Membres du groupe:

- DUMANOIS Arnaud
- MAUROIS Quentin
- BEUREL Simon

1. Introduction

Ce projet consiste à développer un système de terminal et de carte à puce pour créer un porte-monnaie électronique sécurisé destiné à des distributeurs automatiques

Voici les principaux aspects sécurisés :

- Vérification des transactions par un serveur de confiance (back-end) pour :
    - Protéger contre le vol de carte.
    - Éviter l'utilisation de cartes frauduleuses comme des "YesCards" (faux clones).
- Sécurisation contre des distributeurs automatiques malveillants (faisant semblant d’être légitimes).
- Compatibilité avec différents types de distributeurs automatiques.

Le système inclut la protection par code PIN, la signature des transactions avec RSA, et une vérification par serveur pour prévenir les fraudes.

Pour réaliser ce projet, plusieurs problèmes doivent être adressés. Il est primordial d'empêcher l'utilisation frauduleuse de la carte en cas de vol, ce qui nécessite une protection par code PIN. La fiabilité des transactions doit être garantie grâce à une signature numérique et une vérification indépendante par un serveur. De plus, la solution doit être interopérable pour fonctionner avec différents types de distributeurs automatiques. Enfin, les logs des transactions doivent être stockés et consultés de manière sécurisée, et un protocole efficace doit être implémenté pour permettre la gestion des données volumineuses lors des échanges entre la carte et le terminal.

- Résumé des objectifs :
    - Développement d’un applet JavaCard.
    - Mise en place d’applications terminales (distributeur et serveur de vérification).
    - Définition et implémentation de protocoles sécurisés.

---

2. Architecture Globale

- **Présentation des acteurs :**

    - Carte à puce

Le premier acteur qui est très important pour la réalisation de ce projet est la carte à puce sur laquelle il est possible d'y intégrer du code Java en utilisant notamment la librairies "javacard" combiné au JDK8. Le but de cette carte à puce sera de simuler une carte bancaire permettant de réaliser un achat auprès de la vending machine

    - Machine distributrice (vending machine)

Ce deuxième acteur est un programme python permettant de réaliser la connexion à la carte à puce et de lui transmettre les différentes informations sur les produits que la carte peut acheter. La réalisation de cet acteur se fait grâce à la librairie Python "smartcard" (https://pypi.org/project/smartcard/)

    - Serveur de vérification

Le troisième et dernier acteur du projet correspond au serveur de vérification qui va servir d'autorité de confiance pour vérifier les achats réalisés par la carte à puce. Chaque transaction sera vérifiée par le serveur pour protéger contre les potentiels vols de carte ou pour protéger contre l'utilisation de fausses cartes. Ce troisième acteur est développé avec le langage Python.

- **Schéma d'architecture globale :**
    - Communication entre les trois entités.

Les trois entités ne peuvent pas toutes communiquer entre elles. En effet, pour des questions de sécurité nous avons seulement créer 2 canaux de communications:

Le premier est un canal qui lie la vending machine au serveur de vérification. Ce canal est notamment utilisé quand le distributeur automatique à besoin d'avoir une vérification auprès du serveur de vérification, de la transaction qu'elle reçoit pour être sûre que cela n'est pas une arnaque

Le deuxième canal de communication instauré est un canal permettant la communication entre la machine distributrice et la carte à puce. Ce canal est très important car c'est lui qui permet à la carte à puce de pouvoir envoyer les transactions qu'elle souhaite réaliser.

    - Protocoles de signature et de chiffrement.

Lors de ce projet, nous avons utilisé comme demandé le protocole RSA-512 pour pouvoir utiliser du chiffrement asymétrique. Ce protocole permet de pouvoir générer une paire de clé publique/privée. Ce protocole est notamment utilisé sur la carte à puce et sur le serveur de vérification pour qu'ils puissent générer leurs paires de clés.

Le protocole de signature utilisé est le protocole présent au sein de la librairie Python 'RSA' (https://pypi.org/project/rsa/). Grâce au concept de signature, il est possible de pouvoir vérifier avec une clé publique l'origine du message (dans notre cas, que c'est bien la clé privée de la carte qui a été utilisée pour signer la transaction).

---

3. Développement

3.1. Applet JavaCard

L'applet JavaCard implémente plusieurs fonctionnalités essentielles. Lors de l'initialisation, la carte génère une paire de clés RSA 512 bits et enregistre la clé publique du serveur de vérification. Un code PIN est également configuré pour sécuriser son utilisation. En cours de fonctionnement, l'applet gère les transactions en signant et chiffrant les données des achats transmises par le terminal. Elle est également capable de transmettre l'adresse IP du serveur de vérification pour garantir l'authenticité des distributeurs.

Pour permettre la gestion de données volumineuses, un protocole multi-APDU a été développé. Il assure la fragmentation et le réassemblage des messages échangés entre la carte et le terminal. Les méthodes pay() et decryptLog() illustrent cette gestion, respectivement pour le traitement des paiements et la lecture sécurisée des logs.

3.2. Applications en ligne de commande

Deux applications en ligne de commande ont été développées pour interagir avec la carte. La première simule une machine distributrice, envoyant les descriptions des produits à la carte et recevant les données signées en retour. Ces signatures sont validées à l'aide du serveur de vérification pour garantir l'intégrité des transactions.

La seconde application représente le serveur de vérification. Elle gère les logs des transactions et fournit une heure et une date authentifiées pour éviter les attaques de distributeurs malveillants. Ce serveur assure également l'interaction avec l'utilisateur pour la saisie du PIN et permet de tester les fonctionnalités de la carte via une interface simplifiée.

3.3. Interaction avec la carte via APDU

Les communications entre le terminal et la carte à puce se font via des commandes **APDU** (Application Protocol Data Unit). Un APDU est un message structuré permettant d'envoyer des instructions à la carte et de recevoir des réponses de celle-ci. Ces échanges sont essentiels pour la gestion des transactions et le contrôle de l'intégrité des données.

Une commande APDU est composée de six champs :

- **CLA (Class)** : classe de l'instruction, indiquant le type d'instruction à exécuter.
- **INS (Instruction)** : instruction à exécuter sur la carte.
- **P1 (Parameter 1)** : premier paramètre de l'instruction, souvent utilisé pour indiquer le mode d'exécution.
- **P2 (Parameter 2)** : second paramètre de l'instruction, permettant de préciser d'autres options.
- **Lc (Length of data field)** : longueur des données envoyées dans le champ Data.
- **Data** : données envoyées à la carte pour exécution (si nécessaire).
- **Le (Length of expected data)** : longueur des données attendues en réponse.

Une réponse APDU se compose de deux champs :

- **Data** : données renvoyées par la carte en réponse à l'instruction.

- **SW1 SW2 (Status Word)** : code de statut indiquant le résultat de l'exécution de la commande.

---

4. Protocole Cryptographique

- Description détaillée des échanges sécurisés :
    - Entre la carte et la machine distributrice.

Les différents échanges entre la carte à puce et la machine distributrice sont chiffrés notamment en utilisant le protocole RSA mentionné précédemment. En effet, car la carte à puce va vouloir faire une transaction, cette dernière va devoir chiffrer cette transaction avec la clé publique du serveur de vérification, puis elle va signer cette transaction avec sa propre clé publique.

Grâce à ce schéma, on peut éviter les attaques Man-in-the-Middle car la communication sera donc impossible à être déchiffrée. Seul le serveur de vérification sera capable de pouvoir déchiffrer la transaction car c'est la seule entité qui possède la clé privée associée à la clé publique utilisée pour chiffrer la transaction.

    - Entre la machine distributrice et le serveur de vérification.

Concernant les échanges entre la machine distributrice et le serveur de vérification, les échanges sont chiffrés mais de manière indirecte. En effet, comme mentionné précédemment, les transactions envoyées par la carte sont chiffrées, ce qui amène au fait que la transaction qui sera envoyée par la machine distributrice au serveur de vérification sera donc par transitivité également chiffrée.

- Rationnel derrière les choix :
    - RSA 512 bits pour garantir un équilibre entre sécurité et performances sur JavaCard.

Le protocole RSA-512 est le protocole demandé dans le sujet. Ce protocole est justifié par le fait qu'à l'heure actuelle c'est l'un des protocoles cryptographiques les plus utilisés dans le monde car son efficacité et sa robustesse sont démontrées par des calculs mathématiques. Cependant, l'un des défauts d'utilisation que nous faisons est que nos clés sont générées à partir de seulement 512 bits, ce qui peut être attaqué si un attaquant combine la puissance de plusieurs ordinateurs. Cependant, la contrainte technique de la JavaCard nous oblige à garder cette spécificité de 512 bits, mais il serait intéressant de se poser la question d'une potentielle augmentation du nombre de bits utilisés pour générer les paires de clés dans le futur.

    - Signature et chiffrement pour protéger l'intégrité et la confidentialité des données.

Le principe de signature utilisé dans ce projet est un principe très important en cryptographie car il permet de pouvoir lier un message à une personne. En effet, imaginons le problème suivant: Vous êtes Bob et vous recevez une déclaration d'amour, Alice et Manon vous assurent que c'est elle qui a envoyé le message, comment savoir qui dit la vérité ? C'est à cause de ce problème que les protocoles de signatures ont été intégrés.

La signature cryptographique se base sur le chiffrement asymétrique. Pour signer un message, il vous suffit de le signer en utilisant un protocole précis (comme RSA PKCS#1) et une clé privée, et ainsi n'importe quel utilisateur pourra vérifier en utilisant votre clé publique que c'est bien votre personne qui a signé le message envoyé.

C'est donc pour cela que nous avons intégré le concept de signature dans notre projet. Grâce à ce concept, nous pouvons prouver qu'une transaction mal signée est une transaction frauduleuse, car l'attaquant ne connaîtra pas la clé privée nécessaire pour signer la transaction.

---

5. Tests et Résultats

Les tests ont été effectués de manière interactive en échangeant des messages APDU entre la carte et le serveur. Chaque scénario a été conçu pour valider une fonctionnalité spécifique et garantir le bon fonctionnement des différentes entités.

Le système a été testé avec plusieurs scénarios représentatifs des cas d'utilisation. Voici un résumé des résultats obtenus :

1. **Scénario 1 : Connexion, ajout d'articles dans le panier et validation de l'achat**
   Lors de la connexion, l'utilisateur a saisi son code PIN, ce qui a déclenché la génération d'une paire de clés RSA par la carte. La clé publique a ensuite été transmise au serveur pour les communications sécurisées. Après une authentification réussie, l'utilisateur a pu ajouter plusieurs articles dans son panier et valider l'achat. Chaque étape s'est déroulée sans erreur, avec des échanges sécurisés et validés par le serveur.

   Ainsi depuis le terminal de la machine distributrice on effectue les actions suivantes :

```
C:\Python311\python.exe C:\Users\arnau\Documents\PNS\SI5\Bime
=== Machine Distributrice ===
1. Se connecter (PIN requis)
0. Quitter

Choisissez une option: 1
Entrez votre code PIN (4 chiffres): 1234

Authentification...
Authentification réussie!

Initialisation de la connexion sécurisée...
Récupération de l'adresse IP et du port du serveur réussie
Récupération de la clé publique réussie
Échange complet des clés réussi
La clé correspond à celle du serveur
=== Machine Distributrice ===
1. Voir les produits
2. Voir le panier
3. Payer
4. Changer le code PIN
5. Voir l'historique des achats
```

```
Choisissez une option: 1
=== Produits disponibles ===
1. Barre Protéinée - 2.5€
2. Canette Coca-Cola - 1.5€
3. Paquet de chips - 1.8€
4. Madeleine Bretonne - 1.2€
5. Eau minérale - 1.0€

0. Retour au menu

Choisissez un produit (0 pour terminer): 1

Barre Protéinée ajouté au panier!
Appuyez sur Entrée pour continuer...
=== Produits disponibles ===
1. Barre Protéinée - 2.5€
2. Canette Coca-Cola - 1.5€
3. Paquet de chips - 1.8€
4. Madeleine Bretonne - 1.2€
5. Eau minérale - 1.0€

0. Retour au menu

Choisissez un produit (0 pour terminer): 5

Eau minérale ajouté au panier!
Appuyez sur Entrée pour continuer...
=== Produits disponibles ===
1. Barre Protéinée - 2.5€
2. Canette Coca-Cola - 1.5€
3. Paquet de chips - 1.8€
4. Madeleine Bretonne - 1.2€
5. Eau minérale - 1.0€

0. Retour au menu

Choisissez un produit (0 pour terminer): 0
=== Machine Distributrice ===
1. Voir les produits
2. Voir le panier
3. Payer
4. Changer le code PIN
5. Voir l'historique des achats
0. Se déconnecter

Choisissez une option: 3

Traitement du paiement...
Transaction envoyée avec succès
Paiement effectué avec succès!
Appuyez sur Entrée pour continuer...

=== Machine Distributrice ===
1. Voir les produits
2. Voir le panier
3. Payer
4. Changer le code PIN
5. Voir l'historique des achats
0. Se déconnecter
```

Et depuis le terminal du serveur on confirme la communication avec la machine :

```
C:\Python311\python.exe C:\Users\arnau\Documents\PNS\SI5\Bimestre-1\Secu-IOT\PNS_IoT_Security_SI5\Trusted_Server\server.py
Clés RSA générées.
Serveur en écoute sur localhost:12345...
Connexion reçue de ('127.0.0.1', 53428)
Clé publique pour card_3143125856336 sauvegardée avec succès
Clé client stockée: True
Envoi de notre clé publique...
Connexion reçue de ('127.0.0.1', 53446)
Signature vérifiée avec succès pour le client card_3143125856336
Données déchiffrées: Barre Protéinée + Eau minérale : 3.50€
Connexion reçue de ('127.0.0.1', 53468)
logs [{'timestamp': '2025-01-09 18:16:20', 'client_id': 'card_3143125856336', 'encrypted_data': 'hUpuuSrwenHvD7Qa37z6tBWadU8p17NbSKce+YWI4zT4XrstADYfUAUk16hQSyWqfKYiOfg6Q/N6SpzuOFGFtA==', 'signature': 'FnMenLAEnq+a8Z/4ct3WTl
```

2. **Scénario 2 : Consultation de l'historique des ventes** L'utilisateur connecté a accès à l'historique des transactions précédentes grâce à des logs stockés sur le serveur. Ces logs ont été vérifiés et correspondent aux transactions effectuées précédemment, assurant ainsi la traçabilité des achats.

```
Récupération de l'historique des achats...

=== Historique des achats ===
Date: 2025-01-09 18:16:20
Transaction: Barre Protéinée + Eau minérale : 3.50€
Signature vérifiée: Oui
----------------------------------------
```

3. **Scénario 3 : Changement du PIN** Le changement de code PIN a été testé avec succès. La carte a accepté la nouvelle valeur, l'a sauvegardée de manière sécurisée, et le serveur a confirmé que l'authentification fonctionnait avec le nouveau code. Ce processus garantit une flexibilité et une sécurité accrue pour l'utilisateur.

```
Appuyez sur Entrée pour continuer...
=== Machine Distributrice ===
1. Voir les produits
2. Voir le panier
3. Payer
4. Changer le code PIN
5. Voir l'historique des achats
0. Se déconnecter

Choisissez une option: 4
Entrez le nouveau code PIN (4 chiffres): 11111
Le PIN doit contenir exactement 4 chiffres.
Entrez le nouveau code PIN (4 chiffres): 111
Le PIN doit contenir exactement 4 chiffres.
Entrez le nouveau code PIN (4 chiffres): 1111
Modification réussie
Appuyez sur Entrée pour continuer...
```

```
C:\Python311\python.exe C:\Users\arnau\Documents\PNS\SI5\Bimestr
=== Machine Distributrice ===
1. Se connecter (PIN requis)
0. Quitter

Choisissez une option: 1
Entrez votre code PIN (4 chiffres): 1234

Authentification...
Code PIN incorrect!
Appuyez sur Entrée pour continuer...
=== Machine Distributrice ===
1. Se connecter (PIN requis)
0. Quitter

Choisissez une option: 1
Entrez votre code PIN (4 chiffres): 1111

Authentification...
Authentification réussie!

Initialisation de la connexion sécurisée...
Récupération de l'adresse IP et du port du serveur réussie
Récupération de la clé publique réussie
Échange complet des clés réussi
La clé correspond à celle du serveur
=== Machine Distributrice ===
1. Voir les produits
2. Voir le panier
3. Payer
4. Changer le code PIN
5. Voir l'historique des achats
0. Se déconnecter

Choisissez une option:
```

---

6. Documentation Technique

- Dépendances et environnement requis.

Pour faire fonctionner le projet, assurez-vous d'avoir sur votre machine Python3 ainsi que le gestionnaire de paquets Python Pip. Avant de lancer les deux applications terminales python, vous devez vous rendre dans les deux dossiers /Client_App et /Trusted_Server et exécuter la commande:

pip3 install -r requirements.txt

Cette commande permet de pouvoir installer les différents paquets Python nécessaires au bon fonctionnement des deux applications Python écrites.

- Procédure d'installation :

Concernant l’applet Java, cette dernière est déjà intégrée à la carte à puce donnée au début de ce projet. Il vous suffit simplement de brancher la carte à puce à votre machine en utilisant son port USB

- Démarrage des applications terminales.

Pour démarrer la vending maching, vous devez exécuter dans un terminal et en étant dans le répertoire /Client_App la commande suivante:

python3 vending-machine.py

Pour démarrer le serveur de vérification, vous devez exécuter dans un terminal en étant dans le répertoire /Trusted_Server la commande suivante:

python3 server.py

/!\ ATTENTION /!\ Il faut démarrer en premier le serveur de vérification, puis démarrer la vending machine seulement quand le dossier /keys est bien généré.

---

7. Conclusion

- Bilan des forces et faiblesses :
  - Avantages de la solution proposée.

Sécurisation des transactions : L'utilisation de la signature numérique RSA et du chiffrement asymétrique garantit l'intégrité et la confidentialité des données échangées entre la carte, la machine distributrice et le serveur de vérification.

Protection contre la fraude : La vérification systématique des transactions par le serveur limite les risques d'utilisation de cartes clonées ou volées, renforçant ainsi la sécurité du système.

Gestion des logs sécurisée : L'enregistrement et la consultation des journaux de transactions permettent d'assurer une traçabilité complète et fiable des opérations effectuées.

  - Limites identifiées

Puissance de la clé RSA : L'utilisation d'une clé RSA de 512 bits constitue une faiblesse face aux capacités de calcul actuelles. Une clé plus robuste (2048 bits ou plus) serait recommandée, bien que limitée par les contraintes techniques de la JavaCard.

Dépendance au serveur de vérification : La nécessité d'une connexion constante avec le serveur de vérification peut être un frein dans des environnements où la connectivité est instable ou inexistante.

- Perspectives d’améliorations

Mettre en place des protocoles de reprise en cas de perte de connexion avec le serveur.

Renforcer la sécurité physique des composants.