Secrétariat général
de la défense
et de la sécurité nationale

Agence nationale de la sécurité
des systèmes d'information

Stage du 1 février au 31 juillet 2019

Nombre de pages du document : 37

Création d'un démon compatible GlobalPlatform pour un environnement Java Card indépendant

Vincent Giraud

Tuteur professionnel : Guillaume Bouffard
Tuteur académique : Maxime Pelcat

51 boulevard de La Tour-Maubourg 75700 PARIS 07 SP - Tél 01.71.75.84.65 - Fax 01.71.75.84.90

Résumé

Les plateformes embarquées sécurisées sont au cœur de la vie quotidienne de nombreuses personnes dans le monde, que ce soit sous forme de carte, d'objet connecté, ou d'enclave sécurisée dans un système sur puce.

Dès lors, il est naturel de s'inquiéter de leur sécurité et de tenter d'en améliorer la protection. C'est pour cette raison que l'ANSSI a lancé le projet dans lequel s'inscrit ce stage.

Ce projet consiste en la conception et réalisation d'un environnement Java Card complet et sécurisé. Ce stage quant à lui consiste principalement au développement d'un agent logiciel embarqué en charge de divers rôles sécuritaires. Ces rôles incluent notamment la gestion des contenus sur la carte, le contrôle des communications, l'encadrement des interactions entre applications, la gestion de permissions.

Ce document détaille la création d'un tel démon, et justifie les nombreux choix réalisés face aux contraintes d'un tel système en conditions réelles. Si la plupart du travail a eu lieu au niveau applicatif, il a tout de même été souvent influencé par des choix de conception situés plus proche du matériel.

Mots-clés

Java Card, GlobalPlatform, embarqué, sécurité, carte à puce, système

Table des matières

Introduction

Présentation générale de l'agence

L'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) a été créée par le décret n°2116 23 2009-834 du 7 juillet 2009 (Journal officiel du 8 juillet 2009), sous la forme d'un service à compétence nationale. Elle est rattachée au Secrétariat Général de la Défense et de la Sécurité Nationale (SGDSN), autorité chargée d'assister le Premier ministre dans l'exercice des ses responsabilités en matière de défense et de sécurité nationale.

L'ANSSI est l'autorié nationale en matière de sécurité et de défense des systèmes d'information. Elle a pour missions d'assurer la sécurité des systèmes d'information de l'État et de veiller à celle des Opérateurs d'Importance Vitale (OIV), de coordonner les actions de défense des systèmes d'information, de concevoir et déployer les réseaux sécurisés répondant aux besoins des plus hautes autorités de l'État et aux besoins interministériels, et de créer les conditions d'un environnement de confiance et de sécurité propice au développement de la société de l'information en France et en Europe.

L'ANSSI est organisée en cinq sous-directions :

— La Sous-Direction Opération (SDO) assure, au niveau opératif et tactique, la mise en œuvre de la fonction d'autorité de défense des systèmes numériques d'intérêt pour la nation. Elle constitue le centre opérationnel de la sécurité des systèmes d'information ;

— La Sous-Direction du Numérique (SDN) porte la mission de définir, concevoir, mettre en œuvre et soutenir les systèmes d'information en adéquation avec les besoins de sécurité requis pour ces systèmes. Elle intervient au profit du SGDSN dont elle assure le rôle de direction du numérique et des ministères dans le cadre des systèmes d'information interministériels sécurisés dont elle assure la maîtrise d'œuvre ;

— La Sous-Direction Stratégie (SDS) anime le processus de planification stratégique au sein de l'agence, développe et pilote la contribution de l'agence à l'élaboration et à la mise en œuvre des politiques publiques en faveur de la sécurité du numérique, communique vers l'ensemble des publics sur les enjeux de sécurité numérique et fait connaître les missions de l'ANSSI et

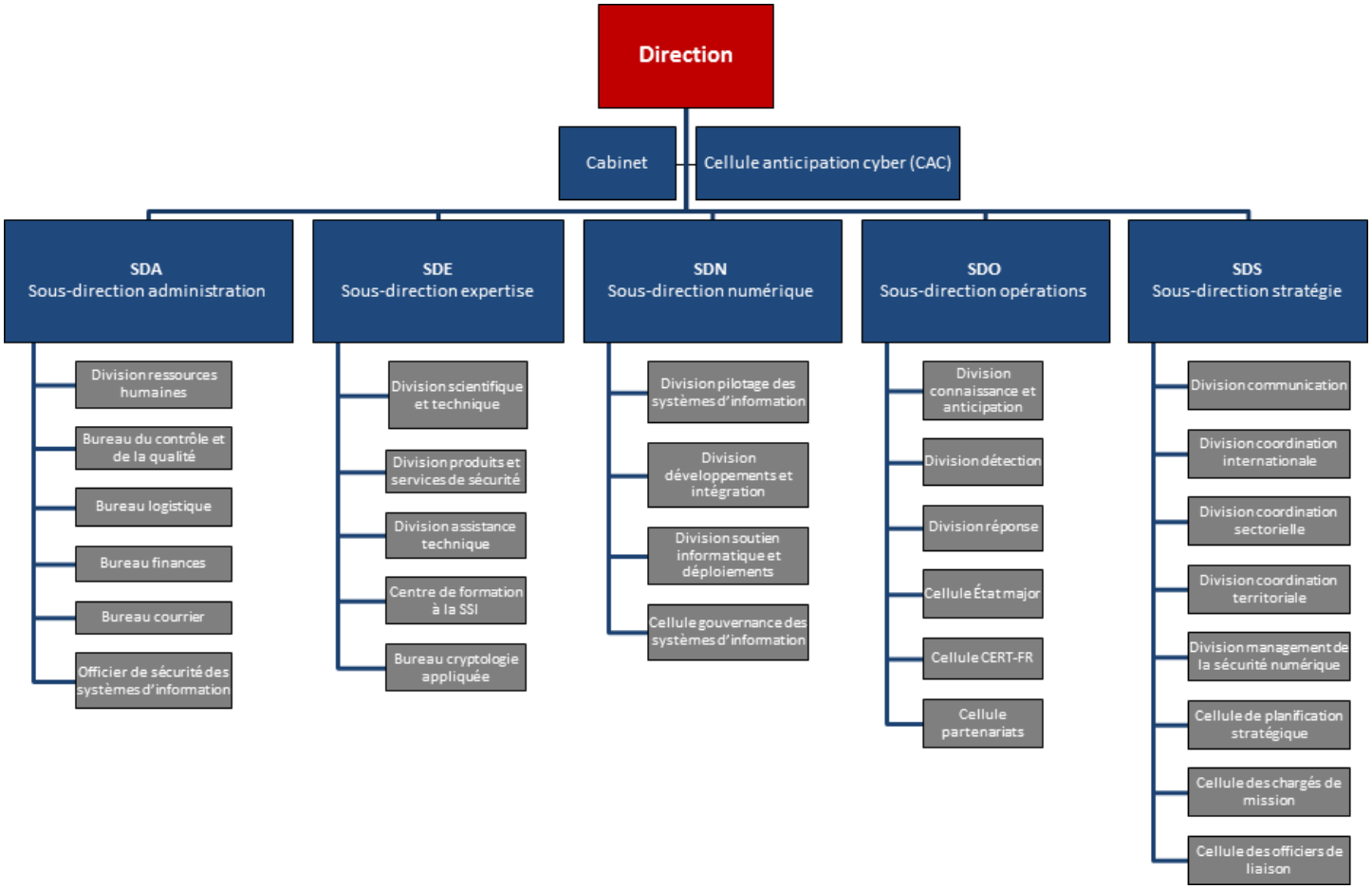

Figure 1 – Organigramme de l'ANSSI

ses activités et recommandations auprès de ses publics ;

— La Sous-Direction Administration (SDA) assure la politique des ressources humaines de l'ANSSI en cohérence avec celle du SGDSN, décline la politique immobilière du SGDSN pour les propres besoins de l'agence et prépare la planification financière de l'ANSSI ;

— La Sous-Direction Expertise (SDE) porte la mission globale d'expertise et d'assistance technique de l'agence. Elle apporte son soutien à l'ensemble des autres sous-directions de l'ANSSI, aux ministères, aux industriels et prestataires de sécurité et aux OIV.

Le laboratoire sécurité des composants

Le stage se déroule au sein du Laboratoire Sécurité des Composants (LSC), intégré dans la division scientifique et technique de la sous-direction Expertise avec d'autres laboratoires. Les activités principales du LSC regroupent des activités d'expertise et de recherche dans le domaine de l'analyse de la sécurité des composants afin d'assurer un haut niveau de sécurité sur des produits civils et gouvernementaux. Chaque expert du laboratoire est chargé des tâches suivantes :

- Assurer une veille scientifique de haut niveau dans le domaine des attaques physiques sur composants. En particulier via le suivi de travaux de recherche académique et la participation aux conférences majeures du domaine.
- Assurer l'élaboration et le maintien à jour d'un référentiel documentaire permettant de fixer les règles et recommandations que doivent suivre les composants implémentant des mécanismes cryptographiques.
- Participer à la mise en place puis au maintien de l'état de l'art au sein du laboratoire de plate-formes d'attaques par canaux auxiliaires et d'attaques par fautes.
- Assurer une activité de conseil et d'assistance auprès des acteurs majeurs du domaine que sont les ministères, les centres d'évaluation et les partenaires du secteur privé.
- Contribuer au développement de produits de sécurité industriels robustes.
- Assurer le suivi des dossiers d'évaluation des composants cryptographiques soumis aux divers processus d'évaluation (qualification, certification Critères-Communs, Certification de Sécurité de Premier Niveau (CSPN)…).
- Valider, les cas échéant, les vulnérabilités identifiées au cours des processus d'évaluation par une mise en œuvre pratique et en étudiant leurs conséquences.
- Assurer le suivi des compétences techniques des centres d'évaluation de produits civils.
- Assurer en collaboration avec la Direction Générale de l'Armement (DGA) une expertise de la sécurité des composants cryptographiques réalisés à des fins gouvernementales.

1 Contexte

1.1 Plateformes embarquées sécurisées

L'amélioration des techniques de production et les progrès de l'industrie ont permis, ces dernières décennies, de réduire drastiquement la consommation des produits électroniques ainsi que les coûts de fabrication. Ces évolutions ont alors permis la popularisation de nombreux périphériques dits intelligents, et le nombre de systèmes informatisés dans la vie d'une même personne a explosé. Le cabinet d'étude Gartner prévoit 20 milliards d'objets connectés en 2020[1]. Cependant, ces périphériques peuvent avoir pour vocation d'interagir avec des éléments personnels et fortement sensibles. Un produit destiné au marché de la santé ou de l'identification (notamment biométrique) va exploiter des données critiques. Or seulement 38 % des entreprises en charge d'un objet connecté prévoient des responsables de sécurité lors du processus de développement[2]. Il devient alors critique de proposer des outils et environnements propices à la création de logiciels sécurisés.

Le marché des plateformes embarquées est fortement développé et son omniprésence peut se faire discrète pour les personnes non-averties. On trouve évidemment de tels produits dans la téléphonie ; mais également dans l'électroménager, ou dans les secteurs du bancaire, du médical, du commercial et du loisir ; où l'on exploite souvent des microcontrôleurs, parfois sous la forme de carte à puce.

La popularisation des plateformes embarquées auprès du grand public a fortement bouleversé le modèle de risque autour de ces produits. La perte ou le vol d'un microcontrôleur contenant des données sensibles sont devenus des risques bien plus présents. Ces périphériques bénéficient d'une grande exposition auprès d'aspects intimes de la vie des citoyens : caméras, compteurs intelligents, serrures électroniques, électroménager intelligent…

Les cartes à puce, telles que les cartes bancaires ou les cartes Subscriber Identity Module (SIM), ne contiennent essentiellement qu'un microcontrôleur. Ce dernier bénéficie de caractéristiques physiques permettant son intégration à la carte, c'est-à-dire d'une petite taille, d'un poids faible, et d'une consommation électrique minimale. Ce périphérique communique avec un agent extérieur soit grâce aux huit contacts physiques présents sur la carte (souvent de couleur dorée) ; soit par

onde électromagnétique : on parle alors de communication sans contact, et une bobine conductrice est insérée au sein du plastique pour faire office d'antenne et recevoir l'énergie d'un lecteur.

Les composants employés dans ce domaine sont des composants sécurisés, ce qui impose certaines contraintes, entraînant un impact fort sur les performances de la puce. Les versions sécurisées des microcontrôleurs bénéficient notamment de protections matérielles contre les attaques physiques, dont les attaques par canaux auxiliaires, basées essentiellement sur la consommation électrique ou les mécanismes de prédiction d'instructions.

Pendant longtemps, le développement de cartes à puce était un domaine très hétérogène, et chaque constructeur avait sa technologie. Petit à petit, elles ont vu leurs caractéristiques être communes, avec le respect progressif de l'ensemble de normes ISO/IEC-7816[3]. Cependant, si cette normalisation ne considère que les aspects physiques des cartes à puce ainsi que leurs communications externes, les applications développées étaient toujours intimement liées à chaque carte physique sur laquelle elle s'exécutait.

1.2 Technologie Java Card

C'est pour harmoniser les environnements d'exécution que s'est développé le standard Java Card, qui a pour vocation de permettre de faire fonctionner des applications écrites en Java sur des cartes à puce. Ainsi résident sous l'application une machine virtuelle[4] et un environnement d'exécution[5], comme le montre la figure 2. L'application par-dessus peut alors fonctionner sur tous les matériels supportés sans avoir à être systématiquement portée.

De plus, le choix du langage Java permet d'augmenter la sécurité des applications, étant donné que ce langage ne permet pas de manipulation directe de pointeurs (il est dit *memory-safe*), ce qui élimine une classe de vulnérabilités.

Étant donnée les ressources fortement contraintes de ce genre d'équipements, toutes les fonctionnalités de Java ne sont pas disponibles, et un sous-ensemble raisonnable en a été choisi. De plus, de par le caractère minimal du système d'exploitation et les fortes exigences de sécurité en termes de confidentialité des données, une ségrégation stricte de la mémoire a été décidée. Parmi les aspects de Java perdus dans Java Card, on trouve les collections. Cela permet de réduire considé-

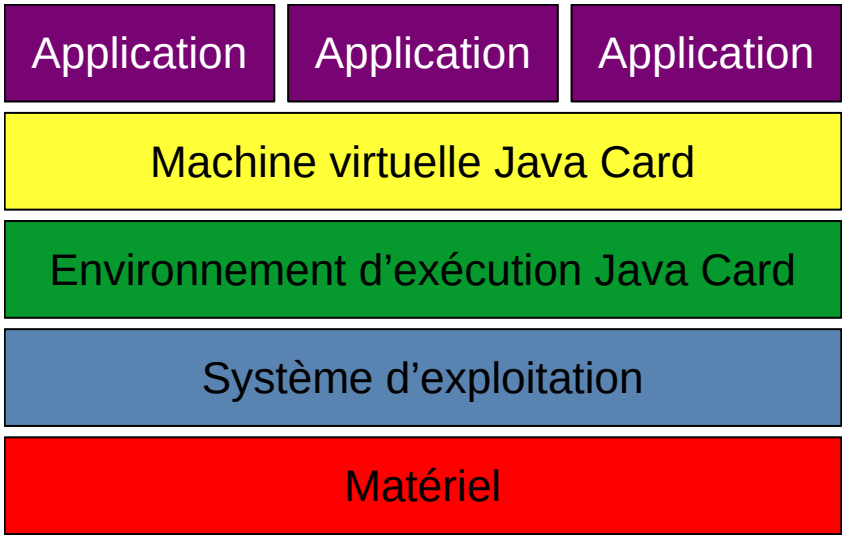

FIGURE 2 – Schéma d'abstraction représentant l'organisation classique d'un environnement prévu pour Java Card

rablement la bibliothèque standard. Aussi, les systèmes Java Card étant dans leur grande majorité embarqués, il est rare d'utiliser de grandes structures de données. Si une organisation précise d'éléments est nécessaire, le développeur devra programmer lui-même la gestion d'attributs. Aussi, le support du type «int» n'est pas assuré. En effet, lors de la conception de Java Card au milieu des années 90, peu de microcontrôleurs étaient équipés de processeurs 32 bits. Par compatibilité, il est conseillé d'éviter ce type même si, aujourd'hui, les processeurs fonctionnant avec des registres plus grands que 16 bits sont la norme.

Les contraintes reposant sur la machine virtuelle Java Card sont de deux types : celles qui sont basées sur la spécification que la machine doit respecter pour interpréter correctement un *applet* Java Card, et celles qui sont basées sur le matériel sur lequel la machine virtuelle doit tourner. Ce sont, pour reprendre un vocabulaire utilisé principalement en réseaux, les contraintes imposées par les couches supérieures et inférieures, qui doivent être respectées pour permettre l'interopérabilité.

Si Java Card est né au milieu des années 90, cela reste une technologie éminem-

ment utilisée aujourd'hui. De nombreuses évaluations de sécurité sont réalisées aujourd'hui dans ce domaine dans le cadre des certifications Critères Communs (CC). Java Card est aussi souvent présent dans les périphériques traitant des données biométriques, par exemple à des fins d'authentification.

Enfin, il convient de noter que malgré son nom, un environnement d'exécution Java Card peut être mis en place sur d'autres objets qu'une carte à puce. En effet, comme énoncé précédemment, une telle carte ne contient en réalité qu'un microcontrôleur. On peut alors envisager un système Java Card sur n'importe quel objet connecté, même si d'autres environnements peuvent être plus adaptés. Aujourd'hui, Apple et Samsung implémentent cet environnement dans une enclave sécurisée au sein de leurs systèmes[6][7] pour développer des solutions de paiement. On retrouve également Java Card dans le domaine des puces destinées à être implantées dans le corps humain[8].

1.3 Implémentation propre à l'ANSSI

Les cartes à puce opèrent dans des domaines à la fois très lucratifs et sensibles. On peut alors constater que tous les acteurs et manufacturiers sont des groupes privés, et qu'à l'heure actuelle, tous les environnements Java Card complets sont privatifs et sous licence propriétaire. Il n'existe ainsi pas d'implémentation ouverte où un citoyen lambda pourrait librement consulter les sources du système, des couches proches du matériel aux interfaces de programmation applicatives haut niveau.

Devant ce constat, il a été décidé au LSC d'initier la création d'une plateforme Java Card. Grâce à ses origines et ses financements à caractère public, elle pourrait alors profiter d'une ouverture et de l'accès libre que les autres implémentations ne peuvent pas se permettre. Plusieurs bénéfices peuvent être tirés d'une telle version :

- Pouvoir accéder au sein d'un environnement Java Card offre un fort potentiel de recherche académique. En pouvant appliquer ses propres innovations dans le système, un chercheur peut ainsi expérimenter bien plus facilement.
- Une version ouverte se révèle fort pratique dans le cadre éducatif. Elle permet aux étudiants d'accéder à un système complet gratuitement et plus facilement. Là encore, elle simplifie leur expérimentation.
- Comme pour les autres projets ouverts de l'ANSSI, les sources peuvent

constituer une première référence auprès de la communauté de développeurs dans ce domaine. D'autres projets peuvent ainsi implémenter des mesures de sécurité plus drastiques grâce au partage de connaissances.

Dans son implémentation propre, le LSC a décidé d'adopter un changement radical par rapport à l'organisation classique d'un tel environnement. Comme le représente la figure 2, un système conventionnel contient un environnement d'exécution Java Card associé à sa machine virtuelle. Cette dernière assure la stricte séparation des applications entre elles.

Dans l'implémentation de l'ANSSI, il a été décidé de faire descendre l'isolation jusqu'au système d'exploitation, comme l'illustre la figure 3. Cela implique la duplication des couches Java Card : il y a désormais une instance de machine virtuelle et d'environnement Java Card par application. Toute interaction entre applications doit désormais descendre et atteindre le système d'exploitation avant de pouvoir transiter.

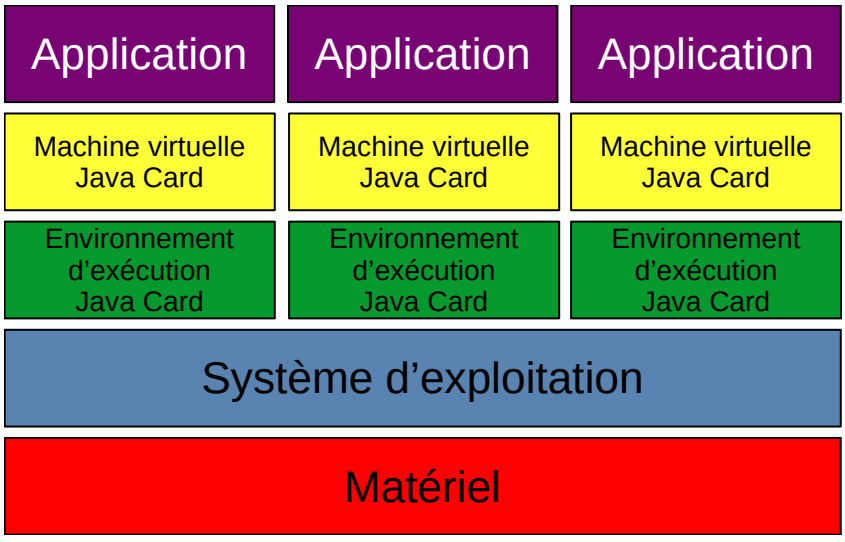

FIGURE 3 – Schéma d'abstraction représentant l'organisation de l'environnement propre à l'ANSSI

Cette nouvelle organisation offre un avantage puisqu'elle permet de déléguer une partie de la ségrégation au matériel. En temps normal, si deux applications veulent partager un segment de mémoire tampon, les accès sont gérés et sécurisés par la machine virtuelle. Cependant ici, on peut profiter de l'unité de protection

mémoire, ou Memory Protection Unit (MPU). Ce périphérique matériel, souvent inclus dans les systèmes sur puce, permet de sectoriser l'ensemble de l'espace mémoire, et de définir des règles de privilèges sur chacune des régions. Ainsi on décharge toutes les vérifications d'accès mémoire au matériel. Profiter de la protection matérielle n'est pas possible lorsqu'une seule machine virtuelle est présente sur la plateforme car le système d'exploitation ne peut différentier des applications Java Card distinctes ; de son point de vue, il ne voit qu'un seul logiciel : l'environnement Java Card. Cette innovation a donné lieu à une publication à l'édition 2018 du Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)[9]. Ce changement de paradigme, ainsi que ses tenants et aboutissants y sont expliqués en détail.

Un tel projet représente une charge de travail conséquente. Guillaume Bouffard, à la tête du projet, s'occupe de l'environnement d'exécution et de la machine virtuelle, qui sont implémentés avec le langage C++. Le système d'exploitation a, lui, été délégué à Léo Gaspard durant un stage en 2016 : pour des raisons de sécurité et pour éviter les erreurs humaines[10], il a choisi de le programmer à l'aide du langage Rust. Utiliser le langage C a néanmoins été nécessaire pour l'interface avec le matériel et pour les fonctions les moins abstraites. En effet, sur les plateformes STM32, la couche d'abstraction du matériel, ou Hardware Abstraction Layer (HAL), est développée en C. On doit pouvoir l'utiliser pour activer ou désactiver un composant, les paramétrer, et contrôler l'horloge, entre autres. Dans le cadre du stage, il a été nécessaire de pouvoir travailler dans chacune de ces couches, et donc de connaitre les langages associés.

Afin de prototyper le système complet, le choix d'une carte de développement « grand public » a été fait. Ainsi, la puce choisie est la STM32F401RE[11], qui a l'avantage d'être assez peu coûteuse et facile à déboguer. Les contraintes de développement sont donc un processeur à 84MHz, une mémoire vive disponible de 96Kio, et une mémoire flash disponible d'un total de 512Kio (incluant à la fois machines virtuelles, applications et leurs données).

1.4 Spécification GlobalPlatform

Si la création de Java Card a largement contribué à la standardisation des cartes à puce, de trop nombreux aspects sont néanmoins restés sans harmonisation sur le marché. La gestion des applications, leurs communications, leurs services, leur hiérarchisation, leurs droits, leur état sont autant d'éléments qui, autrefois, ont été laissés à la charge des manufacturiers. C'est dans ce contexte qu'a été créée, en 1999, l'organisation GlobalPlatform. Réunissant de nombreux acteurs des milieux de l'embarqué et des télécommunications, elle a, depuis plus de 20 ans, été au cœur de la création de nombreuses spécifications. Son document phare, la spécification de carte[12], est souvent désigné par le nom de l'organisation.

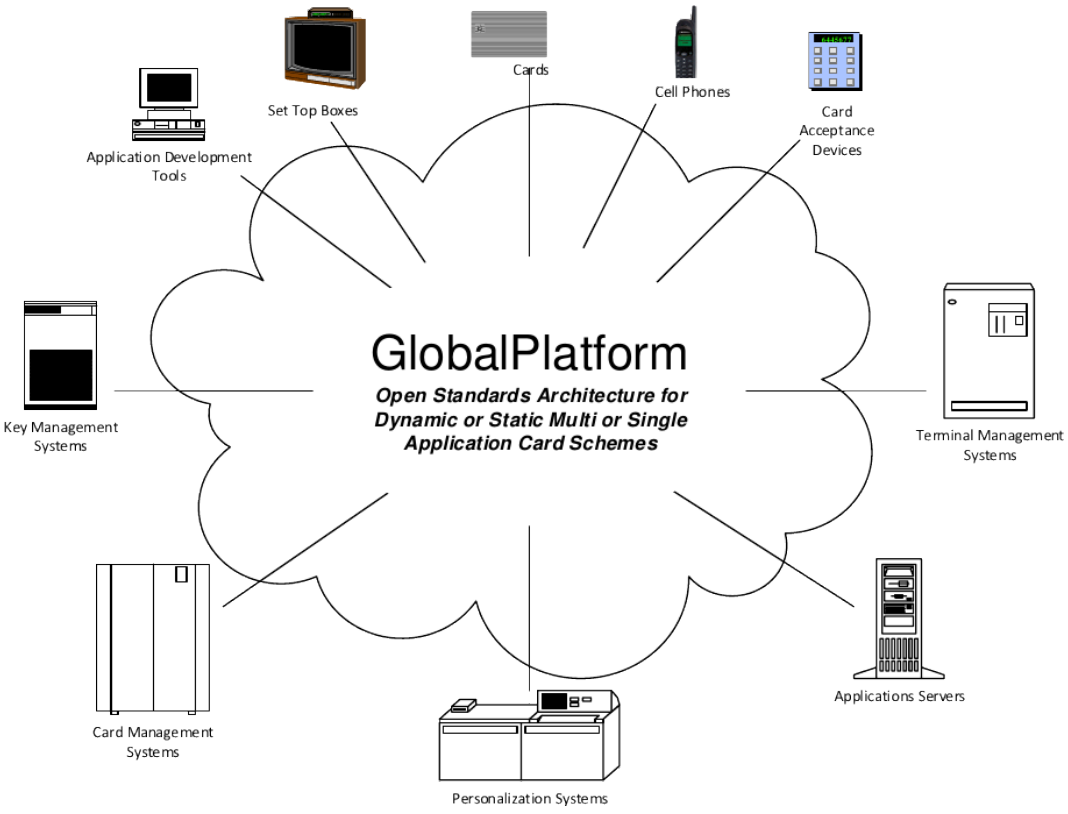

FIGURE 4 – De par son large ciblage, la spécification GlobalPlatform ne concerne pas seulement les cartes à puce, mais tout l'écosystème gravitant autour[12]

La spécification GlobalPlatform dispose donc d'un ensemble de règles qui prennent non seulement place au niveau des applications, mais aussi à celui des interac-

tions entre elles, et à celui des communications externes au périphérique. Ces contraintes et leur implémentation sont abordés dans les parties suivantes de ce document. On peut représenter le rôle de la spécification dans le système tel que fait dans la figure 5.

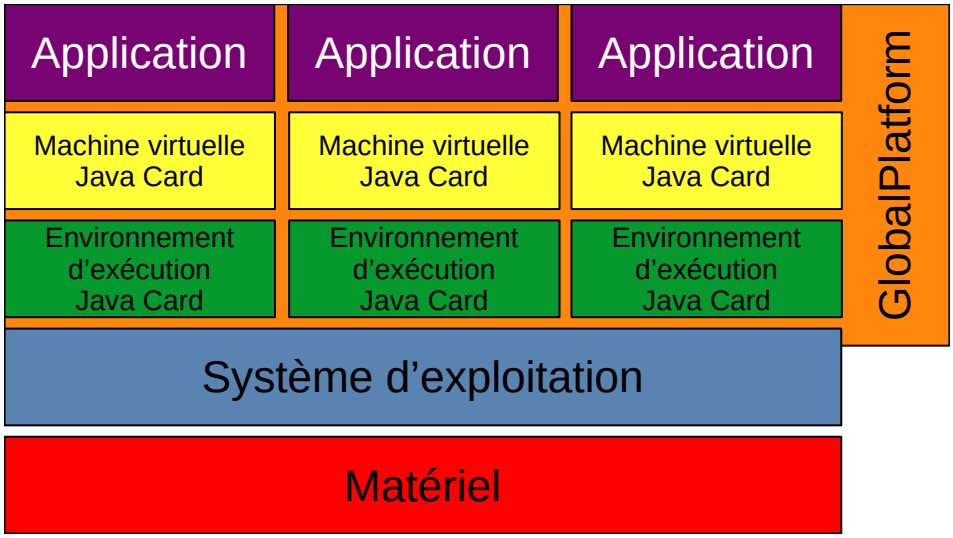

FIGURE 5 – Schéma d'abstraction prenant en compte les exigences de GlobalPlatform

Sa présence englobe la majorité des couches. De par son large spectre d'exigences, une implémentation de la spécification prend souvent une forme pour le moins prédominante auprès des applications. Dans le cadre de ce stage, il a été décidé d'adopter une position de démon privilégié qui obtient la main dès le lancement du système fini, ou dès qu'aucune application ne l'a. Ce paradigme permet de profiter des protections matérielles de la mémoire pour sécuriser les données sensibles. On garde également la possibilité d'interagir avec la machine virtuelle et l'environnement d'exécution par le biais de la même API qu'en temps normal.

Si la spécification GlobalPlatform est souvent implémentée sur des systèmes Java Card, elle reste cependant totalement indépendante de ce langage. Il est aussi possible de l'appliquer sur une plateforme MULTOS. MULTOS est un type de système d'exploitation également populaire sur les cartes à puce. On peut adapter cette architecture pour que la gestion de la sécurité respecte la spécification GlobalPlatform. Cette dernière peut même être implémentée sur une plateforme non

embarquée, mais certains de ses choix structurels nuiraient certainement à la puissance d'une telle installation.

2 Gestion de contenu

2.1 Organisation des contenus

2.1.1 Hétéroclisme des contenus

Une plateforme ne contient pas seulement des applications. Les cycles de vie et les architectures des applications imposent une politique de classification des contenus plus complexe. Étant donné que les contenus tiers sont développés en Java Card, on retrouve l'organisation typique inspirée par Java : un paquet contenant des classes qui peuvent être instanciées. Ce qu'on considère comme une application dans le paradigme Java Card est une instance. Cependant, suite à des réflexions sur le modèle de sécurité que l'on souhaite adopter, il apparait des besoins indispensables : ceux de pouvoir manipuler ou conserver des informations sur une application pas encore instanciée. Aussi, on veut pouvoir appliquer ces pouvoirs distinctement entre différentes classes d'un même paquet. On définit alors trois entités différentes qu'on manipulera avec notre agent de sécurité :

— Les fichiers de chargement exécutables, ou Executable Load Files (ELF) : ce sont des fichiers pouvant contenir plusieurs classes distinctes et instanciables. Équivalents à la notion de paquets en Java, les ELF se révèlent utiles pour regrouper différentes entités néanmoins fournies par le même acteur. On a alors la possibilité de leur appliquer, à toutes et avant instanciation, des politiques de sécurité grâce à ce dénominateur commun.

— Les modules exécutables, ou Executable Modules (EM) : ils correspondent aux données d'une application, consistant essentiellement en son code. On peut éventuellement en trouver plusieurs différents au sein d'un ELF. Ils permettent de déployer des politiques de restrictions ou de droit avant même l'utilisation du contenu.

— Les instances sont le résultat de l'instanciation d'un EM. Une instance désigne le dispositif mis en place pour exécuter un EM. Il peut y avoir plusieurs instances d'un même EM. Grâce à ce concept, on peut alors adapter les règles et permissions durant l'exécution, s'éloignant potentiellement de la politique établie à l'installation.

La mémoire vive étant limitée, on utilise la méthode d'exécution *execute in place* : le code est utilisé directement depuis la mémoire flash, on ne le copie pas dans la mémoire vive avant. Si on a plusieurs instances d'un même EM, alors elles liront toutes la même portion de mémoire flash durant leur exécution. Ce mode de fonctionnement est très économe puisqu'il évite de nombreuses duplications. Évidemment, certaines données ne peuvent pas profiter de cette centralisation. C'est notamment le cas des variables normalement stockées dans le tas : leur caractère imprévisible impose un stockage dans la mémoire vive.

La figure 6 présente un exemple de disposition de contenus sur une plateforme. Ici, une entreprise a développé un ELF. Il contient deux EM différents, présents sur la carte. Cependant seulement un seul est utilisé : l'environnement en a crée deux instances distinctes, utilisant le même code mais évoluant indépendamment.

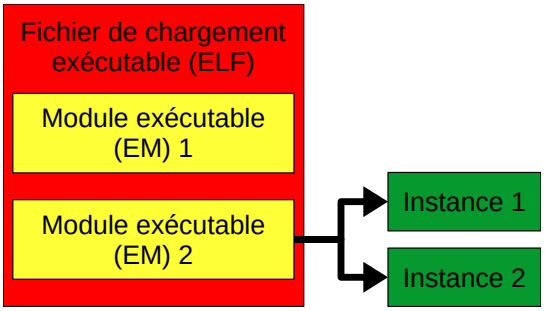

FIGURE 6 – Exemple de relations entre des instances, des EM et des ELF

2.1.2 Hiérarchisation des contenus

La politique de sécurité envers une application est souvent dépendante d'une autre application. Une instance peut vouloir contrôler les communications ou les actions d'une autre. La spécification GlobalPlatform énonce qu'une application pouvant avoir un tel contrôle se prénomme un domaine de sécurité. Du point de vue du démon de sécurité, un domaine de sécurité n'est qu'une application spécialement privilégiée pouvant avoir une autre instance sous son aile.

Pour répondre correctement aux exigences de hiérarchisation, on classera les contenus entre eux sous la forme d'un arbre, à l'instar de l'exemple présenté dans la figure 7.

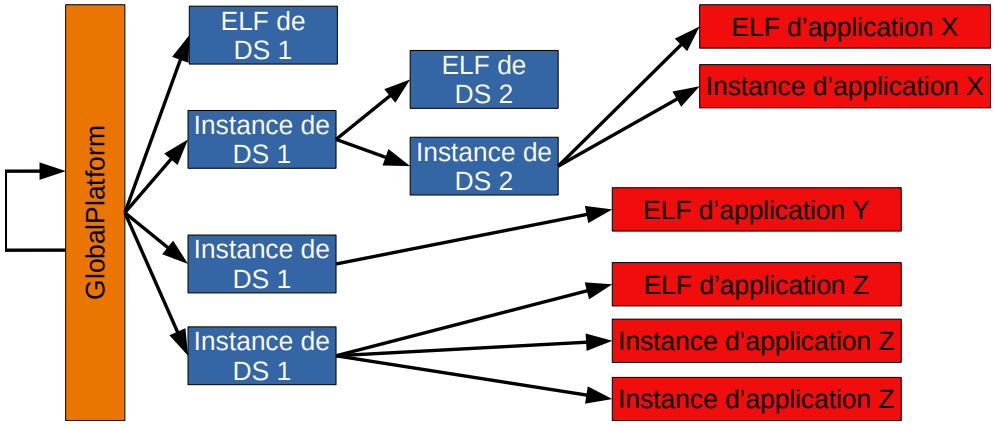

FIGURE 7 – Exemple de hiérarchisation des contenus

Chaque ELF, bien que ce ne soit que des données non exécutées, est enfant du domaine de sécurité qui l'a installé sur la plateforme. Toute instance est enfant du même domaine de sécurité que celui de l'ELF dont il provient. Tout domaine de sécurité est un enfant. Dans notre implémentation, le démon GlobalPlatform est considéré comme un domaine de sécurité. Comme il est à la racine de l'arbre, il est son propre enfant pour respecter la spécification. Enfin on notera qu'une instance, durant exécution, peut être extradée vers un autre domaine de sécurité si certaines conditions sont réunies. Cette possibilité est une illustration de la nécessité de conserver des informations à la fois sur un ELF et sur ses instances : les données sur l'ELF fournissent généralement un choix par défaut pour ses instances, et ces dernières peuvent modifier ces choix à l'exécution.

Sur un plan pratique, cette hiérarchisation est importante. Toute application voulant communiquer avec un acteur extérieur à la carte verra ses messages envoyés et reçus passer non seulement par le démon de sécurité, mais aussi par tous ses parents ; chacun des intermédiaires pouvant bloquer le routage pour une raison quelconque. Une application demandera régulièrement des services à son domaine de sécurité parent, notamment lorsque de la cryptographie et de la signature sont impliqués. Une instance d'application peut donc voir ses possibilités sérieusement bridées par son instance parente. Un opérateur téléphonique pourrait installer une application d'une entreprise tierce sous ses propres domaines de sécurité, et bloquer certaines actions selon l'état de l'abonnement de l'utilisateur final, par

exemple.

2.2 Base de donnée centralisée

Pour que le démon de sécurité puisse appliquer la politique voulue, il est dans l'obligation de stocker de multiples informations personnalisées par entité. Les ELF, EM, et instances ont chacun besoin de données individualisées. Si leur contenu est individuel, les types restent essentiellement les mêmes quelque soit la nature de l'entité. On choisit alors d'établir une structure commune capable de contenir les informations sur un ELF, un EM ou sur une instance. Notre base de donnée centralisée devient alors un large vecteur, où chacun des éléments est une telle structure.

Les informations qui se trouvent dans chacune d'entre elles sont les suivantes :

- Un indicateur permettant de désigner l'entité correspondante comme étant un ELF, un EM ou une instance.
- Un identifiant unique. On profite de l'existence, dans le monde Java Card, du concept d'Applet ID (AID). Il s'agit justement d'un identifiant unique qu'on va réutiliser ici. Le début de cette chaîne d'octets est défini selon le développeur de l'entité, la fin distingue l'entité elle-même.
- Le cycle de vie. Cette information sera détaillée dans la suite du document.
- L'entité parente pour pouvoir se localiser correctement dans la hiérarchisation.
- Les permissions, indispensables pour le démon de sécurité qui gère les droits.
- Les services globaux. Ce concept est abordé dans la suite du document.
- Les AID des EM ou des instances relatifs s'il s'agit, respectivement, d'un ELF ou d'un EM.

Pour les informations dont la taille peut varier, comme les services globaux ou les EM et instances relatifs, on définit un nombre maximal d'éléments. Cela permet d'appliquer une limite préservant la mémoire, et donc empêchant éventuellement des attaques sur cet aspect.

En amont de cette base de donnée, des outils ont été développés pour mieux l'exploiter. On peut ainsi facilement retrouver l'index d'une entité grâce à son AID.

La vérification d'unicité sur certains des éléments a été implémentée. Ce travail est nécessaire car, parmi les choses qui ont été amputées à Java lors de la création de Java Card, on trouve toutes les collections.

2.3 Mise à jour en temps réel du contenu

Quand il s'agit d'assurer la sécurité sur un environnement Java Card, il faut porter une attention particulière aux évolutions du contenu. Des entités peuvent être rajoutées, ou peuvent être personnalisées durant l'exécution : que ce soit en changeant leurs paramètres ou des données décisives. On encourt alors le risque d'exécutions arbitraires ou de comportements indéfinis. Des mesures décisives doivent alors être prises.

Dans un système Java Card en production, tout ELF destiné à être inclus dans la plateforme doit être vérifié et signé. En effet, il est possible d'attaquer la machine virtuelle avec une application malicieuse[13]. La principale technique consiste à utiliser la confusion de type qui exploite des défauts dans la conversion de type. Elle permet, via cette exécution anormale, d'exploiter des références non-prévues. Ce genre d'attaque est identifiable statiquement, c'est-à-dire en analysant seulement le code avant toute exécution. C'est le rôle du Byte Code Verifier (BCV). Il est le premier élément à agir après la compilation d'une application Java Card. Il effectue un certain nombre de vérifications qui s'assurent que le code intermédiaire est bien formé, et qu'il ne risque pas de causer une erreur voire d'exploiter une faille de la machine virtuelle. En effet, la vérification des types est coûteuse et n'est donc souvent pas effectuée lors de l'exécution du programme. Le BCV comble cette lacune en prouvant statiquement que les opérations sont bien typées, avant d'autoriser le chargement de l'ELF sur la carte.

Comme le BCV est appliqué de manière externe à la carte, il faut s'assurer lors de l'installation de l'ELF que ce dernier est bien passé par lui. C'est pourquoi lors de la vérification, une signature numérique est apposée, pour certifier ce passage. Le démon de sécurité doit donc vérifier cette présence pour accepter de recevoir l'application. Grâce à la cryptographie asymétrique, un domaine de sécurité peut utiliser une clé privée pour valider la signature et assurer que la source est de confiance. Tous les contenus sur la carte sont ainsi certifiés.

Pour modifier les données d'une application après instanciation, il est possible pour un acteur extérieur ou pour une autre application d'émettre une requête. Cette requête passe d'abord par une lecture et une interprétation pour vérifier que le message est correctement formé. Cette étape est détaillée par la suite dans ce document, lorsque les communications sont abordées. Ensuite, le message passera par tous les domaines de sécurité parents de l'instance de destination. Ce routage permet à chacun des parents d'appliquer sa propre politique de sécurité vis-à-vis de la requête. En imposant de tels filtres, on obtient de meilleurs remparts pour contrer les messages correctement rédigés mais qui tenteraient d'exploiter une faille de l'application cible.

2.4 Services Globaux

Dans le paradigme GlobalPlatform, une application peut revendiquer la possibilité de rendre un service à n'importe quelle autre application sur la carte ; de la même manière qu'un serveur local puisse répondre aux requêtes issues d'autres processus dans un système plus conventionnel. La principale différence provient du fait qu'ici, cette capacité est enregistrée dans l'environnement d'exécution, qui peut acheminer les demandes de manière invisible. Une application cliente demande ainsi un service, comme par exemple une authentification de l'utilisateur par son code personnel ; suite à quoi le démon de sécurité acheminera automatiquement cette demande (et sa réponse dans le sens inverse) vers l'application compétente. On dit qu'une telle application rend un service global.

Lors du chargement de l'ELF dans la carte, l'environnement apprend quelles sont les compétences potentiellement rendues par chacun des EM. Pour que les instances qui en découlent puissent réellement prendre en charge des demandes, il faut confirmer à l'exécution ce nouveau statut. Cela permet de s'assurer qu'aucune application proclame un service qu'elle ne sache pas rendre.

En temps normal, plusieurs applications peuvent rendre le même service. Le cas échéant, une instance cliente peut sélectionner le serveur qui répondra à sa requête. Cependant, une application peut demander à l'environnement d'enregistrer un service de manière unique. Si la procédure réussit, alors toutes les demandes seront redirigées vers elle sans autre forme de procès. Le choix du serveur ne sera plus possible. Il convient de préciser que le démon de sécurité n'enlève pas les

services globaux des autres instances sur demande d'un enregistrement unique. Ainsi, pour en obtenir un, il faut que toutes les autres applications y renoncent auparavant.

3 Permissions et états de vie

3.1 Application de privilèges

Primordiale dans un système sécurisé, la présence de privilèges est vérifiée sur l'environnement développé à l'ANSSI. Si la spécification GlobalPlatform dispose et décrit les différentes permissions, leur application technique reste relativement libre. Les actions modérées sont clairement énoncées mais les manières de restreindre sont variées.

Chaque entité (ELF, EM ou instance) a ses propres privilèges. Il n'existe pas de principe de groupe, ni de privilège par défaut si on tombe dans un cas non défini. GlobalPlatform spécifie 20 différentes permissions, dont on peut représenter l'état de manière binaire, car pour chacune d'entre elles, il n'y a que deux possibilités : possédée ou non. La figure 8 présente tous les privilèges de GlobalPlatform.

GlobalPlatform dispose d'une forme standardisée pour faire transiter l'ensemble des permissions. Durant l'installation d'une application, ou lorsqu'on récupère des informations sur une entité, il faut que tous les acteurs sur la carte se montrent compatibles avec la manière dont les privilèges sont présentés. Ainsi on doit utiliser trois octets, dont 20 des bits seront chacun utilisés pour représenter l'état d'une permission. Chaque privilège a son bit associé. Les bits restants sont réservés pour un usage futur.

Si la forme de l'ensemble des permissions est normalisée durant le transit, notre démon de sécurité peut lui stocker les permissions de chaque entité sans consigne aucune. On est donc libre dans la manière de conserver cette information. Dans le cadre de l'implémentation de l'ANSSI, une forme très similaire à celle décrite pour le transit a été conservée. Il convient seulement d'être vigilant lors du traitement d'un seul privilège : une erreur humaine dans la gestion des masques binaires est un réel danger.

Enfin, concernant l'exploitation et la vérification des privilèges, elles sont réalisées dans des milieux divers du démon de sécurité. Durant l'exécution, à chaque fois qu'une application tente une action susceptible d'être bloquée, un crochet logiciel dans l'environnement déclenche la vérification, et bloque la demande si nécessaire. On trouve alors de la gestion de permission dans la gestion des commu-

Nom officiel de la permission	Droit associé
Security Domain	L'application est un domaine de sécurité
DAP Verification	L'application peut vérifier une signature pour installation
Delegated Management	Le pouvoir de gérer le contenu de la carte a été délégué à cette application qui doit être un domaine de sécurité
Card Lock	L'application peut passer la carte dans l'état de blocage
Card Terminate	L'application peut passer la carte dans l'état de terminaison
Card Reset	L'application peut modifier les octets historiques de la carte
CVM Management	L'application peut gérer la méthode d'authentification fournie par une application d'authentification
Mandated DAP Verification	Le pouvoir de vérifier une signature pour installation a été délégué à cette application qui doit être un domaine de sécurité
Trusted Path	Cette application peut router des communications
Authorized Management	L'application peut gérer le contenu de la carte. Elle doit être un domaine de sécurité
Token Verification	L'application peut vérifier la délégation de gestion de contenu de la carte
Global Delete	L'application peut supprimer n'importe quel contenu de la carte
Global Lock	L'application peut verrouiller ou déverrouiller n'importe quel contenu de la carte
Global Registry	L'application peut accéder à l'intégralité de la base de donnée
Final Application	L'application est la seule accessible si la carte est verrouillée ou terminée
Global Service	L'application fournit des services globaux aux autres applications
Receipt Generation	L'application peut générer un reçu pour la gestion de contenu déléguée
Ciphered Load File Data Block	Cette application doit être un domaine de sécurité et son ELF associé doit être chargé chiffré
Contactless Activation	L'application peut activer ou désactiver n'importe quelle application (y compris elle-même) sur l'interface sans contact
Contactless Self-Activation	L'application peut s'activer elle-même sur l'interface sans contact sans demander à une application avec le privilège «Contactless Activation»

Figure 8 – Les permissions spécifiées par GlobalPlatform, chacune accompagnée par une traduction de sa description

nications, du contenu, des services, des interfaces, des états de vie…

3.2 Modélisation d'un système à états de vie finis

Certains aspects autour du mode d'utilisation d'une carte à puce influencent grandement son comportement voulu. Une carte étant légère et petite, elle peut facilement être perdue ou volée. De même, durant son cycle de vie, une carte va évoluer entre de nombreuses mains : manufacturier, exploitant, utilisateur, et peut-être celles d'acteurs non prévus. Il est naturel de souhaiter que selon le contexte, le système n'agisse pas de la même manière. En suivant la spécification GlobalPlatform, on implémente une machine à états finis pour la carte, et une autre pour chacune des applications présentes sur la carte.

La carte, à n'importe quel moment de sa vie, ne peut être que dans l'un de ces cinq états :

- **OP_READY** : l'environnement logiciel est stable. Le démon de sécurité est prêt à recevoir des requêtes. Cependant le système n'est pas complètement personnalisé et manque de données.
- **INITIALIZED** : cet état est proche d'OP_READY ; il sert à signaler que le système est personnalisé et alimenté en données mais n'est pas encore prêt pour être mis en production auprès du grand public.
- **SECURED** : le système est prêt pour la production et peut exercer en conditions réelles.
- **CARD_LOCKED** : la carte est bloquée. Ses fonctionnalités sont grandement limitées : le contenu ne peut être modifié, seule une application peut communiquer…
- **TERMINATED** : la carte est hors d'usage. Toutes les fonctionnalités sont bloquées excepté le traitement d'une requête GET DATA qui permet de récupérer une quantité minimale d'information sur le contenu.

Les états de vie ont été énumérés dans l'ordre le plus commun que rencontre une carte au cours de sa vie. Il est important de noter que les étapes ne sont pas simplement reliées entre elles de manière linéaire et irréversible. Par exemple une carte bloquée peut revenir en état sécurisé si certaines conditions sont réunies. Des

étapes peuvent être sautées. On peut dès lors constater l'utilité et le rôle pratique de ce schéma : un utilisateur légitime utilisera sa carte en mode sécurisé. Une perte ou vol signalé entraînera un passage en état verrouillé dès que possible, pour repasser dans le mode précédent si la situation est résolue. Dans le secteur bancaire, la carte est passée en état de terminaison une fois la date de validité passée ; cela équivaut à une destruction logicielle. Le matériel n'est cependant pas impacté et le système pourrait même être reprogrammé en partant de zéro.

Chaque application, à n'importe quel moment de sa vie, ne peut être que dans l'un de ces quatre états :

- **INSTALLED** : une instance de l'EM a été correctement créée. Toute création de lien ou allocation mémoire nécessaire a été faite. L'instance est répertoriée dans la base de donnée du démon de sécurité. Cependant l'application ne peut encore répondre aux requêtes.
- **SELECTABLE** : cet état est proche d'INSTALLED, il sert à signaler que l'instance peut désormais recevoir des requêtes.
- **LOCKED** : l'application est bloquée, elle ne peut plus être exécutée. Ainsi il lui est impossible de répondre aux requêtes.
- **Spécifique à l'application** : cet état regroupe en réalité un ensemble de sous-états définis par le développeur de l'application. Aucune garantie n'est fournie concernant ces sous-états. L'application sera exécutée et les requêtes peuvent lui être transmises, mais il est tout à fait possible que l'instance ne puisse appliquer aucun traitement de par son sous-état.

Là encore, les évolutions entre étapes ne sont pas linéaires. Par exemple, une application bloquée peut être débloquée par le démon, par un domaine de sécurité parent ou un domaine de sécurité assez privilégié. Une instance dans un état spécifique du développeur peut aussi retourner vers l'état sélectionnable.

Enfin, de manière accessoire, on peut constater que selon la spécification GlobalPlatform, les ELF ont un état dans lequel ils sont toujours sensés être : **LOADED**. Aucune évolution n'est ainsi possible. La principale utilité de ce cycle à une seule étape est de simplement attester que l'ELF a été correctement chargé et mémorisé sur la plateforme, et que le démon de sécurité a reconnu sa présence.

3.3 Influences entre état de vie et permissions

Il a été mentionné à quel point les états de vie peuvent se révéler utiles pour modérer la carte selon le contexte de l'utilisateur. Selon le type de restrictions, le cycle de vie va directement influencer les permissions. Le privilège «trusted path» peut être retiré si le domaine de sécurité concerné n'est pas dans un état conforme au routage de messages. À l'inverse, grâce à certaines permissions une application peut contrôler le cycle de vie d'une autre application ou de la carte. On peut notamment citer les privilèges «card lock» ou «card terminate», ainsi que «global registry». En effet, concernant cette dernière permission : si une instance a un accès complet à la base de donnée, il peut modifier des champs contenant des privilèges.

On notera également que les restrictions ne sont pas forcément appliquées au même niveau selon leurs origines. Les limitations induites par un état de vie ont tendance à être effectuée au-dessous de l'entité concernée, de manière opaque : elle ne peut même pas constater qu'elle a été privée d'une action. Au contraire, les limitations provenant des privilèges tendent à être notifiées à l'entité : elle obtiendra un retour lui indiquant un refus. On peut justifier cette différence avec le fait que les blocages réalisés par état de vie sont souvent dus à un risque de sécurité ou de corruption. Dans un tel contexte, on évite toute exécution dangereuse ; cela peut impliquer de priver une application d'un comportement par défaut en cas d'erreur, ou de toute exécution quelconque si besoin.

4 Isolation des contenus

4.1 Protection des contextes

Comme expliqué dans l'introduction aux systèmes Java Card, protéger le contexte d'exécution d'une application est primordial. Sans de tels mesures, on s'expose notamment à des risques de substitution de code. Grâce à la duplication des machines virtuelles proposée par l'implémentation de l'ANSSI, un fil virtuel ne peut accéder de manière classique aux classes des autres même s'il est malicieux. Vient alors la question des appels entre applications.

Dans un environnement Java Card, il n'y a pas de support pour des bibliothèques dynamiques. Le partage et l'accès direct à un code commun pose trop d'implications en termes de sécurité. Il existe cependant un moyen de demander à la machine virtuelle, depuis l'intérieur, de nous fournir un point d'accès pour appeler des fonctions extérieures. La bibliothèque standard Java Card fournit une interface nommée `Shareable`. En étendant cette interface, on peut déclarer des signatures de fonctions qui seront partageables entre contextes. On peut, en plus, rajouter des fonctions dans une classe qui implémente la précédente. Ces fonctions-là resteront privées, contrairement à celles déclarées dès l'extension de l'interface `Shareable`.

Lorsque l'environnement d'exécution Java Card reçoit une demande de fonctions partageables, il la transmet à l'application serveur, avec l'identifiant du client. L'instance serveur peut ainsi analyser la demande et a le choix :

- Refuser pour que l'environnement d'exécution renvoie une référence nulle ou lève une exception
- Accepter pour que l'environnement d'exécution renvoie une référence correcte, permettant d'utiliser une implémentation de la classe partagée

À chaque fois que le client utilisera une fonction du serveur, il y aura changement de contexte pour passer sur celui du serveur. Ce dernier élément est important car on garde ainsi l'isolation des deux côtés.

Avec cette possibilité, on peut alors conceptualiser l'organisation du démon de sécurité. Pour illustrer les propos, la figure 9 est mise à disposition. Pour les nombreux démarrages que va exercer la carte, il faut un point de démarrage fixe

et facilement atteignable. C'est le singleton central. Il est statique, permanent, et réserve sa mémoire durant l'installation du système. On évite ainsi des soucis d'allocation à l'exécution.

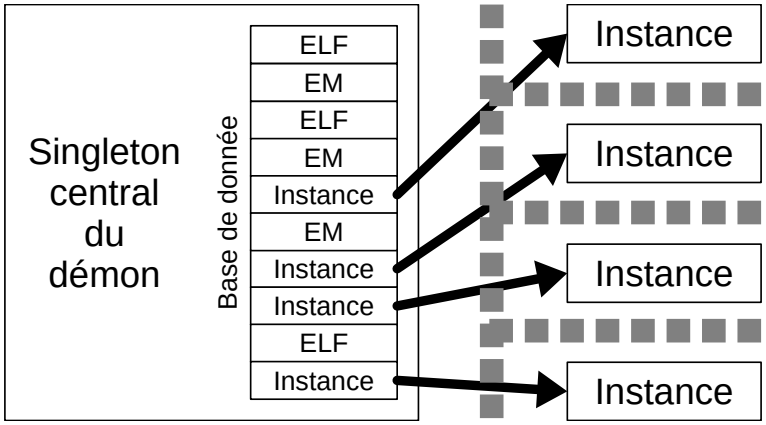

FIGURE 9 – Architecture du démon de sécurité et ses liens avec les instances tierces ; les lignes en pointillés représentent les isolations de contexte fournies par la machine virtuelle Java Card

Comme expliqué auparavant, il a été choisi d'entretenir la base de donnée comme un vecteur ou chaque élément renvoie vers une entité quelconque, c'est-à-dire vers un ELF, un EM, ou une instance. En réalité, chacun de ces éléments est une classe partageable. Si elle est associée à une instance, alors une application cliente peut la demander. Elle n'aura pas accès aux données et aux attributs ; mais seulement à certaines fonctions méticuleusement choisies pour tolérer seulement les interactions raisonnables.

4.2 Protection des données

Le système est prévu pour fonctionner avec un adressage mémoire commun. Ainsi, la mémoire flash, la mémoire vive, et les périphériques sont dans le même espace. Comme expliqué auparavant, le système d'exploitation ségrègue différentes régions personnalisées de la mémoire et y applique des politiques personnalisée. Une classe ou application ne peut accéder aux attributs d'une autre, même en connaissant son adresse.

Cette protection se révèle efficace, au point où il devient délicat de partager des données entre applications. On sait qu'une région mémoire est prévue pour faire office de tampon. C'est cette zone qui est utilisée pour faire transiter des paramètres lors d'appels de fonctions partageables citées auparavant. Comme il est fréquent, dans ce système Java Card, de faire transiter un message sur un chaîne de domaines de sécurité, ce tampon doit alors être réinitialisé à chaque étape.

Lorsqu'une application passe en état bloqué, ses données sont verrouillées. En effet, dans ce cas, l'instance ne peut plus être invoquée ou exécutée. Ainsi sa zone d'adresse n'est jamais cartographiée par l'unité de protection mémoire. Grâce à cet état de vie on bénéficie alors d'une mesure matérielle assurant l'isolation d'une application en danger (ou à l'inverse qui représente un risque). Si la carte elle-même passe dans l'état bloquant, le même phénomène se reproduit : toutes les applications sur la carte ne peuvent plus être lancées et voir leur mémoire réouverte. Seul un sous-ensemble fortement restreint de l'environnement est disponible, à la manière d'un mode sans échec relativement strict. Le déclenchement d'une telle protection matérielle se révèle nécessaire face à la possibilité d'une attaque par injection de faute[14].

La protection de la mémoire ne se fait pas seulement contre l'écriture ou l'écrasement. On souhaite certes éviter la perte de donnée, mais il avant tout primordial d'éviter la fuite ; d'où la protection contre la lecture également. Lorsque la carte est bloquée, on vise alors un modèle de sécurité où une personne physique est dans l'impossibilité d'accéder aux données d'applications, qui contiennent peut-être des clés privées. Il devient alors nécessaire de posséder du matériel de pointe pour lire électromagnétiquement des bits. Cette technique, dont le résultat n'est pas garanti, n'est accessible qu'aux États ou aux grands groupes privés, de par les moyens requis.

Durant le développement, il faut rester vigilant concernant les interfaces qu'on conserve. Déboguer une plateforme Java Card avec les Application Protocol Data Units (APDU) sur le bus ISO/IEC-7816[3] est difficile, voir impossible si des applications ou la carte entière sont bloquées. Il faut veiller à conserver des bus de secours, on peut par exemple conserver une ligne Universal Asynchronous Receiver Transmitter (UART) sur un système Nucleo STM32.

4.3 Communications

4.3.1 Extra-carte

Une carte à puce ne peut fonctionner que lorsqu'elle est auprès d'un lecteur. En effet, elle est alimentée soit l'interface avec contact ou celle sans contact. Dès lors, les communications avec l'extérieur sont primordiales dans le fonctionnement d'une telle plateforme. Comme avec Java Card, elles ont été normalisées pour assurer une interopérabilité la plus grande possible. C'est, entre autres, le rôle de l'ISO/IEC-7816[3]. La partie 3, en particulier, standardise les messages entrants et sortants de la carte. Ils sont appelés Application Protocol Data Units (APDU). Un APDU peut être de l'un des deux types présentés en figure 10 : commande ou réponse.

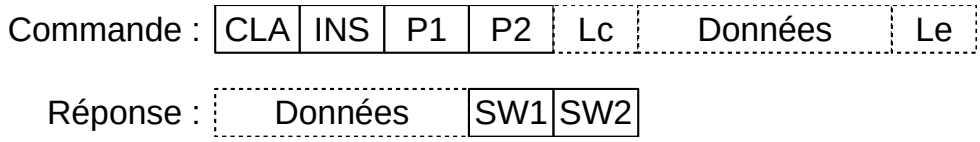

FIGURE 10 – Structure d'APDU de commande et de réponse. Les cases formées par des pointillés représentent des champs facultatifs

Une commande commence par un en-tête obligatoire de 4 octets. Le premier octet, CLA, indique le type de l'instruction. Cela permet notamment de différencier entre une commande propriétaire ou une commande standardisée. Le second désigne l'instruction en elle-même. P1 et P2 sont quant à eux deux octets de paramètres pour l'instruction. Le champ Lc désigne le nombre d'octets que fait le paquet de donnée suivant. Le, quant à lui, quantifie le nombre maximal d'octets que peut faire le paquet de donnée dans la réponse.

La réponse commence donc par les données, suivies de deux champs : SW1 et SW2. Ces deux octets obligatoires servent à témoigner d'un statut vis-à-vis de la commande demandée. La partie 4 de l'ISO/IEC-7816[3] définit des codes pour de nombreuses situations, qu'il y ait eu une erreur ou non. Ainsi, le lecteur de carte peut recevoir une confirmation d'exécution ou un code d'erreur selon la situation.

La norme déclare certains champs comme étant facultatifs. Si Lc est absent, alors cela signifie qu'il y a zéro octet de donnée. Même chose avec Le, qui peut annuler la présence de donnée dans la réponse s'il n'est pas envoyé.

Lorsque le lecteur de carte émet un APDU, celui-ci monte les couches d'abstraction jusqu'à arriver au démon de sécurité. C'est-à-dire qu'il passe par l'interface physique, le système d'exploitation, et l'environnement Java Card avant de parvenir au démon. Comme ce dernier a pour rôle de contrôler les communications, c'est lui qui va réaliser une première étape d'interprétation du message. Cela permet de vérifier au moins la bonne structure du message, et de bloquer sa transmission si nécessaire. Le démon vérifie aussi que le destinataire est en mesure de recevoir une requête : l'application peut-être inexistante, bloquée… Son rôle s'arrête ici, puisque seules les instances cibles elles-mêmes peuvent vérifier des contraintes propres à elles, notamment concernant le contenu du champ de donnée.

4.3.2 Intra-carte

Les différentes instances présentes sur une plateforme ont plusieurs moyens d'interagir entre elles. Ces manières reposent, au fond, toutes sur le système de classe partageable et de tampon de paramètres, expliqués précédemment.

Les interfaces de programmation applicative fournies aux instances n'ont pas forcément besoin d'un mécanisme de tampon de partage. C'est le cas lorsqu'une application veut communiquer avec le démon de sécurité , car il bénéficie d'une isolation typique d'application grâce à sa machine virtuelle. Cependant, ça ne peut pas être le cas de l'environnement lui-même. Ainsi, lorsqu'une instance utilise l'interface vers sa machine hôte, le tampon n'est pas nécessaire, car la machine virtuelle tourne sous le même contexte que son processus invité. Si cette possibilité permet un échange bien plus rapide, cet avantage est difficilement exploitable sans casser la correcte isolation des applications.

L'échange de services globaux, présentés plus tôt dans ce document, se réalise de manière plus complexe. Lorsqu'une instance demande la réalisation d'un service, la requête est d'abord passée au démon de sécurité. Il en vérifiera divers aspects. Il se doit notamment de vérifier que le service est proposé par une quelconque application. S'il est proposé plusieurs fois, il faut vérifier la suggestion de choix proposé par le client. Le démon réalise également un contrôle de permissions et d'états de vie. C'est seulement dans un second temps, si toutes les vérifications précédentes sont valides, que la requête passe dans un second stade de contrôle. Il devient ici possible de faire des vérifications plus propre à l'application serveur, car

on connaît ici mieux ses exigences. La requête se situe dans une classe personnalisée par instance de fourniture de service. On contrôle alors la requête en elle-même, ses paramètres, et éventuellement l'authentification de l'application cliente.

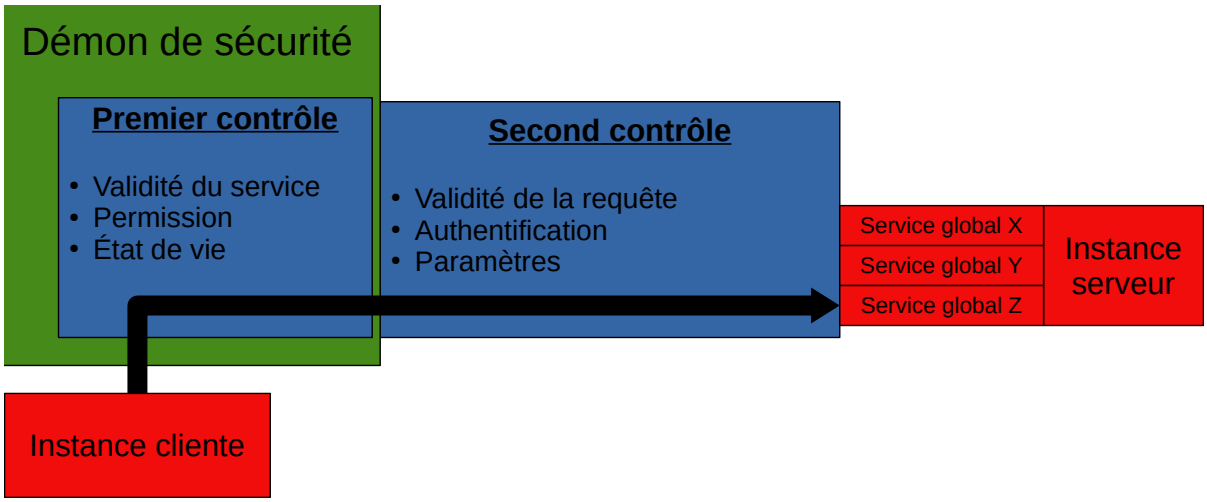

FIGURE 11 – Illustration du contrôle multi-étape d'une demande de service global

Conclusion

Lors de ce stage, un démon gérant plusieurs aspects de sécurité d'un environnement Java Card a été développé. Associé au système ouvert développé à l'ANSSI, on obtient, avec toutes les couches développées en dessous, une structure accessible et pensée pour la sécurité ; qui plus est facilement utilisable et exploitable.

Le démon de sécurité a été pensé pour pouvoir superviser l'intégralité du contenu de la carte à tout moment, pendant le fonctionnement. On a alors un compromis entre ouverture et gestion du risque qui nous permet d'intégrer en sécurité du contenu tiers. L'organisation de cette gestion a été prévue pour correspondre aux contraintes de l'embarqué : mémoire et puissance de calcul limitées.

Grâce à l'implémentation de permissions individuelles pour chaque instance, on peut contrôler les droits avec une granularité fine, en restant en adéquation avec la spécification GlobalPlatform. Les états de cycle de vie permettant d'adopter une politique spécifique à une instance ou à toute la carte ; on obtient un outil utile pour l'exploitant car il adapte le fonctionnement de la carte en fonction de son contexte extérieur. Tous ces éléments rassemblés offre de grandes possibilités de paramétrage de la sécurité sur la plateforme.

La protection des contextes, primordiale sur un tel système, a bénéficié d'une attention particulière et fait collaborer toutes les couches d'abstraction. Avec le démon de sécurité, un développeur tiers obtient des techniques standardisées de partage de services et de requêtes. Il exploite également les protections matérielles de la mémoire pour gérer les situations de risque. Et les communications, essentielles, sont contrôlées pour assurer leur sécurité.

Près de 2500 lignes de code ont été rédigées durant ce stage. Pour assurer la réutilisabilité, certaines mesures ont été prises autour du projet. La compilation du démon a été incluse dans le Makefile de construction de l'API Java Card. L'intégralité de la contribution est documentée à l'aide de Javadoc. Un fichier de type «lisez-moi» a été rédigé pour guider une première utilisation. Comme pour n'importe quelle application Java Card, tout le code du démon a été vérifié par un BCV pour éviter les comportements à risques. Le gestionnaire de version Git a été utilisé durant tout le développement, offrant ainsi la possibilité de revenir à des versions antérieures.

Pour l'environnement Java Card complet, les possibilités futures sont nombreuses. Le BCV appliqué statiquement et à l'extérieur de la carte souffre de défauts qu'il convient de corriger[15]. La recherche autour des failles en canaux auxiliaires fait de nombreux progrès.[16]. De plus, le caractère ouvert du système en fait un excellent cadre de recherche pour améliorer n'importe quelle couche.

En ce qui concerne le démon de sécurité, quelques aspects restent à finaliser. En effet, les primitives cryptographiques doivent être implémentées par des personnes formées en la matière. Les canaux logiques, fonctionnalité facultative permettant des communications virtuellement parallèles avec plusieurs applications à la fois, n'a pas été implémentée par manque de temps. La spécification GlobalPlatform n'étant pas obsolète, il faut également suivre ses versions futures.

Après les standards vidéo et l'intelligence artificielle, ce stage m'a apporté un nouvelle expérience du monde de la recherche : une dans la sécurité informatique. J'ai pu découvrir le travail dans une agence publique composée de personnes à la fois talentueuses et bienveillantes. J'ai été initié au travail dans un contexte sensible et confidentiel, et été entouré de nombreux experts à l'état de l'art sur leur domaine.

Ainsi je tiens à remercier l'ANSSI pour m'avoir offert cette expérience instructive et chaleureuse, et l'Institut National des Sciences Appliquées (INSA) pour son suivi au sein de ces six mois. J'exprime particulièrement ma gratitude à Guillaume Bouffard, mon tuteur de stage, pour cette opportunité et pour m'avoir guidé et aidé durant ce travail.

Ce stage se pose dans la continuité de multiple immersions dans la recherche, que ce soit au département Électronique et Informatique Industrielle (EII) de l'INSA, ou au Rochester Institute of Technology (RIT). En particulier, la recherche dans la sécurité informatique, étant un domaine qui me plaît, représente une réelle perspective de carrière pour moi. La réalisation d'un doctorat n'a jamais autant été envisageable dans mes projets professionnels.

Mis à part mes affinités personnelles, le secteur de la sécurité informatique représente une voie d'avenir. Beaucoup d'industriels ont, notamment grâce à l'ANSSI, pris conscience des risques numériques. Le développement du tout connecté entraîne une forte demande de personnes compétentes sur le marché du travail. Les enjeux seront nombreux dans le futur.

Acronymes

AID Applet ID. 16

ANSSI Agence Nationale de la Sécurité des Systèmes d'Information. II, 1, 2, 7, 8, 20, 25, 31, 32

APDU Application Protocol Data Unit. 27–29

BCV Byte Code Verifier. 17, 31, 32

CC Critères Communs. 7

CSPN Certification de Sécurité de Premier Niveau. 3

DGA Direction Générale de l'Armement. 3

EII Électronique et Informatique Industrielle. 32

ELF Executable Load File. 13–18, 20, 21, 23, 26

EM Executable Module. 13, 14, 16, 18, 20, 23, 26

HAL Hardware Abstraction Layer. 9

INSA Institut National des Sciences Appliquées. 32

LSC Laboratoire Sécurité des Composants. III, 3, 7, 8

MPU Memory Protection Unit. 9

OIV Opérateur d'Importance Vitale. 1, 2

RIT Rochester Institute of Technology. 32

SDA Sous-Direction Administration. 2

SDE Sous-Direction Expertise. 2

SDN Sous-Direction du Numérique. 1

SDO Sous-Direction Opération. 1

SDS Sous-Direction Stratégie. 1

SGDSN Secrétariat Général de la Défense et de la Sécurité Nationale. 1, 2

SIM Subscriber Identity Module. 4

SSTIC Symposium sur la Sécurité des Technologies de l'Information et des Communications. 9

UART Universal Asynchronous Receiver Transmitter. 27

Références

[1] Mark Hung. Leading the IoT : Gartner Insights on How to Lead in a Connected World, 2017. https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

[2] Yves Grandmontagne. Face aux risques de l'iot... l'indifférence des responsables informatiques, October 2018. https://itsocial.fr/enjeux/securite-dsi/cybersecurite/face-aux-risques-de-liot-lindifference-responsables-informatiques/.

[3] ISO/IEC 7816. https://www.iso.org/search.html?q=ISO%2FIEC%207816&hPP=10&idx=all_en&p=0&hFR%5Bcategory%5D%5B0%5D=standard.

[4] Oracle. *Java Card 3 Platform, Virtual Machine Specification, Classic Edition, Version 3.0.4.* Oracle, September 2011.

[5] Oracle. *Java Card 3 Platform, Runtime Environment Specification, Classic Edition, Version 3.0.4.* Oracle, September 2011.

[6] Apple. ios security guide, 2019. https://www.apple.com/business/docs/site/iOS_Security_Guide.pdf.

[7] Anne-Marie LESAS. *Vers un environnement logiciel générique et ouvert pour le développement d'applications NFC sécurisées.* Université d'Aix-Marseille, September 2017.

[8] Dangerous Things. Vivokey flex one. https://dangerousthings.com/product/vivokey-flex-one/.

[9] Guillaume Bouffard et Léo Gaspard. Hardening a java card virtual machine implementation with the mpu, June 2018. https://www.sstic.org/2018/presentation/hardening_a_java_card_virtual_machine_implementation_with_the_mpu/.

[10] Jim Blandy. *Why Rust ?* O'Reilly, September 2015.

[11] STMicroelectronics. *RM0368 Reference manual, STM32F401xB/C and STM32F401xD/E advanced ARM ® -based 32-bit MCUs.* STMicroelectronics, May 2015.

[12] GlobalPlatform. Globalplatform card specification v2.3.1, March 2018. `https://globalplatform.org/specs-library/card-specification-v2-3-1/`.

[13] Erik Mostowski, Wojciech et Poll. Malicious code on java card smartcards : Attacks and countermeasures. In François-Xavier Grimaud, Gilles et Standaert, editor, *Smart Card Research and Advanced Applications*, pages 1–16, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[14] Tiana Razafindralambo et Guillaume Bouffard et Jean-Louis Lanet. A Friendly Framework for Hidding fault enabled virus for Java Based Smartcard. In Nora Cuppens-Boulahia et Frédéric Cuppens et Joaquín García-Alfaro, editor, *DBSec 2012, Paris, France, July 11-13,2012. Proceedings*, volume 7371 of *Lecture Notes in Computer Science*, pages 122–128. Springer, 2012.

[15] Aymerick Savary et Marc Frappier et Jean-Louis Lanet. Detecting Vulnerabilities in Java-Card Bytecode Verifiers Using Model-Based Testing. In Einar Broch Johnsen et Luigia Petre, editor, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2013.

[16] Guillaume Bouffard et Bhagyalekshmy N. Thampi et Jean-Louis Lanet. Security automaton to mitigate laser-based fault attacks on smart cards. *IJTMCC*, 2(2) :185–205, 2014.

— FIN DU DOCUMENT —