ici) pour accéder à l'interface de gestion des transactions.



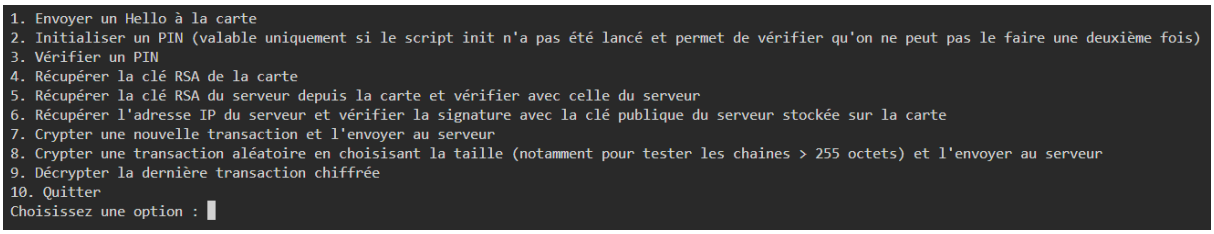
2. **Récupération de la clé RSA** : Le serveur récupère la clé publique RSA de la carte pour identifier celle-ci.
3. **Vérification de la carte** : Le serveur vérifie si la carte n'est pas signalée comme volée et si elle est enregistrée dans sa base de données.
4. **Demande du PIN** : Une fois la carte validée, l'utilisateur doit saisir son PIN, qui est envoyé à la carte pour vérification.

5. **Envoi des transactions à la carte** : Le serveur récupère toutes les transactions associées à la carte, les concatène, puis les envoie à la carte via un transfert **multi-APDU** (similaire à pay).
6. **Vérification et décryptage par la carte** :
    - La carte traite chaque bloc de 128 octets, composé de **64 octets pour la signature** et **64 octets pour les données chiffrées**.
    - Elle utilise sa **clé publique** pour vérifier la signature des 64 premiers octets par rapport aux 64 octets suivants.
    - Si la signature est valide, la carte utilise sa **clé privée** pour décrypter les données.
7. **Résultat** : Si toutes les signatures sont valides, les transactions déchiffrées sont renvoyées au serveur. Le serveur utilise une **expression régulière (regex)** pour détecter les horodatages (au format YYYY-MM-DD HH:MM:SS) et formate correctement les transactions, en les affichant ligne par ligne dans l'interface utilisateur au lieu d'une seule ligne compacte.

Réalisation des tests

Un script de test (test.py) a été développé pour valider les principales fonctionnalités de l'application. Lors de son exécution, un menu interactif permet de tester divers aspects du système, tels que :

- L'interaction avec la carte (communication, initialisation et vérification du PIN, récupération de clés).
- La validation des échanges sécurisés avec le serveur (adresse IP, clés RSA, signatures).
- Le traitement des transactions (chiffrement, envoi au serveur, déchiffrement).



Ce tableau récapitule les différentes instructions APDU utilisées pour communiquer avec l'applet JavaCard.

Instruction	**Valeur (hex)**	**Description**
CLA_PROJECT	0x42	Classe de l'applet utilisée pour toutes les commandes.
INS_HELLO	0x01	Envoie un message de test.
INS_SET_PIN	0x02	Initialise un PIN sur la carte.
INS_VERIFY_PIN	0x03	Vérifie le PIN soumis par l'utilisateur.
INS_GET_RSA_KEY	0x04	Récupère la clé publique RSA générée sur la carte.
INS_STORE_SERVER_KEY	0x05	Déploie la clé publique du serveur sur la carte.
INS_GET_SERVER_RSA_KEY	0x06	Récupère la clé publique RSA du serveur depuis la carte.
INS_GET_SERVER_IP	0x08	Récupère l'adresse IP du serveur, signée pour validation.
INS_PAY	0x09	Envoie une/des transaction(s) à la carte pour chiffrer et signer.

Instruction	Valeur (hex)	Description
INS_DECRYPT_LOG	0x0A	Envoie une/des transaction(s) chiffrer et signer à la carte pour déchiffrer.
INS_SIMPLE_ENC	0x0B	Effectue un chiffrement simple pour tester.