République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département Informatique

Projet de fin d'étude pour l'obtention du diplôme Licence

Option

Génie des **T**élécommunications et **R**éseaux

Thème

Initialisation et mise en œuvre de carte à puce

Sujet Proposé par :

Pr. A. BELKHIR

Présenté par :
BEKKOUCHE Oussama
TOUMI Nassima

Soutenu le 07/06/2015

Devant le jury composé de :

Mr. K. ATIF **Président**
Mme. C. BENZAID **Examinatrice**

Binôme N°097/2015

Remerciements

Nous tenons à remercier nos deux familles respectives (parents, frères et soeurs) pour leur soutien indéfectible et leurs conseils ainsi que nos amis et proches.

Nous tenons également à exprimer notre reconnaissance à nos professeurs et en particulier à notre promoteur Pr. BELKHIR pour nous avoir encadré, orienté, aidé et conseillé tout au long de l'année.

Nous adressons nos sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques nous ont aidé d'une quelconque manière.

Introduction générale

L'utilisation grandissante des systèmes de communications électroniques dans tous les domaines a engendré la nécessité de mettre en place des solutions de sécurité pour l'identification et l'authentification des personnes et pour assurer la confidentialité des données.

Plusieurs solutions ont été proposées parmi lesquelles l'utilisation des cartes à puces : Leurs portabilité, flexibilité et leurs capacités à stocker des données et à effectuer des calculs cryptographiques en ont fait un moyen efficace pour assurer un bon niveau de sécurité dans un système électronique.

D'autre part l'introduction de la technologie Java Card a permis de démocratiser l'utilisation des cartes à puce en facilitant le développement d'applets pour cartes à puce et en introduisant le concept de cartes *multiapplicatives*. Il est donc possible à présent de mettre en œuvre des cartes à puce dans des systèmes électroniques sans grandes contraintes.
Aujourd'hui, la carte à puce est utilisée pour des applications diverses telles que le contrôle d'accès, le commerce électronique, les applications d'administration, l'e-ID ...etc

Ce projet s'inscrit dans le cadre de la réalisation de systèmes mettant en œuvre cartes à puce, le but étant de développer des solutions pour usage administratif et commercial. Nous avons donc abordé les étapes d'implémentation de cartes à puce dans un système électronique :

- Création et configuration d'une autorité de certification.
- Développement de l'applet Java Card et de l'application terminal.
- Installation de l'applet et initialisation des données utilisateur.

Nous avons ensuite détaillé la conception et les fonctionnalités de chaque application selon son usage.

Nous avons structuré notre mémoire de la manière suivante :

- **Chapitre I :** État de l'art.
- **CHAPITRE II :** Mise en œuvre de cartes à puce.
- **CHAPITRE III :** Applications.

Chapitre I : État de l'art

I. Introduction

1. Présentation

Le terme « carte à puce » désigne une carte en plastique intégrant une puce électronique qui peut stocker une certaine quantité de données, son contenu est accessible par un lecteur de cartes via une interface électronique (cartes à contact) ou à distance grâce à une antenne intégrée (cartes sans contact). Les domaines d'utilisation des cartes à puces sont très variés, ils s'étendent de la téléphonie (carte SIM) au secteur de la santé avec les cartes de sécurité sociale en passant par le domaine financier (cartes bancaires).

2. Types de cartes à puce

Il existe plusieurs types de cartes à puce :[6]

2.1 Cartes à puces synchrones

— Les cartes « à mémoire » dont la puce contient simplement une mémoire, elles ne peuvent donc effectuer aucun calcul, ne peuvent être programmées et ne proposent aucune protection des données ; l'écriture dans la mémoire peut être rendue impossible une fois initialisée. Ce type de cartes est souvent utilisé pour les services prépayés.

— Les cartes « à logique câblée » dont la mémoire est accessible à travers des circuits préprogrammés et figés pour une utilisation spécifique. Contrairement au type précédent, ces cartes proposent une protection des données via un système d'authentification.

2.2 Cartes à microprocesseur (Asynchrones)

Plus communément appelées « Smart Cards », elles contiennent un processeur et de la mémoire, le tout dans un microcontrôleur. Ceci permet de programmer la carte pour différents services tout en assurant une sécurité des données.

Architecture :

— CPU : Microprocesseur qui effectue les calculs.
— ROM (Read Only Memory) : Contient le système d'exploitation et éventuellement le JCRE (Java Card Runtime Environment).
— RAM (Random Access Memory) : Permet de stocker temporairement les données sensibles, son contenu est effacé à chaque nouvelle utilisation.

— EEPROM (Electrically Erasable Programmable Read Only Memory) : Ou mémoire non-volatile (NVM) est une mémoire applicative de faible capacité (1Ko à 256Ko pour les Java Cards) utilisée pour stocker les applets à exécuter, son contenu n'est pas effacé lors de nouvelles utilisations de la carte.
— I/O : Port d'entrée/sortie contrôlé par le microprocesseur afin de garantir des communications standardisées sous forme d'APDUs (Application Protocol Data Unit.
— Crypto-Processeur : Processeur dédié aux calculs cryptographiques.

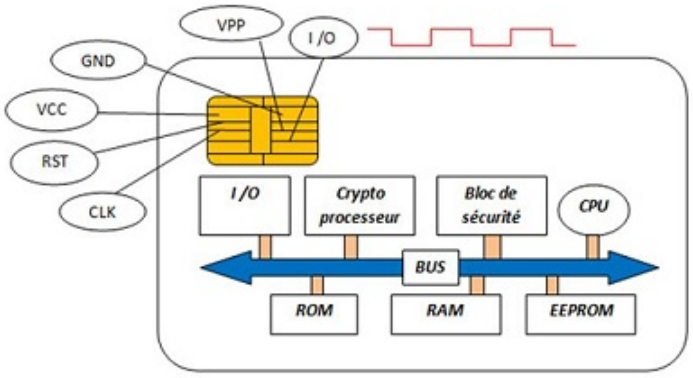


FIGURE 1 – Architecture d'une carte à puce asynchrone

Remarque : Le contact VPP n'est pas utilisé car la mémoire EEPROM possède son propre générateur de tension.

3. Cycle de vie d'une carte à puce

La fabrication : Inscription du système d'exploitation en mémoire ROM définissant les fonctionnalités de base de la carte, dont le contenu n'est pas chiffré. La connaissance de son code, bien que difficile ne doit pas rendre possible des attaques autorisant la lecture de la mémoire non volatile.

L'initialisation : Inscription en EEPROM des données communes à l'application.

La personnalisation Inscription en EEPROM des données relatives à chaque porteur.

L'utilisation :
— Envoi d'APDUs de commande à la carte.
— Traitement de ces commandes par l'OS de la carte :

- Commande reconnue : Traitement en interne de la commande ==> Lecture/écriture de données en EEPROM, et envoi d'un APDU de réponse.
- Commande inconnue : Renvoi d'un code d'erreur.

Fin de vie : Par invalidation logique, saturation de la mémoire, destruction physique, perte, vol, etc.[7]

II. Normes ISO/IEC-7816

Les technologies à base de cartes à puce sont normalisées par différents organismes afin d'assurer une interopérabilité entre les différents types de cartes et lecteurs.
L'ensemble de normes ISO/IEC-7816 a été défini par l'International Organization for Standardization (ISO) et l'International Electrotechnical Commission (IEC),il concerne les cartes à puce à contact et se décompose en 15 parties. Parmi les plus importantes nous pouvons citer :[1]

1. Norme ISO/IEC 7816-1

Définit les formats du contenant en plastique.
Il existe trois formats normalisés de cartes à puce : ID1 (celui des cartes bancaires), ID00 et ID000 (celui des cartes SIM).

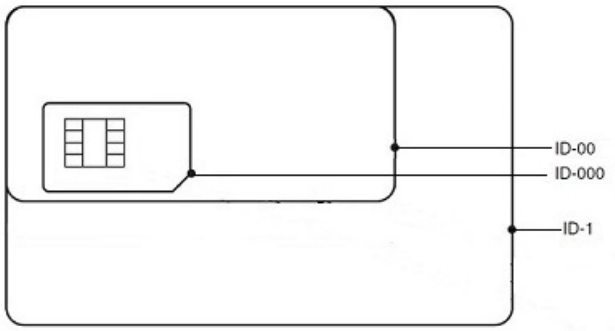


FIGURE 2 – Formats du contenant en plastique d'une carte à puce

Le fabricant ne produit qu'un format de cartes (ID1) et c'est aux clients de réduire la dimension selon leurs besoins. La norme ISO 7816-1 définit également les contraintes physiques auxquelles la carte doit résister.

2. Norme ISO/IEC 7816-2

Définit l'architecture de l'interface de contacts électroniques.

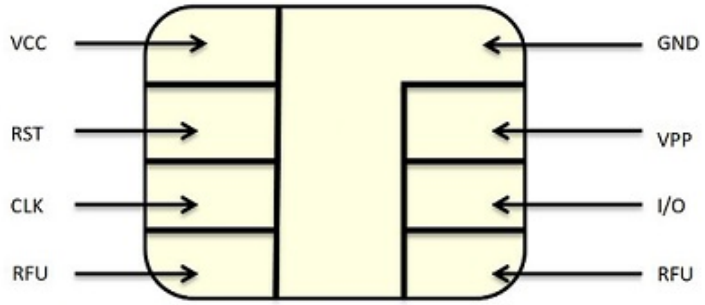


FIGURE 3 – Architecture de l'interface des contacts électroniques de la puce

VCC : Tension électrique de 3 à 5 Volts. **RST :** Ou Reset, il initialise le microprocesseur. **CLK :** Ou Clock, donne un signal d'horloge à la carte. **RFU :** Réservé pour une utilisation ultérieure (Reserved for Future Use). **GND :** ou Ground, masse électrique de la carte. **VPP :** Tension de programmation de la carte, inutilisée à présent. **I/O :** Entrées/sorties de données.

3. Norme ISO/IEC 7816-3

Parmi les caractéristiques données par la norme 7816-3 :

— La communication est du type Semi-duplex.
— La carte ne prend jamais l'initiative de l'échange d'informations, elle ne fait que répondre à des commandes, la syntaxe des APDUs de commande et de réponse est détaillée dans la partie suivante de la norme.
— Description des signaux électriques, tels que le voltage, la fréquence d'horloge qui est fournie par l'interface et la vitesse de communication.
— Description des protocoles de transmission (TPDU, Transmission Protocol Data Unit) avec T=0 : protocole orienté octet et T=1 : protocole orienté paquet.
— Précision des étapes de mise sous tension de la carte et de son arrêt .

4. Norme ISO/IEC 7816-4

Décrit les commandes de base des cartes à puce ainsi que les mécanismes de sécurité implantés pour accéder au contenu des fichiers. Elle définit donc le format des «paquets de données» échangés entre un lecteur et une carte : APDUs (Application Programming Data Units) de commande et de réponse :[6]

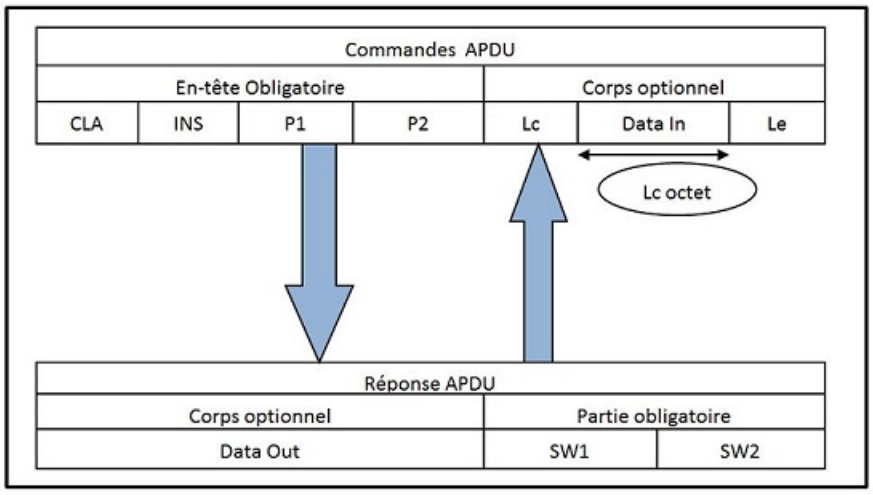


FIGURE 4 – Formats des APDUSs de commande et de réponse

— **CLA ou Classe (1 octet) :** Détermine le groupe d'instructions concernées.
— **INS ou Instruction (1 octet) :** Détermine l'opération à effectuer par la carte.
— **P1 (1 octet) et P2 (1 octet) :** Les deux paramètres que la carte reçoit avec la commande. Leur signification dépend de la commande envoyée.
— **Lc (1 octet) (Optionnelle) :** Détermine la longueur de la chaîne de réponse que le maître s'attend à recevoir en provenance de la carte ou bien le nombre d'octets qui suit la commande, dans ce cas

la carte s'attendra à recevoir Le octets supplémentaires (Data In) avant de traiter la commande dans sa totalité
- **Data (longueur variable) :** Une séquence d'octets correspondant aux données de réponse.
- **SW1 (1 octet) & SW2 (1 octet) :** (Status Word), état de traitement par la carte.

III. Outils de mise en œuvre de systèmes utilisant les cartes à puce

1. Interaction physique

1.1 Lecteurs

Un lecteur de carte est un appareil électronique qui permet d'établir un canal de communication entre les applications clientes sur l'ordinateur et la puce de la carte. Il existe plusieurs types de lecteurs :[2]

- **Les lecteurs simples :** Permettent de lire les informations stockées dans la puce de la carte en introduisant le code PIN par le biais du clavier du PC.
- **Les lecteurs pour ordinateur portable :** S'insèrent dans une fente PC-CARD des ordinateurs.
- **Les lecteurs avec PIN-Pad :** Permettent d'améliorer le niveau de sécurité en introduisant le code PIN à partir du lecteur, le code est directement envoyé à la carte suivant la norme PC/SC et ne sort donc pas du lecteur ce qui rend son interception très difficile.

FIGURE 5 – Lecteur de carte avec PIN-Pad

2. Interaction logicielle

2.1 Systèmes d'exploitation

Architecture : Schématiquement, un système d'exploitation d'une carte à puce comporte les éléments suivants :[8]
- Un bloc de gestion des ordres (APDUs).
- Une bibliothèque de fonctions cryptographiques.
- Un module de gestion de la RAM.
- Un module de gestion de la mémoire non volatile (EEPROM…), qui stocke les clés des algorithmes cryptographiques et les secrets partagés).
- Un bloc de gestion d'un système de fichiers localisé dans la mémoire EEPROM.
- Un module de gestion des événements indiquant une attaque probable.

Les systèmes d'exploitation des cartes à puce peuvent être classifiés en 2 types :[3]

Systèmes d'exploitation fermés (Native) : Les systèmes fermés sont généralement des systèmes mono-application, dédiés à un usage unique, ils sont de ce fait très performants et leur consommation de mémoire est optimisée.L'inconvénient est qu'un langage de bas niveau doit être utilisé pour développer des applications qui seront exécutées dans ces systèmes d'exploitation.

Systèmes d'exploitation ouverts (global) : Les systèmes d'exploitation ouverts sont des systèmes d'exploitation dont l'architecture permet aux développeurs d'utiliser des langages de haut niveau (C, C++, Java, Basic) pour écrire leurs applications. Ils ont généralement un interpréteur ou une machine virtuelle qui peut convertir le code de programme de haut niveau en un code machine ; ceci les rend indépendants du matériel mais le code en devient plus complexe et plus long, les besoins en ressources matérielles sont également plus importants. Les principales caractéristiques de ces systèmes d'exploitation peuvent être exprimées comme suit :

- Exécuter une application sur différentes cartes à puce.
- Séparer le système d'exploitation de l'application.
- Le support de différentes interfaces de communication et de protocoles.
- La possibilité de mettre à jour les fichiers, les programmes, et même certains modules du système d'exploitation (si le système d'exploitation est stocké sur une mémoire inscriptible comme une mémoire Flash) après la production de la carte.
- Plusieurs systèmes de contrôle d'accès et des mécanismes de sécurité.
- Utilisation d'un très fort pare-feu pour isoler les applications entre elles.

Les trois systèmes d'exploitation ouverts les plus populaires sont : **MultOS, Basic Card et Windows**.

2.2 Logiciels applicatifs

L'emploi de cartes à puces dans des systèmes informatiques nécessite deux types de logiciels :[8]

- Le logiciel de terminal qui contient l'application de l'utilisateur final, un logiciel de niveau système qui permet l'attachement de lecture de carte à puce à la plateforme de terminal, et un logiciel de niveau système spécifique qui permet l'utilisation des cartes à puce et qui supporte l'application de l'utilisateur final.
- Le logiciel de la carte qui contient le système d'exploitation et les applications de la carte (utilisé pour personnaliser une carte à puce pour une application particulière).

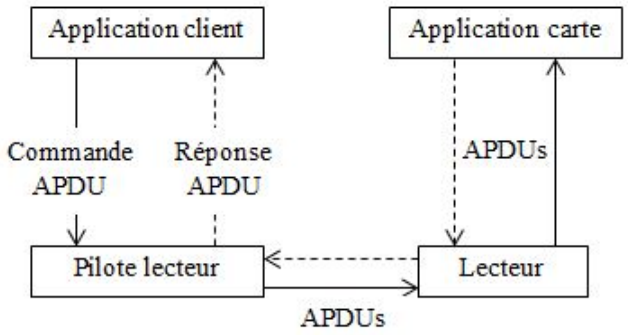


FIGURE 6 – Communication entre le terminal et la carte à puce

2.3 Cartes à puce multiapplicatives et la technologie Java Card

Une carte à puce multi-applicative est une carte qui peut contenir plusieurs applications de même domaine ou de différents domaines (identification, sécurité, paiement ...). Le but des cartes à puce multi-applicatives est de :

— Permettre à plusieurs applications de coexister.
— Partager des données entre plusieurs applications.
— Possibilité de charger/décharger des applications en cours de vie de la carte.

Grâce à leur architecture ouverte qui facilite l'intégration des cartes dans les applications et le développement de code dans les cartes à puce, les cartes à puce ouvertes sont très utilisées comme cartes multi-applicatives. Parmi les technologies les plus utilisées dans ce type de cartes nous avons : [9]

— MULTOS.
— WINDOWS FOR SMART CARD.
— JAVA CARD.

La technologie Java Card sera abordée plus en détail dans la section 4 de ce chapitre à la page 9.

3. Sécurité

Les constantes tentatives de vol et de corruption dont les données sensibles font l'objet ont mis en évidence la nécessité de mettre en place une protection fiable de celles-ci.
Cette protection consiste en l'utilisation de mécanismes de contrôle d'accès basés sur l'identification et l'authentification (vérification d'identité) de la personne ou entité.

3.1 Identité

Le concept d'identité s'est imposé dans la société moderne au vu de son importance dans l'attribution de droits d'accès, la propriété, ou le suivi de différentes actions. Apporter la preuve de son identité est devenu d'autant plus important avec le développement de technologies permettant les communications et transactions à distance.
L'identité numérique d'un individu est composée de données formelles (coordonnées, certificats…) et informelles (activité sur les réseaux sociaux et professionnels). Toutes ces bribes d'information composent une identité numérique plus globale qui caractérise un individu, sa personnalité, son entourage et ses habitudes.[4]

3.2 Authentification

Lorsque le lecteur et la carte à puce entament une communication d'informations (appelée session), une vérification d'identité doit se faire afin d'éviter les usurpations d'identité dans les deux sens.
L'authentification désigne cette vérification d'identité, elle se fait à trois niveaux :

3.2.1 Authentification de l'utilisateur :

Permet d'authentifier le détenteur de la carte, elle se fait grâce aux facteurs suivants :
— Ce qu'il sait : Mot de passe ou code PIN.
— Ce qu'il possède : Carte, jeton, etc.
— Ce qu'il est : Données biométriques telles que l'empreinte digitale, vocale ou rétinienne.
L'association de plusieurs facteurs assure une protection accrue des données.[4]

3.2.2 Authentification du terminal :

Permet de s'assurer que le terminal en question appartient au système et est autorisé à consulter/modifier les informations contenues dans la carte à puce. Le procédé d'authentification peut employer une seule clé pour le cryptage et le décryptage (symétrique) ou deux clés différentes (asymétrique).

Authentification unilatérale par cryptage asymétrique : Le concept du cryptage asymétrique consiste en l'utilisation d'une clé publique pour le cryptage du message et d'une autre privée pour le décryptage, il permet de s'assurer que seul le destinataire (détenant la clé privée) puisse comprendre le message. Lors de l'authentification, la carte à puce génère un message aléatoire, le crypte grâce à la clé publique du terminal et la lui envoie. Le terminal à son tour le décrypte et renvoie le message décrypté en clair, la comparaison entre le message initial et celui décrypté permet d'authentifier le terminal.[5]

3.2.3 Authentification de la carte à puce :

L'architecture d'une Smart Card permet de stocker les informations d'identification dans la mémoire et d'effectuer certains calculs cryptographiques grâce au microprocesseur. L'authentification de la carte à puce peut être effectuée grâce au procédé expliqué précédemment ou en employant les signatures digitales :[5]

Signatures digitales Aussi appelées signatures numériques, elles reprennent le principe de la signature manuelle et sont utilisées pour identifier l'auteur d'un document, elles permettent donc la non répudiation (La personne ne peut pas nier son émission) car elles sont uniques. Elles peuvent aussi garantir l'intégrité d'un document (non altération).
L'obtention d'une signature et sa vérification s'appuient sur le principe de l'authentification asymétrique, la clé privée permet de générer une signature et la clé publique de la vérifier. Le calcul d'une signature digitale nécessite au préalable l'utilisation de fonctions de hachage : ce procédé consiste en la compression d'un document en un bloc de taille fixe, l'opération inverse est impossible ; le but de l'utilisation du hachage est d'obtenir une signature de taille fixe.
Lors de la personnalisation de la carte à puce, les informations spécifiques à la carte sont chargées dans la puce ainsi qu'une signature digitale calculée à partir de ces données grâce à une clé secrète. Elles sont toutes les deux impossibles à modifier pendant toute la durée de vie de la carte.
La carte envoie les informations citées précédemment et la signature au moment de la connexion au terminal, le terminal utilise la clé publique pour décrypter la signature numérique et compare le résultat avec le haché des informations envoyée par la carte, s'ils sont identiques, la carte est authentifiée.

3.3 Certificats

L'utilisation de signatures numériques et de clés a soulevé le problème d'authenticité des clés, toute personne peut vérifier une signature numérique au moyen d'une clé publique mais cette clé doit être à son tour authentifiée. Pour cela, la clé publique doit être certifiée par un organisme de confiance appelé « autorité de certification ».La combinaison d'une clé publique signée par cette autorité, de la signature numérique et d'autres paramètres additionnels tels que les algorithmes de hachage et de génération de signature, est appelée « certificat ». Une autre entité impliquée dans le processus de certification est le centre de confiance ou Trust Center, il est chargé de mettre à disposition du public un fichier de certificats que toute personne peut consulter afin de récupérer la clé publique correspondant à un document signé. Le processus de certification est détaillé dans la norme X.509 ou ISO/IEC 9594-8.

VI. La technologie Java Card

1. Présentation

La technologie Java Card définit une plate-forme pour cartes à puce sécurisée, portable et multi-application qui permet la gestion et la mise en œuvre de plusieurs programmes (applets) écrits en Java .

2. Architecture

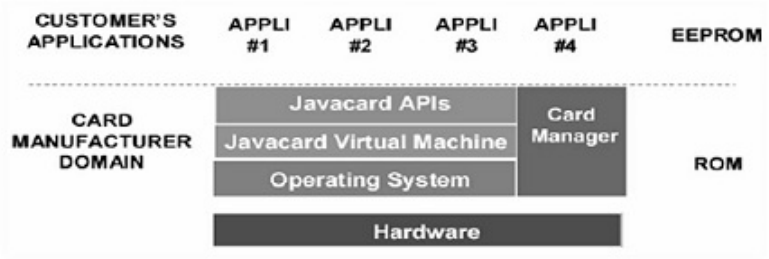


FIGURE 7 – Architecture d'une Java Card

2.1 Système d'exploitation

Le système d'exploitation : contient un ensemble de fonctions de bas niveau qui permettent au Java Card Runtime Environnent (JCRE) de gérer les ressources physiques de la carte telles que les entrées-sorties, la gestion de la mémoire ou le coprocesseur cryptographique. Parmi les OS qui supportent la technologie Java Card nous avons : **Athena, IDCOR, JCOP**[3]

2.2 Machine virtuelle Java (interpréteur) JCVM

JCVM est le cœur de l'architecture Java Card. C'est le moteur d'exécution des Applets (Cardlets) qui garantit la sécurité et gère le partage des données. Il exécute les programmes en convertissant le code pré-compilé en appel aux méthodes natives du système d'exploitation.[5]

2.3 Le Framework Java Card (Standard Class Libraries)

Le Java Card Framework est un ensemble d'APIs (Application Program Interface) qui contiennent l'ensemble de classes employées pour créer une applet.
Le Framework Java Card contient quatre packages :[7]

- **javacard.framework :**
  Package de base pour les cartes Java Card. Il définit des classes telles que : Applet et PIN, qui sont les blocs de construction fondamentaux pour les programmes Java Card, et les classes APDU, Systeme et Util, qui fournissent des services système aux programmes Java Card tels que le traitement des APDUs et le partage des objets.
- **javacardx.framework :**
  Ce package fournit une conception orientée objet pour un système de fichiers compatible avec la norme ISO 7816-4. Il supporte les fichiers élémentaires (EF), les fichiers dédiés (DF) et les fichiers orientés APDU comme spécifié dans ISO-7816.
- **javacardx.crypto et javacardx.cryptoEnc :**
  Prennent en charge les fonctionnalités cryptographiques requises dans les cartes à puce.

2.4 Les applets

Les applications Java Card sont représentées sous forme d'applets articulées autour de l'héritage d'une classe Applet du package javacard.framework, elles sont identifiées grâce à leur AID. Leur chargement sur la carte à puce se fait en trois étapes :[10]

- **La compilation de 'java file' :**
  La compilation des programmes Java Card commence par l'utilisation d'un compilateur Java du JDK standard. Les classes de l'API Java Card doivent être importées et définies dans le ClassPath du compilateur pour pouvoir faire l'édition des liens. Le fichier issu de la compilation 'bytecode Java' est appelé « Class file ».

- **La conversion du 'class file' (JCDK) :**
  La simple compilation est laissée à la charge au compilateur Java classique, seule la conversion est prise en charge par le JCDK Java Card, Le fichier issu de la conversion 'bytecode JavaCard' est appelé « cap file ».

  La vérification est faite lors de la conversion :

  - Vérifier que les images chargées des classes Java sont correctes.
  - Vérifier la conformité du code par rapport au langage Java Card.
  - Initialiser les variables statiques.
  - Convertir les références symboliques des classes, méthodes et champs vers une forme plus compacte et mieux adaptée à la carte.
  - Optimiser le bytecode au niveau du chargement et de l'édition des liens.
  - Allouer l'espace nécessaire et créer les structures de données pour représenter les classes auprès de la machine virtuelle.

- **Le chargement du 'cap file' :**

  Une fois le code converti, nous nous retrouvons avec un fichier .cap, une étape supplémentaire est nécessaire pour que l'applet passe du monde extérieur vers la carte à puce : c'est le chargement qui fait partie de la procédure d'installation de l'applet, L'installation d'une applet se fait, alors que le JCRE tourne, en deux phases :

  A. Chargement du code de l'applet (cap file) sur la carte
  B. Création d'une instance de l'applet

  Pour installer l'applet, l'installeur prend le fichier .cap et télécharge le contenu du fichier sur la carte. L'installeur écrit le contenu dans l'EEPROM et fait les liens entre les classes dans le fichier .cap et les classes résidant sur la carte. Il crée et initialise les données qui seront utilisées par le JCRE en interne pour se servir de l'applet. Si l'applet nécessite plusieurs packages, chaque fichier .cap correspondant à un package sera chargé sur la carte.

V. Conclusion

Une carte à puce, personnalisée avec le consentement de son porteur et mettant en œuvre les conditions suffisantes de sécurité, peut stocker des données et de les préserver contre la plupart des attaques et de mettre en œuvre les procédures d'identification, authentification et de signature électronique.

Ces capacités de stockage, de calcul et de protection de données ainsi que leur portabilité ont fait la popularité des cartes à puce, elles sont à présent utilisées dans plusieurs applications fonctionnant selon certaines normes et technologies : téléphonie, carte bancaire, carte de santé ...etc. Elles représentent donc un élément important dans les systèmes informatiques, notamment les systèmes sécurisés.

Chapitre II :
Mise en œuvre de cartes à puce

I. Introduction

La mise en œuvre d'une carte à puce dans un système informatique pour notre projet passe par plusieurs étapes :

- Précision des mécanismes de chiffrement.
- Création et configuration de l'autorité de certification (CA).
- Développement de l'applet.
- Création d'un outil "Reader Manager".
- Installation de l'applet.
- Initialisation de la carte.
- Utilisation.

II. Précision des mécanismes de chiffrement

Lors du procédé d'authentification, la carte et le terminal emploient le chiffrement asymétrique RSA (Intitiales des trois inventeurs Rivest, Shamir, Adleman), avec des paires de clés de 2048 bits.

RSA : La force du chiffrement RSA réside dnas la difficulté de factoriser le produit de deux grands nombres premiers.Les deux clés se composent chacune d'un exposant et d'un modulo, le modulo est commun aux deux clés.La paire de clés est générée comme suit :[11]

- Calcul du modulo *n* à partir du produit de deux grands nombres premiers *p* et *q*.
$$n = p.q$$
- Choix de l'exposant de la clé publique *e* qui est premier avec le produit *(p-1)(q-1)*, le nombre de Fermat 65537 est souvent utilisé.
- Choix de l'exposante de la clé privée *d* tel que :
$$e.d \equiv 1 \bmod (p-1)(q-1)$$

Le résultat *C* du cryptage du message *M* s'obtient grâce à la formule :

$$C = M^e \bmod n$$

Le résultat *M* du décryptage de *C* s'obtient grâce à la formule :

$$M = C^d \bmod n$$

Le chiffrement RSA est assez complexe et ne peut être employé pour crypter tous les échanges vu la rapidité de calcul limitée de la carte à puce. Par souci de gain de temps, les échanges suivant l'authentification sont donc sécurisés à l'aide du chiffrement symétrique AES (Advances Encryption Standard) avec une clé de 128 bits qui sera générée à chaque nouvelle session.

AES : Le chiffrement AES s'appuie sur un algorithme de calcul matriciel qui effectue un cryptage et décryptage par blocs de 128 bits à l'aide de clés de 128, 192 ou 256 bits. Lorsque la longueur du message à crypter n'est pas multiple de 128, nous devons recourir au *bourrage* ou "*padding*".
L'AES est jugé rapide et facile à implémenter, il est également le plus sûr des algorithmes de chiffrement symétriques[4]. Toutes ces caractéristiques en font le meilleur choix pour sécuriser une session d'échange d'informations.

L'algorithme de hachage utilisé est *SHA1* (Secure Hash Algorithm), il est combiné à *RSA* pour générer des signatures digitales.
Ces mécanismes de chiffrement sont utilisés dans les deux applications développées.

III. Création et configuration de l'autorité de certification racine(CA)

Dans la plupart des organisations, une autorité de certification racine est le premier service de rôle des services de certificats Active Directory (AD CS, Active Directory Certificate Services) installé, elle peut être la seule autorité de certification déployée dans une infrastructure à clé publique de base.
Pour installer une autorité de certification racine l'utilisateur doit appartenir au groupe local *Administrateurs* ou à un groupe équivalent. S'il s'agit d'une autorité de certification d'entreprise, il est nécessaire d'appartenir au minimum au groupe Admins du domaine ou à un groupe équivalent.[12]
L'installation d'une autorité de certification racine dans un environnent Windows server 2012 se fait selon la procédure suivante :[13]

- Ajout du rôle *Services de certificats Active Directory* (AD CS) avec le service *Autorité de certification* au serveur, *WIN-N2J430TRRNA* dans le domaine *USTHB.INFO* dans notre cas.
- Configuration la CA en tant qu' *Autorité de certification d'entreprise* du domaine *USTHB.INFO* et spécifier que c'est une CA racine.
- Spécification de l'algorithme de chiffrement utilisé (RSA), de la taille de la clé (2048 bits) et de l'algorithme de hachage (SHA1).

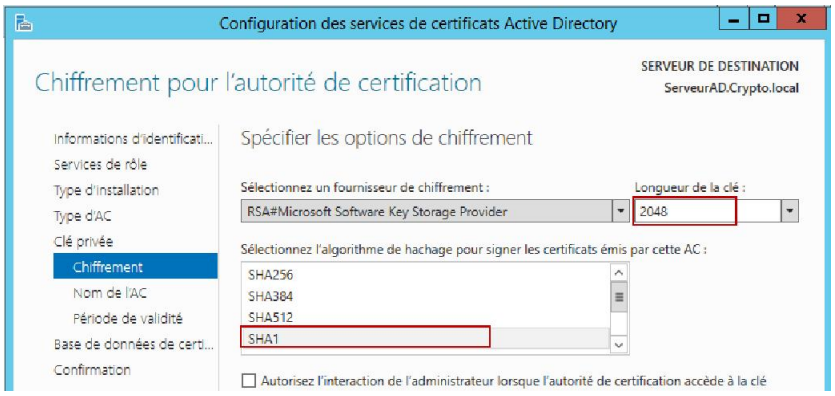


FIGURE 8 – Algorithmes de chiffrement, hachage et taille de clé

— Indication des informations concernant l'AC telles que son nom commun (*USTHB-WIN-N2J430TRRNA* dans notre cas)et sa période de validité (5 ans).
— Dupliquer le modèle prédéfini de certificats *SmartCard User* (norme X509) pour créer un modèle personnalisé *SmartCardPFE*. **Note :** La structure des certificats X509 est détaillée dans l'annexe A à la page 35.

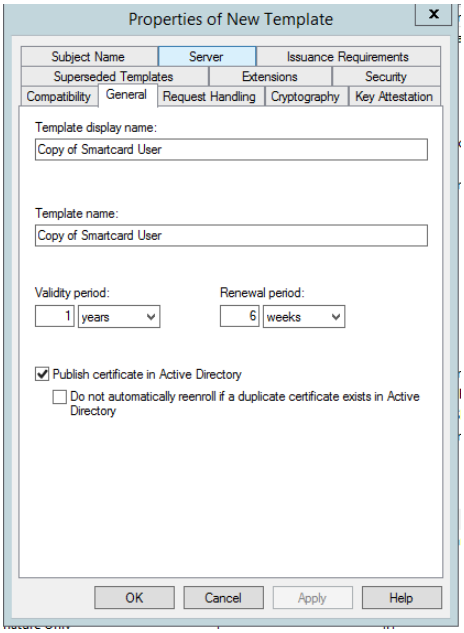


FIGURE 9 – Personnalisation du modèle de certificats

— Modification des autorisations des membres du groupe des administrateurs qui gèrent l'initialisation de la carte dans l'onglet *Sécurité* des propriétés du modèle pour qu'ils puissent demander des certificats (inscription).
— Ajout du modèle *SmartCardPFE* à la liste des modèles de certificats à délivrer.

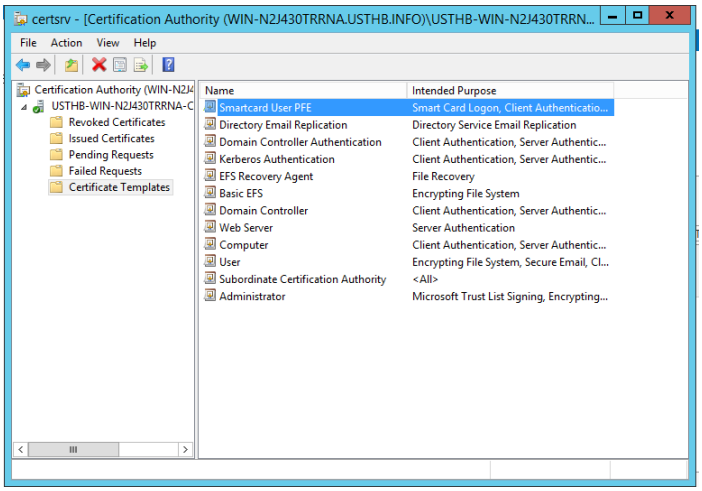


FIGURE 10 – Ajoute du modèle en tant que modèle de certificat à délivrer

IV. Développement de l'applet et application terminal

L'étape de développement concerne les deux applications de carte (Java Card) et de terminal, elle se fait au moyen d'un environnement de développement Java tel qu'Eclipse.

1. Applet Java Card

Permet de gérer les données contenues dans la carte et de communiquer avec le terminal, son développement requiert un JCDK (Java Card Development Kit). Une applet Java Card doit contenir les méthodes de base suivantes :[14]

- **Méthode install() :** Appelle le constructeur de l'applet et permet de créer et d'initialiser une instance de cette applet dans la carte.
- **Méthode select() :** Une fois initialisée, l'applet reste suspendue tant qu'elle n'a pas été sélectionnée par le JCRE ; cette action a lieu lorsque l'applet reçoit l'APDU de sélection.
- **Méthode deselect() :** Est appelée lorsqu'une autre application est sélectionnée dans la carte.
- **Méthode process() :** Cette méthode permet de traiter tous les APDUs reçus par la carte. Le traitement se fait par rapport à la classe, l'instruction, les paramètres et les données reçues. La récupération se fait avec les commandes :

```
byte [] buffer = apdu.getBuffer();
byte Classe=buffer[ISO7816.OFFSET_CLA];
byte Instruction=buffer[ISO7816.OFFSET_INS];
byte Parametre1=buffer[ISO7816.OFFSET_P1];
byte Parametre2=buffer[ISO7816.OFFSET_P2];
byte [] Data =new byte[buffer[ISO7816.OFFSET_LC]];
Util.arrayCopy(buffer, (short)ISO7816.OFFSET_CDATA,Data,(short)0,(byte)
    buffer[ISO7816.OFFSET_LC]);
```

Avec :

- `ISO7816.OFFSET_X` la position de chaque champ dans l'APDU.
- `Util.arrayCopy` l'instruction permettant de copier le contenu du buffer à partir de la position des données dans le tableau `Data`, la longueur du contenu à copier étant celle des données.

Une fois l'applet programmée, nous devons lui assigner ainsi qu'à son package des AIDs uniques de 5 à 16 octets pour pouvoir les différencier d'éventuels applets/packages installés sur la même carte. Après assignation de l'AID, le package devra être compilé afin de produire un fichier *.cap* qui sera utilisé pour installer l'applet.[15]

2. Application terminal

Permet de communiquer avec la carte au moyen d'APDUs et d'interpréter les résultats obtenus. L'application terminal est développée en langage Java en utilisant le package *javax.smartcardio* et emploie les commandes de base suivantes :

- **Connexion au lecteur** :

```
TerminalFactory tf = TerminalFactory.getDefault();
CardTerminals list = tf.terminals();
cad = list.getTerminal("OMNIKEY CardMan 3821 0");
if(cad == null){
  JOptionPane.showMessageDialog(null, "Branchez le lecteur !");
  System.exit(1);
  }
```

— **Connexion à la carte** :

```
c = cad.connect("T=1");
canal = c.getBasicChannel();
```

— **Envoi de commandes APDU à la carte et récupération des APDUs de réponse** :
Par exemple l'APDU de sélection de l'applet dont l'AID est : `0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00` est envoyé via les commandes :

```
CommandAPDU selectApdu = new CommandAPDU(new byte [] {(byte)0x00,(byte) 0xA4,(byte) 0x04,(byte) 0x00,(byte) 0xb,(byte) 0x01,(byte) 0x02,(byte) 0x03,(byte) 0x04,(byte) 0x05,(byte) 0x06,(byte) 0x07,(byte) 0x08,(byte) 0x09,(byte) 0x00,(byte) 0x00,(byte) 0x7F} );
ResponseAPDU selectReponse = canal.transmit(selectApdu);
```

Nous pouvons vérifier que l'opération a bien été effectuée et traiter d'éventuelles erreurs en réupérant le Status Word avec la commande :

```
short StatusWord=selectReponse.getSW();
```

La récupération des données envoyées par la carte se fait avec la commande :

```
byte[] data=selectReponse.getData();
```

V. Création d'un outil "Reader Manager"

ReaderManager est un programme écrit en C++ et qui se base sur la partie 10 de la norme PC/SC, le rôle principal de ce programme est de communiquer avec le lecteur de carte à puce pour la lecture du PIN d'une façon sécurisée. ReaderManager fait appel aux fonctions natives de Windows définies dans la norme PC/SC : SCardEstablishContext() ,SCardListReaders() ,ScardConnect(), SCardTransmit() ,ScardControl() . . . etc. Ces fonctions sont disponibles dans la bibliothèque WinSCard.lib qui fait partie du SDK de Windows[16].ReaderManager se compose essentiellement des fonctions :

— **ecoute()** : Permet la création d'une socket d'écoute continue sur le port *27015*.
— **select ()** : Permet d'envoyer l'APDU de sélection l'applet qui effectue la vérification.
— **verifyPIN()** : Permet l'envoi de la commande de lecture de PIN et retourne la réponse de la carte.

Fonctionnement :

— ReaderManager fait l'écoute à travers une socket en attendant une demande de vérification de PIN envoyée par le programme principal Java ; une fois la demande reçue, il se connecte à la carte et sélectionne l'applet qui effectuera la vérification du PIN.
— Il envoie ensuite une commande au lecteur avec la fonction ScardControl() avec comme paramètre le code de contrôle 0x06 qui signifie « FEATURE_ VERIFY_ PIN_ DIRECT » , et le buffer qui va contenir la réponse de la carte après la vérification de PIN ainsi que d'autres paramètres qui définissent le message à afficher dans le lecteur, la langue, la méthode de validation de PIN, le format de PIN , APDU d'encapsulation de PIN. . . etc.
— L'utilisateur tape son code PIN dans le clavier du lecteur, le PIN est encapsulé dans un APDU de vérification et envoyé à la carte pour vérification.
— ReaderManager récupère la réponse de la carte puis envoie cette réponse à l'application principale à travers la connexion de socket précédemment établie.

Notes :

— Le PIN ne sort jamais du lecteur, il est directement envoyé à la carte.
— ReaderManager fonctionne en arrière plan et démarre automatiquement à l'ouverture de session.

VI. Installation de l'applet

L'installation de l'applet dans la carte se fait à l'aide de GlobalPlatformPro, un outil Open Source qui permet de charger et de gérer les applets sur des cartes à puce Java Card, il est disponible en ligne de commande et permet de réaliser des opérations telles que :[17]

— Lister les packages installés dans la carte.
— Installer un package.
— Supprimer un package et toutes les applets de ce package.

Notes :

— Cet outil sera intégré aux applications et son utilisation sera transparente pour l'utilisateur final.
— Les commandes utilisées avec les résultats retournés sont détaillés dans l'Annexe A à la page 33.

VII. Initialisation de la carte

Une fois l'applet installée, la carte doit être initialisée avec les informations de l'utilisateur, cette initialisation se fait en plusieurs étapes et n'est possible qu'à partir des terminaux d'administration :

1. Sélection de l'applet :

Après son installation, une applet doit être sélectionnée pour fonctionner et communiquer avec le terminal. La sélection se fait avec l'identifiant unique (AID) de l'applet défini lors de son développement.

2. Authentification du terminal :

La carte ne peut être initialisée par n'importe quel terminal, il faut donc l'authentifier. La clé publique commune à tous les terminaux étant présente dans l'applet dès son installation ; la carte génère un message aléatoire, le crypte avec la clé publique RSA du terminal et la lui envoie, le terminal décrypte le message avec sa clé privée et le renvoie en clair à la carte qui le compare avec le message initial.

3. Génération d'une clé de session :

Une fois le terminal authentifié, la carte génère une clé de session AES de 128 bits qui permettra de crypter les échanges et l'envoie au terminal après l'avoir crypté avec la clé publique RSA de celui-ci. Les échanges entre la carte et le terminal durant cette session seront cryptés gâce à cette clé.

4. Initialisation des infos utilisateur :

L'initialisation des infos de l'utilisateur se fait à partir d'un terminal d'administration, ces infos désignent le porteur comme le nom, prénom...etc ainsi qu'un identifiant unique comme le numéro de sécurité sociale ou le matricule pour un étudiant, elles sont cryptées et envoyées à la carte pour y être stockées.

5. Initialisation du certificat et de la clé privée RSA :

Cette opération se déroule sur un terminal d'administration et nécessite OpenSSL, un outil Open Source qui implémente les deux protocoles SSL (Secure Socket Layer) et TLS (Transport Layer Security) ainsi qu'une librairie cryptographique ; il offre la possibilité de générer et d'effectuer des opérations sur des clés cryptographiques et des certificats.[18] L'opération se déroule en plusieurs étapes :

— Le terminal génère une paire de clés RSA qui sera attribuée à la carte, et utilise l'identifiant unique de l'utilisateur (matricule de l'étudiant, par exemple 201200007196) afin de générer une requête de certificat qui sera envoyée à l'autorité de certification.

— Le terminal envoie ensuite la requête contenant la clé publique générée à l'autorité de certification au niveau du serveur qui lui renvoie le certificat demandé.
— Ce certificat sera transmis à la carte ainsi que le modulo et exposant de la clé privée récupérés à partir du fichier *.pem* contenant la paire de clés et cryptée à l'aide de la clé de session AES.

Les commandes utilisées avec les résultats retournés sont détaillés dans l'Annexe A à la page 34.

6. Génération de la signature :

Après avoir reçu le certificat ainsi que le modulo et l'exposant, la carte initialise sa clé privée et génère une signature à partir des informations de l'utilisateur en utilisant l'algorithme de hachage *SHA1* combiné à *RSA*, cette signature permettra d'authentifier la carte par la suite.

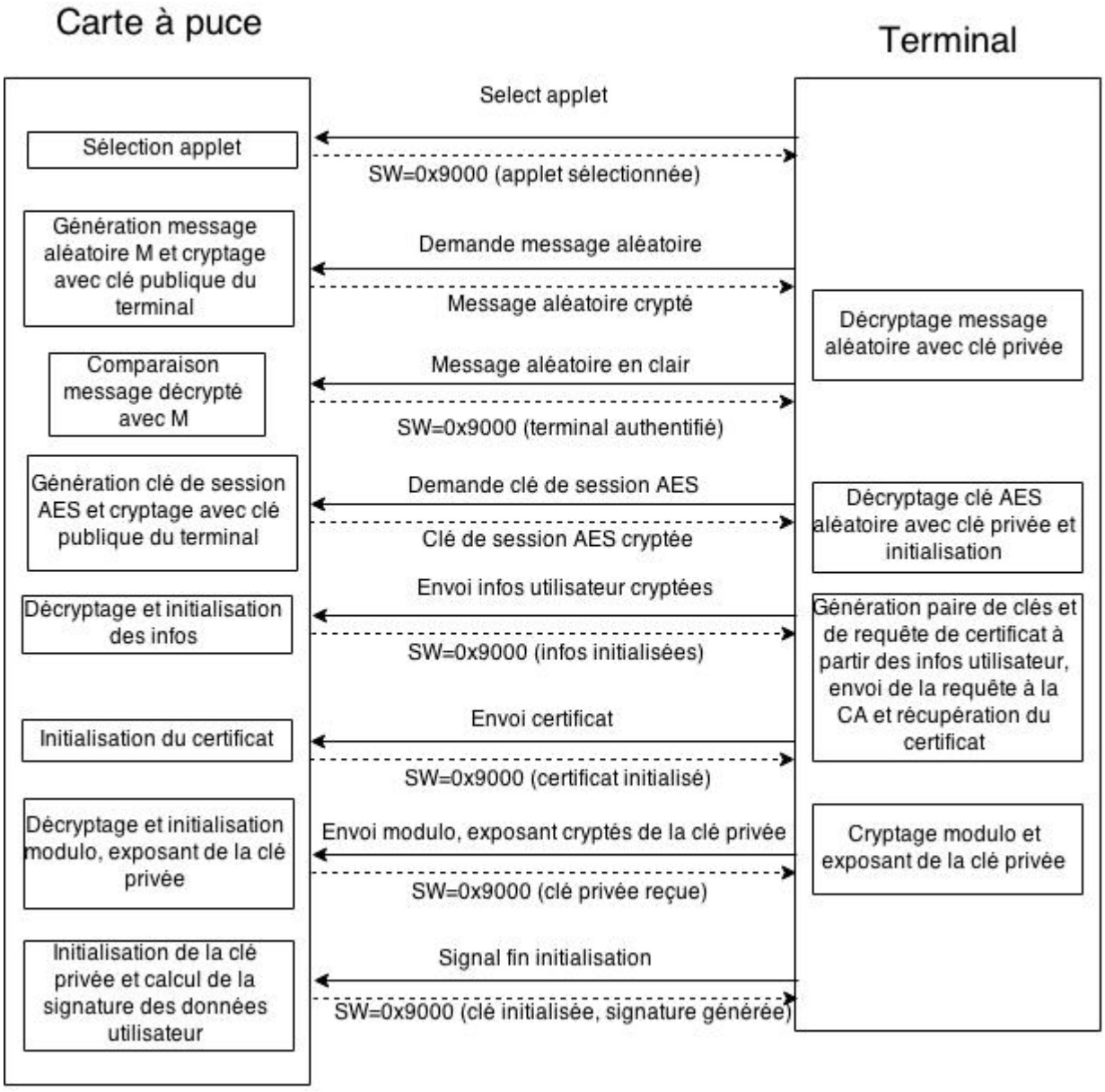


FIGURE 11 – Étapes de l'initialisation de la carte à puce

VIII. Utilisation

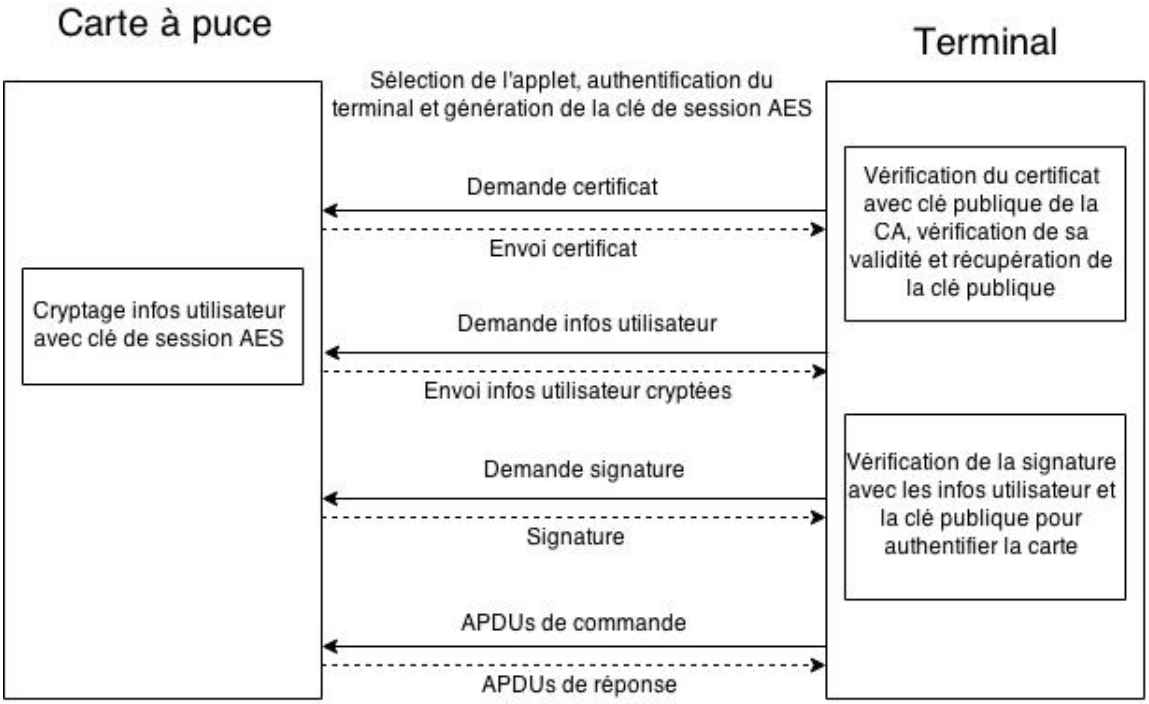


FIGURE 12 – Utilisation de la carte à puce

1. Sélection, authentification du terminal et génération d'une clé de session AES :

Ces trois opérations se déroulent de la même manière que lors de l'initialisation de la carte.

2. Authentification de la carte :

2.1 Récupération du certificat et de la clé publique :

Après avoir généré la clé de session AES, la carte envoie son certificat au terminal. Ce dernier le convertit au format *.pem* avec la commande :

```
openssl x509 -in certificat.cer -inform DER -out certificat.pem
```

Il vérifie ensuite sa validité ainsi que son authenticité grâce à la commande :

```
openssl verify -CAfile certificat-CA.pem certificat.pem
```

Où le fichier *certificat-CA.pem* est le fichier contenant le certificat de l'autorité de certification au format *.pem*. Cette commande permet de vérifier que ce certificat est en cours de validité et qu'il a bien été émis par l'autorité dont le certificat a été mis en paramètre.Si le certificat est authentifié et en cours de validité, la commande retourne comme résultat :

```
certificat.pem: OK
```

Une fois le certificat authentifié, le terminal peut récupérer la clé publique de la carte à partir de ce dernier en traitant le fichier *.pem*.

2.2 Récupération des données, leur signature et comparaison :

La carte envoie ensuite la signature calculée lors de l'initialisation ainsi que les données utilisées pour la générer (ces données sont cryptées à l'aide de la clé de session AES), le terminal peut vérifier cette signature avec la clé publique et les données initiales. Ceci permet de vérifier que cette signature a bien été générée par le détenteur de la clé privée correspondante et donc d'authentifier la carte.

3. Authentification du porteur de la carte :

L'authentification du porteur se fait par la saisie dans le lecteur de son code PIN défini lors de l'initialisation de la carte ; si le porteur saisit un code erroné plusieurs fois, la carte se bloque et ne peut être débloquée qu'en réinitialisant le PIN par un terminal d'administration.

4. Échange d'informations :

L'échange d'informations se fait selon le type et l'utilisation de l'application terminal qui envoie des APDUs de commande à la carte. Cette partie sera détaillée dans le troisième chapitre à la page 21.

5. Renouvellement du certificat :

La durée de validité d'un certificat étant limitée à une année, celui-ci doit être doit être soit renouvelé soit remplacé par un nouveau certificat. Le nouveau certificat ainsi que la nouvelle clé privée seront envoyés à la carte qui recalculera la signature à partir des informations utilisateur.Cette opération n'est possible qu'avec des terminaux d'administration.

6. Réinitialisation du PIN :

Si l'utilisateur introduit un code PIN erroné plusieurs fois, la carte se bloque et le code PIN doit être réinitialisé avec une nouvelle valeur et un nombre d'essais remis à zéro.Cette opération n'est possible qu'avec des terminaux d'administration.

IX. Conclusion

La mise en œuvre d'une carte à puce dans un système informatique a nécessité dans ses diverses étapes l'usage de plusieurs outils tels que GlobalPlatformPro, OpenSSL et ReaderManager ; de programmes écrits en Java, Java Card et en C++ ainsi que la création et la configuration d'une autorité de certification sur un serveur.

Des mécanismes de sécurité tels que le chiffrement et la signature ont également été mis en place afin d'assurer l'authentification, la confidentialité et l'intégrité des données.

Chapitre III : Applications

I. Introduction

La carte à puce peut être employée dans plusieurs domaines tels que la santé, le contrôle d'accès, le commerce électronique, les applications d'administration ...etc.
Nous avons choisi de la mettre en œuvre dans deux types d'applications :

— Une application commerciale de restaurant universitaire.
— Une application administrative pour permettre aux étudiants de récupérer leurs différents documents.

II. Application commerciale

L'application commerciale est une application porte-monnaie pour restaurant universitaire, les étudiants peuvent recharger leur balance et en débiter le montant voulu pour acheter des repas.

1. Fonctionnalités :

1.1 Administration

Ces actions ne peuvent être effectués que par un terminal d'administration :

— **Initialistation :** Chaque étudiant souhaitant bénéficier de cette carte peut la demander en se munissant d'un certificat de scolarité. La carte sera d'abord initialisée avec ses informations personnelles : PIN, nom, prénom, matricule, filière ; ainsi qu'une clé privée accompagnée d'un certificat pour les besoins d'authentification. Le certificat n'est valide que pour une année universitaire et doit être renouvelé à chaque rentrée pour pouvoir continuer à utiliser la carte.
— **Gestion du PIN :** Permet de réinitialiser le PIN du propriétaire avec une nouvelle valeur, le nombre d'essais erronés est remis à zéro.
— **Gestion des certificats :** Permet de renouveler le certificat de la carte pour une année supplémentaire.

1.2 Caisse

— **Authentification du porteur** : L'étudiant doit d'abord introduire son code PIN pour être authentifié :
  - S'il introduit un PIN invalide trois fois, la carte se bloque et le PIN doit être réinitialisé par un terminal d'administration.
  - Si le PIN est valide, les informations du propriétaire de la carte ainsi que le montant de sa balance et l'historique de la dernière transaction s'affichent.

— **Recharge :** Lorsque l'étudiant désire effectuer des achats, il choisit les produits à ajouter au panier virtuel en précisant la quantité ; le montant total du panier augmente ou diminue selon l'ajout ou suppression de produits.
— **Débit :** Lorsque l'étudiant désire effectuer des achats, il choisit les produits à ajouter au panier virtuel en précisant la quantité ; le montant total du panier augmente ou diminue selon l'ajout ou suppression de produits.
Le montant est ensuite débité de la balance à condition que la somme totale ne dépasse pas la valeur de la balance.
— **Historique :** Après chaque transaction réussie, l'historique est actualisé avec le type, montant et date de cette dernière.

2. Applications terminal :

Les deux types de fontionnalités étant indépendants et nécessitant des droits différents, nous pouvons les séparer en deux applications terminal utilisées chacune dans un contexte différent :

— **Admin Tool** : Permet à un administrateur d'initialiser une carte à puce vierge et personnaliser cette carte avec les informations de son porteur elle permet aussi le renouvellement de certificat de la carte est la réinitialisation de code PIN.
— **USTHB Moneo-Resto** : Est en quelque sorte une caisse qui permet d'authentifier le client par ca carte à puce et son code PIN, afficher les informations de client, recharger le crédit de client, débiter le montant des achats et enfin imprimer un ticket.

Chaque application peut être utilisée respectivement par un groupe d'utilisateurs ayant les droits d'exécution spécifiés dans l'Active Directory de notre serveur.

2.1 L'application Admin Tool :

FIGURE 13 – Écran d'accueil de l'application Admin Tool

Nous pouvons classifier les classes de l'application « Admin Tool » en deux catégories :

2.1.1 Les classes d'affichage :

— **La classe Start** : Le point d'entrée de l'application « Admin Tool » (contient la méthode main) elle permet l'affichage pour l'administrateur du menu principal qui contient les choix suivants : gérer le contenu de la carte à puce, renouveler l'abonnement Moneo-Resto, personnaliser la carte avec les informations de son porteur, Réinitialiser le code PIN.

Note : la classe Start lance un Timer pour vérifier la présence de la carte d'une façon continue, si la carte n'est pas présente dans le lecteur tous les boutons sont désactivés.

— **La classe Écran** : Permet de gérer le contenu de la carte à puce en faisant appel à l'outil GlobalPlatformPro (lister les packages installés, installer un package, supprimer un package...).

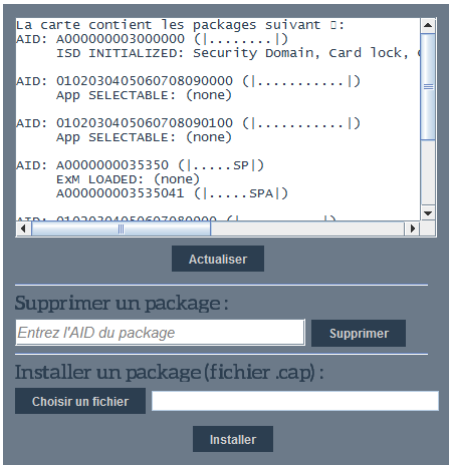


FIGURE 14 – Gestion du contenu de la carte à puce

— **La classe Formulaire** : Permet l'affichage d'un formulaire à remplir pour personnaliser la carte avec les informations de son porteur. **Note :**Une vérification de la validité des champs est effectuée avant l'envoi des informations.

— **La classe NewPIN** : Permet la réinitialisation du code PIN du client dans le cas où le nombre d'essai de PIN est nul.

FIGURE 15 – Personnalisation de la carte avec les infos utilisateur

2.2.2 Les classes de gestion :

— **La classe CardManagement** : Est la classe la plus importante dans le programme car c'est celle qui est responsable de la communication directe avec la carte, les autres classes ne font qu'appeler

les méthodes de cette classe. Les méthodes et attributs principaux de cette classe sont détaillés dans l'annexe B à la page 36.

— **La classe CertificateManagement** : Permet de créer un fichier qui contient la clé privée et un fichier qui contient la demande de certificat de cette clé pour le client, puis envoie la demande à l'autorité de certification, cette classe fait appel à des commandes de l'outil OpenSSL pour gérer les certificats.
Note : la méthode *public byte [] getCert()* de cette classe permet de récupérer le certificat signé par l'autorité de certification.
— **La classe RsaPrivateKeyMang** : Permet d'extraire le modulo de la clé RSA et l'exposant à partir du fichier généré précédemment par la classe CertificateManagement.
— **La classe EnvoiMultiple** : Permet d'envoyer les données de taille supérieure à 127 octets en plusieurs APDUs de taille égale ou inférieure à 127 octets.
— **La classe TextAreaOutputStream** : Permet la création d'une sortie standard vers une zone de texte, par exemple le résultat des commandes de l'outil GlobalPlatformPro est affiché dans un objet Écran.

2.2 L'application USTHB Moneo-Resto :

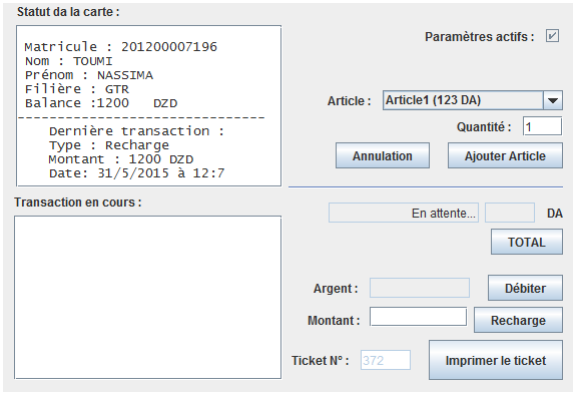


FIGURE 16 – Écran d'accueil de l'application USTHB Moneo-Resto après authentification

2.2.1 Les classes d'affichage :

— **La classe Start** : Point d'accès à l'application *USTHB Moneo-Resto*, elle propose à l'utilisateur le choix de configurer la caisse, ajouter une liste des produits disponibles dans le restaurant universitaire et de lancer la caisse.
— **La classe Modifier** : Permet de charger une liste de produits dans la caisse, créer une nouvelle liste de produits, ajouter un article à une liste qui existe déjà, supprimer un article d'une liste.
— **La classe Caisse** : Permet d'ouvrir la caisse et d'authentifier le client par sa carte à puce et son code PIN pour effectuer des opérations d'achat et de recharge.
— **Les classes ajouterArticle et supprimerArticle** : Sont appelées par la classe Modifier pour respectivement ajouter un article à une liste et en supprimer un.
— **La classe Config** : Permet de gérer les paramètres de la caisse tels que le nom de la caisse et la liste des articles à charger par défaut.

2.2.2 Les classes de gestion :

— **La classe JComboBoxListe** : Permet d'initialiser un objet de type JComboBox (une liste déroulante) avec les éléments d'un fichier *.liste* .

— **La classe ListeReader** : Hérite de l'objet ArrayList<String>, elle permet de lire une liste (fichier *.liste*).
— **La classe CardManagement** : A le même rôle que la classe CardManagement de l'application Admin Tool avec quelques fonctionnalités ajoutées comme les fonctions de débit, recharge et d'authentification ainsi que la récupération des informations du propriétaire de la carte. Les méthodes et attributs principaux de cette classe sont détaillés dans l'annexe B à la page 36.
— **La classe Cmd** : Permet l'exécution des lignes de commandes dans l'application d'une façon transparente, par exemple les commandes openSSL pour la vérification de certificat.
— **La classe EnvoiMiltiple** : La même classe que celle de l'application Admin Tool.
— **La classe GetDate** : Permet de créer un objet qui contient la date actuelle, il est utilisé pour enregistrer l'historique de la carte.

3.Applet Java Card :

Nous choisissons de donner à cette applet l'AID :
`0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01`
Dans le package : `0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00`
Afin de réaliser les opérations décrites précédemment, l'applet Java Card doit prendre en charge les APDUs des deux classes de commandes suivantes :

Classe ADMIN :

Est désignée par la valeur `0xB0` dans le champ Classe des APDUs, elle regroupe les commandes concernant l'initialisation des informations et ressources cryptographiques, l'authentification, le renouvellement de certificats et la réinitialisation du code PIN. Ces commandes appellent des méthodes qui effectuent les opérations demandées :

Commande	INS	P1	P2	Données	Méthode appelée
SET_INFO	0x70	0x00	0x00	Infos cryptées	init(buffer)
SET_L	0x71	0x00	0x00	Len/127, Len%127	setLengthCert(buffer)
SET_CERT	0x72	i	0x00	Bloc i du certificat	setCert(buffer)
SET_MOD	0x73	i	0x00	Bloc i du modulo	setPrivParam(buffer, modulo)
SET_EXP	0x74	i	0x00	Bloc i de l'exposant	setPrivParam(buffer, exposant)
FIN_INIT	0x75	0x00	0x00	null	finInit()
GET_INF_SIGN	0x76	0x00	0x00	null	getInfosSign(apdu)
GET_CERT_L	0x77	0x00	0x00	null	getCertLength(apdu)
GET_CERT	0x78	i	0x00	null	getCert(apdu)
GET_AL_M_T	0x7A	i	0x00	null	getAlM(apdu)
REP_DEC_M_T	0x7B	0x00	0x00	Message décrypté	repAlM(buffer)
GET_SIGN	0x7D	i	0x00	null	getSign(apdu)
GET_AES_KEY	0x7E	i	0x00	null	getAESKey(apdu)
ADMIN	0x60	0x00	0x00	Nouveau PIN crypté	changePIN(buffer)

TABLE 1 – APDUs de la classe ADMIN

Méthode	Description
init(byte[] buffer)	Prend en paramètre le buffer (tampon) de l'APDU, en extrait les informations relatives au propriétaire, les décrypte et initialise les tableaux qui les contiennent (nom, prénom, matricule, filière). Cette méthode initialise également l'objet OwnerPIN avec le code PIN du propriétaire.
setLengthCert(byte[] buffer)	Extrait les deux octets A et B permettant de calculer la longueur du certificat tels que : longueur=A*127+B.
setCert(byte[] buffer)	Permet l'initialisation du certificat (Réception par blocs de 127 octets avec la position du bloc spécifiée dans P1)
setPrivParam(byte[] buffer,byte[] param)	Peut prendre en second paramètre le modulo ou l'exposant de la clé privée RSA de la carte et permet de l'initialiser (Réception par blocs cryptés de 64 octets avec la position spécifiée par P1).
finInit()	Construit la clé privée à partir du modulo et de l'exposant précédemment initialisés et génère une signature avec celle-ci à partir des informations du propriétaire.
getInfosSign(APDU apdu)	Envoie les informations qui ont été utilisées lors de la génération de la signature après les avoir crypté avec la clé de session AES.
getCertLength(APDU apdu)	Envoie les deux octets A et B servant à calculer la longueur du certificat.
getCert(APDU apdu)	Envoie le certificat en plusieurs blocs de 127 octets, le numéro du bloc à envoyer est spécifié dans le P1 de l'APDU de commande.
getAlM(APDU apdu)	Génère un message aléatoire, le crypte avec la clé publique du terminal et l'envoie.
repAlM(byte[] buffer)	Compare le message déchiffré avec le message initial afin d'authentifier le lecteur.
getSign(APDU apdu)	Envoie la signature en plusieurs blocs.
getAESKey(APDU apdu)	Génère une clé de session AES de 128 bits, la crypte avec la clé publique du terminal et l'envoie.
changePIN(byte[] buffer)	Réinitialise l'objet OwnerPIN avec une nouvelle valeur.
encrypt(byte[] data)	Retourne la valeur cryptée avec la clé de session AES.
decrypt(byte[] data, byte[] output)	Décrypte le tableau *data* avec la clé de session AES et place le résultat dans le tableau *output*.

TABLE 2 – Méthodes appelées par les APDUs de la classe ADMIN

Note : Toutes les réponses de l'applet Java Card sont décrites dans l'annexe C à la page 39.

Classe RESTO :

Est désignée par la valeur `0x80` dans le champ Classe des APDUs, elle regroupe les commandes concernant les transactions, récupération des informations du propriétaire et la vérification du PIN. Ces commandes appellent des méthodes qui effectuent les opérations demandées :

Commande	INS	P1	P2	Données	Méthode appelée
CONSULT_BAL	0x50	0x00	0x00	null	getBalance(apdu)
CONSULT_MAT	0x50	0x01	0x00	null	getInfo(apdu,encrypt(matricule))
CONSULT_NOM	0x50	0x02	0x00	null	getInfo(apdu,encrypt(nom))
CONSULT_PRENOM	0x50	0x03	0x00	null	getInfo(apdu,encrypt(prenom))
CONSULT_FILIERE	0x50	0x04	0x00	null	getInfo(apdu,encrypt(filiere))
DEBIT	0x40	0x00	0x00	Montant crypté	debit(buffer)
RECHARGE	0x30	0x00	0x00	Montant crypté	recharge(buffer)
VERIFY	0x20	0x00	0x00	Code PIN	verify(buffer)
SET_HIST	0x10	0x00	0x00	Historique crypté	setHist(buffer)
GET_HIST	0x12	0x00	0x00	null	getHist(apdu)

TABLE 3 – APDUs de la classe RESTO

Méthode	Description
getBalance(APDU apdu)	Permet de retourner le montant crypté de la balance.
getInfo(APDU apdu,byte[] data)	Permet d'envoyer le tableau passée en second paramètre.
debit(byte[] buffer)	Permet de débiter le montant spécifié de la balance après décryptage de celui-ci, retourne des exceptions lorsque le montant dépasse la balance ou lorsqu'il est négatif.
recharge(byte[] buffer)	Permet de recharger la balance du montant spécifié après son décryptage, retourne des exceptions lorsque le montant est négatif.
verify(byte[] buffer)	Effectue la vérification du code PIN introduit par l'utilisateur.
setHist(byte[] buffer)	Enregistre le type, montant et la date de la dernière transaction effectuée après décryptage de ceux-ci.
getHist(APDU apdu)	Crypte les informations de la dernière transaction et les envoie.
encrypt(byte[] data)	Retourne la valeur cryptée avec la clé de session AES.
decrypt(byte[] data, byte[] output)	Décrypte le tableau *data* avec la clé de session AES et place le résultat dans le tableau *output*.

TABLE 4 – Méthodes appelées par les APDUs de la classe RESTO

III. Application administrative

Permet aux étudiants d'accéder à leurs documents comme les relevés de notes, et l'emploi du temps à l'aide d'une carte à puce personnalisée avec les informations spécifiques à chaque étudiant.

1. Fonctionnalités :

1.1 Administration

L'administration se fait de la même manière que pour l'application commerciale.

1.2 Consultation des documents

L'étudiant doit d'abord introduire son code PIN pour être authentifié :

— S'il introduit un PIN invalide trois fois, la carte se bloque et le PIN doit être réinitialisé par un terminal d'administration.
— Si le PIN est valide, les informations du propriétaire s'affichent.

Il pourra ensuite télécharger ses différents documents administratifs tels que ses relevés de notes.

2. Stockage des documents :

Pour que les étudiants puissent accéder à leurs documents et les télécharger, ceux-ci doivent être stockés dans une base de donnés se trouvant sur le serveur et accessible via celui-ci.

2.1 Microsoft SQL Server

Microsoft SQL Server est un système de gestion de bases de données relationnelles *SGBDR* développé et commercialisé par la société Microsoft. Pour les requêtes, SQL Server utilise T-SQL (Transact-SQL), une implémentation de SQL qui prend en charge les procédures stockées et les déclencheurs.
SQL Server est un SGBD relationnel, ce qui rend possible la définition des relations entre les tables de façon à garantir fortement l'intégrité des données qui y sont stockées. Ces relations peuvent être utilisées pour modifier ou supprimer en chaîne des enregistrements liés.

Pour les besoins de notre application, nous avons installé une base de données SQL Server 2012 Express sur notre serveur Windows 2012 afin d'héberger les différents documents des étudiants (Relevés de notes, emplois du temps...), sa gestion se fait à l'aide de l'interface graphique d'administration *SQL Server Management Studio* qui est accessible avec l'authentification Windows (compte administrateur) ou avec l'authentification interne (compte sysadmin).[19]

2.2 La propriété FILESTREAM de SQL Server

FILESTREAM permet aux applications SQL Server de stocker des données non structurées, telles que des documents et des images, dans le système de fichiers. Les applications peuvent tirer parti des APIs de diffusion et des performances du système de fichiers, et en même temps maintenir la cohérence transactionnelle entre les données non structurées et les données structurées correspondantes.
FILESTREAM intègre le Moteur de base de données SQL Server avec un système de fichiers NTFS en stockant les données d'objet *blob varbinary(max)* en tant que fichiers dans le système de fichiers. Les instructions Transact-SQL peuvent insérer, mettre à jour, interroger, rechercher et sauvegarder des données FILESTREAM. Les interfaces de système de fichiers Win32 fournissent l'accès de diffusion en continu aux données.

2.3 Création de la base de données

Dans un premier temps nous allons tester notre application sur l'ensemble des étudiants du département d'informatique de l'USTHB, pour ce faire nous avons créé une base de données *UsthbBDD* qui contient une table *DepINF*, cette table contient la liste des étudiants du département d'informatique ainsi que les relevés de notes annuels et l'emploi du temps de l'année en cours.

3. Applications terminal :

Comme dans l'application précédente (à usage commercial) nous allons créer deux sous applications différentes , l'une destinée aux administrateurs de la scolarité qui permet de personnaliser les cartes (Admin Tool), et l'autre (My e-doc) destinée aux étudiants pour pouvoir accéder à leur compte et télécharger

les différents documents, cette application va interagir avec une base de données SQL Server installée sur notre serveur et qui contient les documents de tous les étudiants.

3.1 L'application *Admin Tool* :

Cette application est basée essentiellement sur notre première application qui porte le même nom avec des petites différences dans le formulaire (la classe Formulaire) de personnalisation des cartes à puce.

3.2 L'application *My e-doc* :

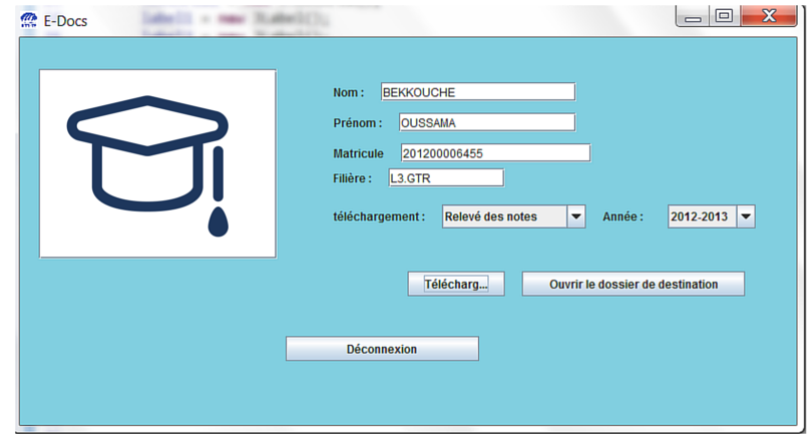


FIGURE 17 – Écran d'accueil de l'étudiant après authentification

3.2.1 Communication avec SQL Server : Pour qu'une application Java soit capable d'interagir avec un SGBD comme SQL SERVER, il est nécessaire d'utiliser un API spécial au SGBD en question, c'est ce qu'on appelle un JDBC (Java DataBase Connectivity). Le JDBC est en quelque sort un pilote pour les bases de données qui permet à une application Java de connecter à une base de données et d'exécuter des requêtes SQL. Microsoft offre un JDBC gratuit pour ses bases de données SQL SERVER dont la version la plus récente est JDBC Drivers 4.1.
Exemple de connexion sécurisée à notre base de données UsthbBDD avec le JDBC :

```
String connectionUrl = "jdbc:sqlserver://WIN−N2J430TRRNA;"
                                        + "databaseName=UsthbBDD;user=sa;
                                            password=PFE2012nchlh;"
                    + "encrypt=true;trustServerCertificate=true;";
Connection con = DriverManager.getConnection(connectionUrl);
```

Note :

- La propriété *encrypt* dans l'URL de connexion égale à "true" pour indique à SQL Server l'utilisations de SSL(Secure Sockets Layer) pour chiffrer la communication.
- la propriété trustServerCertificate égale à "true" indique que le JDBC fait confiance au serveur et n'exige pas de valider son certificat.

3.2.2 Les classes d'affichage :

— **La classe Start** : Point d'entrée de l'application (My e-doc), elle affiche un message d'accueil à l'étudiant et lui demande de s'authentifier avec sa carte à puce et son code PIN afin de se connecter à son compte.

— **La classe Espace** : Permet de lire les informations de l'étudiant à partir de la carte à puce et de les afficher. Elle propose également le téléchargement des documents de cet étudiant.

3.2.3 Les classes de gestion :

— **La classe CardManagement** : cette classe offre les mêmes fonctionnalités de la CardManagement de l'application USTHB Moneo-Resto sauf les fonctions de crédit et de débit.

— **La classe JComboBoxListe** : Permet d'initialiser un objet de type JComboBox (une liste déroulante) avec les éléments d'un fichier *.liste*.

— **La classe ListeReader** : Hérite de l'objet ArrayList<String>, elle permet de lire une liste (fichier *.liste*).

— **La classe Cmd** : permet l'exécution des lignes de commandes dans l'application d'une façon transparente, par exemple les commandes openSSL pour la vérification du certificat.

— **La classe EnvoiMultiple** : Permet d'envoyer les données de taille supérieure à 127 octets en plusieurs APDUs de taille égale ou inférieure à 127 octets.

— **La classe SQLManagement** : Permet de se connecter à la base de données, d'envoyer les requêtes et de récupérer les résultats.

4. Applet Java Card :

Nous choisissons de donner à cette applet l'AID :
`0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x01 0x01`
Dans le package : `0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x01`
Afin de réaliser les opérations décrites précédemment, l'applet Java Card doit prendre en charge les APDUs des deux classes de commandes suivantes :

Classe ADMIN :

Est désignée par la valeur `0xB0` dans le champ Classe des APDUs. Les APDUs et méthodes sont similaires à ceux de la Classe ADMIN de l'application commerciale.

Classe ID :

Est désignée par la valeur `0x80` dans le champ Classe des APDUs, elle regroupe les commandes concernant la récupération des informations du propriétaire et la vérification du PIN. Ces commandes appellent des méthodes qui effectuent les opérations demandées :

Commande	INS	P1	P2	Données	Méthode appelée
CONSULTATION_MAT	0x50	0x01	0x00	null	getInfo(apdu,encrypt(matricule))
CONSULTATION_NOM	0x50	0x02	0x00	null	getInfo(apdu,encrypt(nom))
CONSULTATION_PRENOM	0x50	0x03	0x00	null	getInfo(apdu,encrypt(prenom))
CONSULTATION_FILIERE	0x50	0x04	0x00	null	getInfo(apdu,encrypt(filiere))
VERIFY	0x20	0x00	0x00	Code PIN	verify(buffer)

TABLE 5 – APDUs de la classe RESTO

Méthode	Description
getInfo(APDU apdu,byte[] data)	Permet d'envoyer le tableau passée en second paramètre.
verify(byte[] buffer)	Effectue la vérification du code PIN introduit par l'utilisateur.
encrypt(byte[] data)	Retourne la valeur cryptée avec la clé de session AES.
decrypt(byte[] data, byte[] output)	Décrypte le tableau *data* avec la clé de session AES et place le résultat dans le tableau *output*.

TABLE 6 – Méthodes appelées par les APDUs de la classe ID

Note :
Toutes les réponses de l'applet Java Card sont décrites dans l'annexe C à la page 39.

IV. Perspectives

Usage commercial :

— Possibilité de débiter le montant des achats directement du compte CCP de l'étudiant.
— Tester l'application dans une Intranet réelle.

Usage administratif :

— Utilisation de la base de données de l'UTSHB afin de généraliser l'application de *My e-doc* aux autres départements et facultés de l'USTHB.
— Possibilité d'effectuer les choix de spécialités pour l'orientation via l'application.
— Ajout de comptes enseignants et administration pour la gestion des emplois du temps.

V. Conclusion

Dans ce chapitre, nous avons détaillé les fonctionnalités ainsi que les principales méthodes employées dans les applications Java Card de la carte et les applications terminal pour la mise en œuvre de la carte à puce pour les deux usages suivants :

— **Usage administratif** : Consultation et téléchargement des documents administratifs des étudiants (Relevés de notes, emplois du temps).
— **Usage commercial** : Porte-monnaie électronique pour restaurant universitaire.

Conclusion générale

Ce projet est axé sur la mise en œuvre de cartes à puce dans des systèmes électronique, dans le cadre de sa réalisation, nous avons développé deux solutions de mise en œuvre de carte à puce :

— Une solution à usage commercial qui consiste en un porte-monnaie électronique *USTHB Moneo-Resto* pour restaurant universitaire.

— Une solution à usage administratif *My e-doc* qui permet aux étudiants du département d'Informatique de consulter et de télécharger leurs documents administratifs tels que leurs relevés de notes et leurs emplois du temps.

Afin de mener à bien ce projet, nous-nous sommes familiarisés avec les cartes à puces, la technologie Java Card, ainsi que Windows Server et l'outil Open SSL. Nous avons également implémenté des mécanismes de sécurité employant les certificats, le cryptage, les signatures électroniques et la vérification du code PIN de l'utilisateur. Ces mécanismes nous ont permis d'assurer l'authentification, la confidentialité et l'intégrité des données.

Les solutions développées essaient de répondre au mieux aux objectifs du projet mais peuvent cependant être améliorées afin de proposer davantage de fonctionnalités et pouvoir être implémentées dans des systèmes plus étendus.

Annexe A

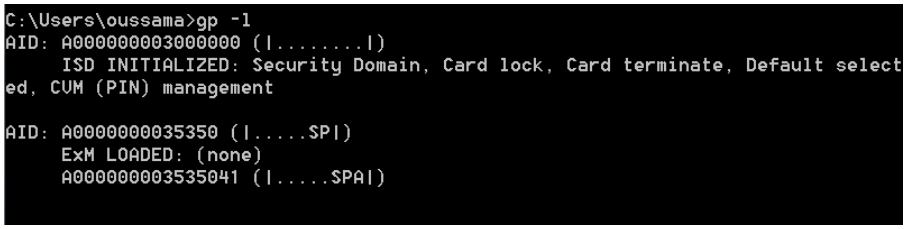
Commandes et résultats

1. Commandes de Global Platform Pro

— Lister les packages installés dans la carte à puce :

```
gp -l
```

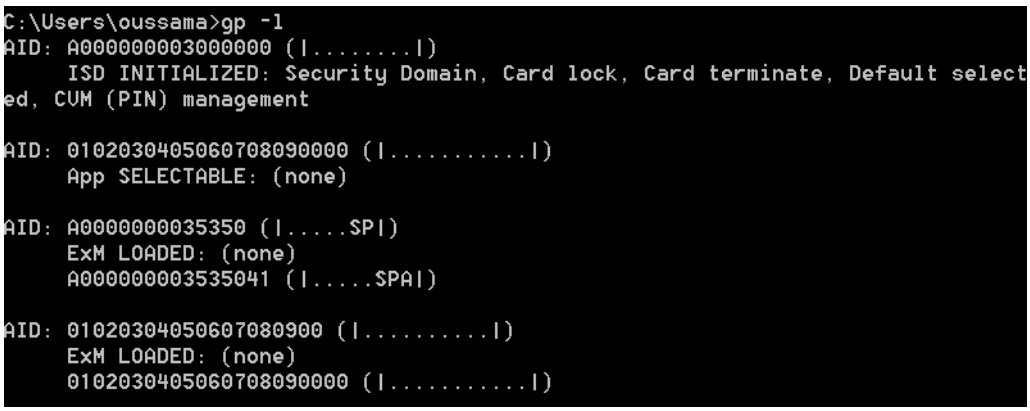
La commande donne comme résultat pour une carte sans applet installée :



— Installer un package en donnant le chemin de son fichier *.cap* :

```
gp -install C:\Users\oussama\workspace\RESTO\pack-resto.cap
```

Liste des packages après exécution de la commande :



— Supprimer un package avec tous les applets qu'il contient en mentionnant son AID :

```
gp -delete 01020304050607080900 -deletedeps
```

2. Commandes de génération de clés et demande de certificat

— Génération d'une paire de clés RSA 2018 bits avec Open SSL :

```
openssl req -newkey rsa:2048 -keyout myKey.pem -nodes -out
  myReq.req -subj /CN=201200007196/O=USTHB/C=DZ/L=Alger
```

Le fichier *myKey.pem* contient la paire de clés RSA, nous pouvons afficher son contenu avec :

```
openssl rsa -in myKey.pem -text
```

Parmi les informations affichées :

```
Private-Key: (2048 bit)
modulus:
00:da:fc:7a:c7:b3:1f:fb:ca:91:82:e5:10:d4:f3:
ae:45:d5:d8:56:66:23:d6:81:9d:3d:a4:a9:76:b5:
...
08:c3:04:a8:b6:38:9c:96:f2:3e:80:31:85:46:2d:
9a:7d
publicExponent:
65537 (0x10001)
privateExponent:
00:d3:7e:ab:1a:39:de:b6:f0:b2:4e:67:eb:34:b0:
50:6c:f4:16:67:83:68:33:60:a5:3b:86:cf:80:6d:
...
32:36:81:ee:53:85:dd:79:8f:b1:85:94:81:d7:97:
88:45
```

Ainsi que la clé privée en format *PKCS* encodée en base64 :

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA2vx6x7Mf+8qRguUQ1POuRdXYVmYj1oGdPaSpdrWHrzACXh6C
TYQUaxOKYEZDZl64Gl8X7GyrjWOs4bsFXEz7g787n/QOQ2Nxv8Sv/TAY5sjSEnlO
...
koTxWXuoeB/SsmkJypjNnF7+WK3upMW0mD1spgSfjizuwhzYwqmM
-----END RSA PRIVATE KEY-----
```

— Envoi de la requête à l'autorité de certification :

```
certreq -submit -binary -attrib
"CertificateTemplate:SmartCardPFE " -config
USTHB-WIN-N2J430TRRNA-CA myReq.req certificat.cer
```

Avec *SmartCardPFE* le modèle de certificats configuré lors de la création de la CA, *USTHB-WIN-N2J430TRRNA-CA* le nom de la CA, *myReq.req* la requête créée précédemment et en dernier le nom du fichier dans lequel nous voulons stocker le certificat reçu. Nous pouvons consulter les informations contenues dans ce certificat après avoir exécuté ces deux commandes :

```
openssl x509 -in certificat.cer -inform DER -out certificat.pem
openssl x509 -in certificat.pem -text
```

Nous auront pour résultat entre autres informations :

```
Version: 3 (0x2)
Serial Number:
3c:00:00:00:23:9c:6b:47:45:ad:be:67:42:00:00:00:00:00:23
Signature Algorithm: sha1WithRSAEncryption
Issuer: DC=INFO, DC=USTHB, CN=USTHB-WIN-N2J430TRRNA-CA
Validity
Not Before: May 11 09:32:25 2015 GMT
Not After : May 10 09:32:25 2016 GMT
Subject: C=DZ, L=ALGER, O=USTHB, CN=201200007196
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
Modulus:
          00:da:fc:7a:c7:b3:1f:fb:ca:91:82:e5:10:d4:f3:
          ae:45:d5:d8:56:66:23:d6:81:9d:3d:a4:a9:76:b5:
                                ...
          08:c3:04:a8:b6:38:9c:96:f2:3e:80:31:85:46:2d:
                              9a:7d
Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
     45:fd:1a:5f:d3:52:7f:02:e5:ca:60:7a:f7:31:ff:94:36:7c:
     bc:34:95:db:55:0f:bc:e9:4f:19:de:45:a6:ba:9a:c2:59:12:
                                ...
     65:db:c3:b0:68:4e:02:98:8d:ae:e6:d6:de:83:20:3a:7a:b7:
                           f8:85:ef:59
```

La signature a été générée avec la clé privée de l'autorité de certification et permet d'authentifier le certificat par la suite.

Le certificat en format *PKCS* encodé en base64 :

```
                    ---BEGIN CERTIFICATE---
MIIGCTCCBPGgAwIBAgITPAAAACOca0dFrb5nQgAAAAAAIzANBgkqhkiG9w0BAQUF
ADBQMRQwEgYKCZImiZPyLGQBGRYESU5GTzEVMBMGCgmSJomT8ixkARkWBVVTVEhC
                                ...
FxjvWynmLHQ8xtNLYDS8qF6zcQH5ct35+RDrEVK8Hunmp/4c9Z5LZdvDsGhOApiN
                        rubW3oMgOnq3+IXvWQ==
                    ---END CERTIFICATE---
```

Annexe B

Méthodes et attributs principaux de la classe CardManagement

1. Méthodes principales :

Méthodes communes aux classes CardManagement de toutes les applications	
Méthode	**Description**
protected static void evalSW(ResponseAPDU rep)	Évalue la réponse de la carte.
private static byte [] decryptAuth(byte []data)	Déchiffre le message aléatoire de la carte avec la clé publique RSA du terminal.
protected static boolean verifyCardConnection()	Teste la présence de la carte dans le lecteur.
protected static byte select()	Envoie un APDU de sélection à la carte .
protected static boolean verifyTerminal()	Demande un message aléatoire chiffré à la carte pour authentifier le terminal .
protected static void setAESKey()	Demande une clé de session AES à la carte.
protected static byte[] AESEncrypt	Chiffre un message avec la clé de session AES.
protected static byte[] AESDecrypt(byte[] data)	Déchiffre un message avec la clé de session AES.
protected static boolean verifyCard()	Vérifie la validité du certificat de la carte à puce et de la signature.
public CardManagement()	Constructeur de la classe, il permet également d'initialiser les clés du terminal et de se connecter au lecteur.

TABLE B.1 – Méthode communes aux classes CardManagement de toutes les applications

Méthodes spécifiques à la classe CardManagement des applications *Admin Tool*	
Méthode	**Description**
protected static boolean init(String PIN,String matricule,String nom,String prenom,String filiere,int type)	Si type = 0 : Elle envoie les informations du client à la carte avec le certificat et la clé privée. Si type = 1 : Elle réinitialise le certificat et la clé privée.
protected static void update(String PIN)	Permet de réinitialiser la valeur du PIN.

TABLE B.2 – Méthodes spécifiques à la classe CardManagement des applications *Admin Tool*

Méthodes spécifiques à la classe CardManagement de l'application *USTHB Moneo-Resto*	
Méthode	**Description**
protected static boolean authentification()	Se connecte à la socket de l'outil ReaderManager pour l'authentification du client par son code PIN à travers le lecteur.
private static byte [] consultation(byte P1,CardChannel canal)	Permet d'envoyer un APDU de consultation à la carte pour récupérer une informations selon la valeur de paramètre P1.
protected static String getInfo()	Fait appel à la méthode consultation() pour retourner une chaîne qui contient toutes les informations du client à la fois
protected static void recharge(byte [] montant)	Permet d'envoyer un APDU de recharge avec le montant crypté en paramètre.
protected static void debit(byte [] montantArray)	Permet d'envoyer un APDU de débit avec le montant crypté en paramètre.

TABLE B.3 – Méthodes spécifiques à la classe CardManagement de l'application *USTHB Moneo-Resto*

2. Attributs principaux :

Attributs communs aux classes CardManagement de toutes les applications	
Attribut	**Description**
final static byte ADMIN_CLA = (byte)0xB0	Le champ CLA pour les APDUs de commande.
private static PrivateKey priv	La clé privée RSA du terminal.
private static PublicKey pub	La clé publique RSA du terminal.
private static byte [] msgAlCrp	Tableau qui contient le message aléatoire de la carte chiffré avec RSA.
private static byte [] msgAlClaire	Tableau qui contient le message aléatoire de la carte après le déchiffrement RSA.
protected static Signature dsa	L'objet signature utilisé pour vérifier la signature de la carte.
protected static CardTerminal cad	L'objet qui représente le lecteur de cartes à puce.
static Card c	L'objet qui représente la carte à puce.
static byte[] AESData= new byte[256]	Contient les données utilisées pour la génération de la clé de session AES (cryptées avec la clé publique RSA du terminal).

TABLE B.4 – Attributs communs aux classes CardManagement de toutes les applications

Attributs spécifiques à la classe CardManagement de l'application *USTHB Moneo-Resto*	
Attribut	**Description**
final static byte CAISSE_CLA = (byte)0x80	Le champ CLA pour les APDUs de commande de transactions.
final static byte VERIFY = (byte) 0x20	Le champ INS pour l'APDU de commande de vérification du PIN.
final static byte RECHARGE = (byte)0x30	Le champ INS pour l'APDU de commande de recharge.
final static byte DEBIT =(byte)0x40	Le champ INS pour l'APDU de commande de débit.
final static byte CONSULTATION =(byte)0x50	Le champ INS pour les APDUs de commande de consultation des informations (balance, matricule...).
final static byte SETHIST=(byte)0x10	Le champ INS pour l'APDU de commande d'envoi de l'historique.
final static byte GETHIST=(byte)0x12	Le champ INS pour l'APDU de commande de la récupération de l'historique.

TABLE B.5 – Attributs spécifiques à la classe CardManagement de l'application *USTHB Moneo-Resto*

Annexe C

Réponses de l'applet Java Card

SW1 SW2	Signification
0x9000	Opération réussie.
0x6300	Authentification PIN échouée.
0x6301	Authentification PIN requise.
0x6302	Nombre d'essais PIN restants nul.
0x6402	Terminal non reconnu.
0x6700	Longueur incorrecte.
0x6800	Valeur de la variable pointée nulle (NullPointerException).
0x6801	L'indexe dépasse la longueur du tableau (ArrayIndexOutOfBoundsException).
0x6811	Exception cryptographique : valeur non autorisée.
0x6812	Exception cryptographique : clé non initialisée.
0x6813	Exception cryptographique : algorithme non supporté.
0x6814	Exception cryptographique : l'objet signature ou de cryptage n'a pas été correctement initialisé.
0x6815	Exception cryptographique : L'algorithme de cryptage ou de signature n'effectue pas de padding et la taille message d'entrée n'est pas multiple de la taille de blocs traités .
0x6820	Exception de sécurité.
0x6982	Statut de sécurité non satisfait.
0x6999	Échec lors de la sélection de l'applet.
0x6A83	Montant de transaction invalide.
0x6A84	Montant maximal de la balance dépassé, transaction non effectuée.
0x6A85	Balance négative, transaction non effectuée.
0x6A86	Paramètres P1-P2 incorrects.
0x6CXX	Longueur incorrecte, SW2 indique la longueur exacte.
0x6D00	Code de l'instruction invalide ou non supporté.
0x6E00	Classe non supportée.
0x6F00	Exception non définie.

TABLE C.1 – Principaux codes d'erreur retournés par la carte

Sommaire

Table des figures

Liste des tableaux

Bibliographie

[1] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 7816 : Smart Card Standard. Norme, 1987.

[2] Ouadjaout Lamine and Zergoug Salim. La carte d'identité électronique (eid). Master's thesis, USTHB, 2010.

[3] Asgari Reza and Ebrahimi Atani Reza. Classification of smart card operating systems. *Computer Engineering and Applications Journal*, pages 13–22, 2012.

[4] Ghernaouti Solange. *Sécurité informatique et réseaux, 4ème édition*. Dunod, 2013. ISBN : 978-2-10-059912-7.

[5] Effing Wolfgang and Rankl Wolfgang. *Smartcard Handbook 4th edition*. John Wiley & Sons, Ltd, 2010. ISBN : 978-0-470-74367-6.

Webographie

[6] Zhiqun Chen. How to write a java card applet : A developer's guide, Juillet 1999.
http://www.javaworld.com/article/2076450/client-side-java/how-to-write-a-java-card-applet--a-developer-s-guide.html.

[7] Zhiqun Chen and Rinaldo Di Giorgio. Understanding java card 2.0, 1998.
http://www.javaworld.com/article/2076617/embedded-java/understanding-java-card-2-0.html.

[8] Guo Heng. Smart cards and their operating systems.
http://www.scardsoft.com/documents/COS/heng_guo.pdf.

[9] Sauveron Damien. La carte à puce multi-applicative et sa sécurité.
http://damien.sauveron.fr/fileadmin/damiensauveron/slides/JourneCarte.pdf.

[10] Mahmoudi Ramzi. Programmation carte à puce – javacard, 2013.
http://www.research-ace.net/~mahmoudi/engineering/smart/JC03.pdf.

[11] CryptAGE. Le RSA.
http://www.cryptage.org/rsa.html.

[12] Installer une autorité de certification racine.
https://technet.microsoft.com/fr-fr/library/cc731183.aspx.

[13] How to install certificate authority on windows server 2012.
http://careexchange.in/how-to-install-certificate-authority-on-windows-server-2012.

[14] Writing a java card applet.
http://www.oracle.com/technetwork/java/javacard/intro-139322.html.

[15] Aid (java card api and subsets).
http://www.win.tue.nl/pinpasjc/docs/apis/jc222/javacard/framework/AID.html.

[16] Pc pinpad smartcard reader, pc/sc application note.
http://support.gemalto.com/fileadmin/user_upload/user_guide/Pinpad/PCPinpad_PC-SC_UserGuide.pdf.

[17] Globalplatformpro list / install / delete applets.
https://github.com/martinpaljak/GlobalPlatformPro.

[18] About the openSSL project.
https://www.openssl.org/about/.

[19] SQL Server 2008 - gestionnaire de base de données.
http://www.artduweb.com/windows/mssql.

ملخص

الاستعمال المتزايد لأنظمة الاتصال ألالكتروني ,خلق الحاجة الى تطوير حلول امنية من اجل اثبات هوية مستخدمي هذه الانظمة. من بين هذه الحلول ,استعمال البطاقات الذكية التى تسمح بتخزين المعلومات الخاصة بحاملها من اجل اثبات هويته في النظام . في هذا العمل قمنا باقتراح حلين لاستخدام البطاقات الذكية الاول اداري والثاني تجاري ,هذه الحلول تسمح ببرمجة البطاقات الذكية و تغيير محتواها و الاتصال بها بطريقة امنة بواسطة الأجهزة النهائية. انجاز هذا المشروع مر بعدة مراحل ,من بينها تطوير البرنامج المثبت في البطاقة باستعمال لغة JAVA CARD و التطبيقات المثبتة على الاجهزة النهائية و تقنيات اثبات الهوية مثل التشفير والتوقيع الالكتروني والشهادات الالكترونية.

Résumé

L'utilisation grandissante des systèmes de communications électroniques a créé le besoin de mettre en place des solutions de sécurité pour l'authentification des personnes. Une solution consiste en l'utilisation des cartes à puces qui permettent de stocker les informations liées au porteur et d'appliquer les mécanismes d'authentification dans un système.

Dans ce travail, nous proposons deux solutions de mise en œuvre de cartes à puce pour usage administratif et commercial, ces solutions permettent d'initialiser les cartes à puce, de gérer leur contenu et de communiquer de façon sécurisée avec le terminal.

La réalisation passe par le développement d'applets dans le langage Java Card ainsi que les applications terminales et la mise en place de mécanismes d'authentification tels que le chiffrement et l'usage de certificats.

Abstract

The growing use of electronic communication systems created the need to implement security solutions to authenticate persons. One of these solutions consists in using Smart Cards that can store user-related data and apply authentication mechanisms.

In this work, we propose two implementation solutions for Smart Cards in administrative and commercial usage, these solutions allow users to initialize Smart Cards, to manage their content and to communicate securely with the terminal.

The realization consists in developing Java Card applets, terminal application and setting authentication mechanisms as encryption and certificate usage.