

Synthèse des méthodes garantissant la sécurité des cartes multi-applicatives Introduction des cartes multi-applicatives ouvertes

Damien Sauveron^{*,**}

LMSI, Laboratoire Méthodes et Structures Informatique
Université de Limoges
83 rue d'Isle, 87000 Limoges, FRANCE.
damien.sauveron@unilim.fr, <http://msi.unilim.fr/>

Résumé Jusqu'à la fin des années 1990, il était impossible d'avoir plusieurs applications sur une même carte à puce. Aujourd'hui de nombreux standards de cartes multi-applicatives, et en particulier la technologie Java Card, permettent de faire cohabiter plusieurs applications sur le même support. La flexibilité de ces cartes laisse augurer le développement de nouveaux services, grâce à la possibilité offerte de charger dynamiquement les applications après leur émission, mais elle soulève aussi quelques craintes auprès des clients industriels et gouvernementaux. En effet, si le dynamisme de ces plates-formes permet d'abaisser les coûts de déploiement des applications, il crée également de nouveaux chemins d'attaques. Ainsi, pour obtenir une assurance forte quant à la sécurité des cartes multi-applicatives, plusieurs techniques sont utilisées. L'objectif de cet article est de dresser un état de l'art des méthodes d'assurance de la sécurité des cartes multi-applicatives. Il présente également l'évolution pressentie vers des cartes multi-applicatives ouvertes et les problèmes auxquels elles seront confrontées.

MOTS-CLÉS : *Carte multi-applicative, Sécurité, Évaluation sécuritaire, GlobalPlatform, Méthodes formelles, Attaques*

1 Introduction

Dans le domaine des cartes à puce, mettre en œuvre la sécurité est toujours une tâche très difficile. Cela se révèle particulièrement vrai dans le cadre des cartes à puce multi-applicatives. Bien loin de vouloir décrire de manière exhaustive tous les mécanismes sécuritaires embarqués sur les cartes, cet article s'intéresse aux techniques d'assurance employées pour protéger les biens embarqués (*i.e.* cela peut être du code ou des données) sur les cartes multi-applicatives. Notre introduction présente un résumé des grandes idées que nous développerons dans la suite de notre article.

1.1 Contexte : le développement sécurisé sur cartes à puce

La spécification et le développement des systèmes d'exploitation embarqués sont des phases délicates qui nécessitent la mise en commun de compétences fortes en sécurité informatique, en électronique mais également en cryptologie. En effet, les travaux menés lors

^{*} LaBRI (Talence, FRANCE) et LMSI (Limoges, FRANCE).

^{**} Ce travail a été partiellement réalisé durant ma thèse au sein de l'équipe *Systèmes et Objets Distribués* du LaBRI.

de la dernière décennie ont montré qu'il était possible de réussir des attaques matérielles sur les cartes à puce avec un équipement parfois rudimentaire – par exemple en exploitant les canaux cachés [1,2,3,4,5]. Dès lors, la fabrication de circuits toujours plus sécurisés [6,7] est un préalable indispensable à la conception d'architectures matérielles et logicielles totalement sûres. Par ailleurs la sécurité ne relevant pas de la seule responsabilité de la plate-forme, les applications destinées à être embarquées sur les cartes (originellement mono-applicatives fermées) doivent impérativement utiliser les services de sécurité offerts par la plate-forme sous-jacente de façon à ne pas mettre en péril leurs propres biens ou même ceux de la plate-forme. Ainsi, devant l'importance des enjeux économiques des marchés exploitant la carte à puce, les gouvernements ont mis en place des schémas de certification sécuritaire des produits des technologies de l'information (TI) afin d'établir la sécurité des produits certifiés. Dans ce cadre, des tiers de confiance indépendants sont chargés de conduire des évaluations sécuritaires de produits TI en examinant le code du produit, les documentations de développement et les éléments de preuve nécessaires afin de s'assurer que le produit du fabricant fait bien ce qu'il prétend faire et de façon efficace. Au terme de ces processus de développement et de certification, la carte à puce et l'application embarquée constituent en principe l'un des systèmes les plus sécurisés du marché.

1.2 Problématique : les technologies multi-applicatives des cartes à puce

Si ces étapes permettent d'assurer l'obtention d'un produit possédant un très haut degré de sécurité, c'est au prix d'un coût élevé aussi bien pour la fabrication du produit (*i.e.* puce sécurisée, OS sécurisé, application sécurisée) que pour l'évaluation et la certification de ces différents éléments. De plus, le processus de masquage est souvent très long et ne permet pas de répondre efficacement aux besoins extrêmement volatiles des marchés. Ainsi, afin de réduire ces coûts, mais aussi afin d'avoir une plus grande flexibilité et donc d'être plus réactif aux demandes des industriels utilisant les cartes, des technologies de cartes multi-applicatives ont été proposées. À terme, ces technologies devraient permettre une réduction des coûts à plusieurs niveaux :

- lors du développement, grâce principalement à l'utilisation de langages de programmation de haut niveau qui simplifient considérablement la mise au point des applications, mais également grâce à l'utilisation d'une plate-forme commune comme support d'exécution d'applications pouvant être de natures très diverses ;
- lors de l'évaluation et de la certification, puisqu'il suffit théoriquement d'évaluer et de certifier une seule fois la plate-forme puis au coup par coup les applications qu'elle devra embarquer, plutôt que d'évaluer l'ensemble à chaque fois qu'on veut installer une nouvelle application.

Ces technologies répondent aux besoins de flexibilité et de réactivité en fournissant le dynamisme qui manquait jusqu'alors aux cartes à puce classiques, en autorisant le chargement d'applications même après la phase de fabrication, et en permettant aux fournisseurs d'applications de faire jouer la concurrence entre les divers fabricants de cartes proposant une implantation de la même technologie. De plus, comme ces cartes peuvent gérer plusieurs applications au sein d'un même environnement sécurisé, elles permettent aussi de développer des applications collaborant de manière sécurisée sur un même support – chose auparavant

assez difficile. Nous décrirons les différentes technologies de cartes multi-applicatives et leurs caractéristiques communes lors de la section 2.

Néanmoins, si ces technologies multi-applicatives résolvent beaucoup des problématiques du monde de la carte à puce, elles en créent également de nouvelles et notamment dans le domaine essentiel de la sécurité. Par exemple, puisqu'il est possible d'embarquer plusieurs applications de différents fournisseurs, un émetteur d'application potentiellement malhonnête, ou tout simplement peu informé des recommandations sécuritaires, pourrait charger une application malicieuse ou involontairement mal programmée qui risquerait de mettre en péril la sécurité de toute la plate-forme. Aussi, afin de contrer de telles menaces, et tant que ces technologies n'ont pas été prouvées sûres à 100% à l'aide de méthodes formelles, plusieurs procédures et mécanismes d'assurance ont été mis en place. Parmi les procédures, le recourt à l'évaluation sécuritaire de la plate-forme et de l'application à charger permet de s'assurer de leur bon fonctionnement et de l'efficacité des fonctions de sécurité mises en œuvre. L'implantation sur ces cartes multi-applicatives du standard GlobalPlatform [8] permet aussi de restreindre le chargement des applications qu'aux seules personnes dûment authentifiées. Nous présenterons donc dans la section 3, les techniques d'assurance pour la sécurité des cartes multi-applicatives que constituent : l'évaluation sécuritaire et la certification, le standard GlobalPlatform et les méthodes formelles.

1.3 Perspectives : les cartes multi-applicatives ouvertes ?

Aujourd'hui, ces deux exemples illustrent bien l'état de l'art des méthodes utilisées pour assurer la sécurité d'une plate-forme multi-applicative sur carte à puce. S'il est certain que ces procédures conduisent à améliorer la sécurité globale du système, c'est au détriment de sa flexibilité. Heureusement, dans un futur proche et au vu des travaux de recherche récents sur les vérificateurs embarqués [9], il devrait être possible de disposer de plates-formes sécurisées et totalement ouvertes. Dans ce cadre, la plate-forme sera évaluée et certifiée une fois et une seule et les applications pourront être chargées dessus sans aucune restriction puisque la plate-forme assurera la sécurité d'une application vis-à-vis des autres. Ainsi, les industriels n'auront plus qu'à faire évaluer les applications critiques nécessitant une assurance forte quand au fait qu'elles ne dévoilent pas malgré elles leurs biens indépendamment de toute plate-forme (*i.e.* il faudra juste vérifier qu'elles sont bien programmées en accord avec les spécifications de l'application et il ne sera plus nécessaire de prendre en compte tout l'environnement puisqu'elle sera destinée à être chargée sur une plate-forme déjà évaluée et certifiée). Toutefois de nouvelles menaces pèsent sur ces cartes. Ainsi, lors de la section 4 nous définirons ce que sont les cartes multi-applicatives ouvertes et nous présenterons les nouveaux problèmes auxquels elles auront à faire face.

2 Les cartes multi-applicatives

Comme nous l'avons évoqué lors de l'introduction, historiquement les cartes à puce étaient mono-applicatives. À la fin des années 1990, avec le développement des télécommunications, une tendance se dégageait sur le marché des Technologies de l'Information (TI) : la flexibilité. Les besoins du marché des TI devenaient de plus en plus volatiles et nécessitaient souvent

d'être capable de déployer rapidement des applications sécurisées sur cartes à puce pour répondre à des demandes de nouveaux services (ponctuelles ou non). Cependant, le déploiement via des processus de masquage des nouvelles applications sur de nouvelles cartes étant incompatible avec les besoins du marché tant en terme de délai que de coût, les grands acteurs du secteur de la carte se sont regroupés pour mettre en place des standards de technologies multi-applicatives. Ces dernières permettraient à terme la réduction des coûts de fabrication et d'évaluation pour le client final grâce au dynamisme (*i.e.* à la possibilité de charger des applications à différents instants du cycle de vie de la carte) mais également grâce à l'interopérabilité (et donc grâce aussi à la concurrence que se livrent les différents fabricants de cartes implantant la même technologie multi-applicative).

De façon évidente, une carte multi-applicative est une carte capable d'embarquer plusieurs applications. En revanche, multi-applicatif, ne signifie pas multi-tâches. Les applications ne s'exécutent pas de façon concurrente car les systèmes d'exploitation des cartes à puce possèdent une unique *thread*.

Les deux principaux standards de cartes multi-applicatives sont Java Card [10,11] et MULTOS [12]. Récemment, deux nouvelles technologies les ont rejoint avec la naissance de Smartcard.NET [13,14] de Hive-Minded et la MultiApplication BasicCard ZC6.5 [15] de Zeit-Control. Windows for Smart Cards de Microsoft peut également être cité même s'il semble aujourd'hui défunt. Tous partagent les mêmes concepts et l'architecture d'une carte implantant ces technologies (*cf.* Fig. 1) consiste en :

- un système d'exploitation embarqué qui supporte le chargement et l'exécution de plusieurs applications (chacune à leur tour). Ce système d'exploitation est en fait constitué d'un environnement d'exécution et d'une machine virtuelle (VM) qui mettent en place des caractéristiques sécuritaires (e.g. un pare-feu entre les applications). Dans ce système, on peut se représenter la machine virtuelle comme un processeur virtuel sur lequel fonctionne un environnement d'exécution qui permet de cacher la complexité sous-jacente de la carte en fournissant des APIs de haut niveau (*i.e.* des bibliothèques universelles qui seront présentes sur toutes les cartes implantant la même technologie).
- des applications qui sont interprétées par la VM. Elles s'exécutent donc au dessus du processeur virtuel et demandent des ressources et des services systèmes à l'environnement d'exécution au travers des APIs.

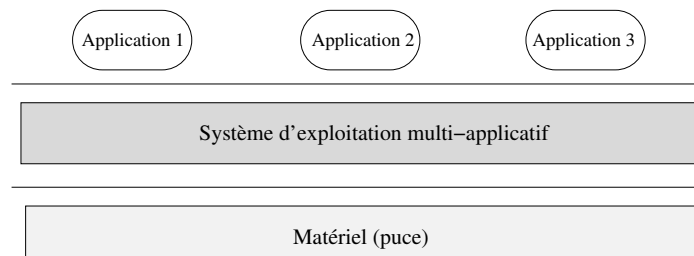


FIG. 1. Architecture d'une carte à puce multi-applicative. / Architecture of a multiapplication smart card.

Les mécanismes de machine virtuelle, d'environnement d'exécution virtuel et de bibliothèques universelles permet aux cartes des différents constructeurs implantant une même technologie d'être interopérables et donc de pouvoir accueillir une même application. Par ailleurs les cartes multi-applicatives offrent la dynamicité en permettant le chargement d'applications à n'importe quel moment du cycle de vie de la carte.

Nous allons maintenant présenter brièvement les différentes technologies de cartes multi-applicatives.

2.1 Java Card

En 1997, Java Card a été le premier standard multi-applicatif à apparaître. Cette technologie permet aux cartes à puce et à d'autres périphériques à mémoire limitée de faire fonctionner des applications écrites en langage Java, appelées *applets*. Selon Sun microsystems, elle définit une plate-forme sécurisée pour cartes à puce, portable et multi-application qui incorpore beaucoup des avantages du langage Java : programmation orientée objet, réutilisations des environnements de développement existants, langage fortement typé, plusieurs niveaux de contrôle d'accès aux méthodes et aux variables, portabilité (*Write Once, Run Anywhere*), etc. Cependant, cette technologie présente également quelques inconvénients, notamment au niveau de la sémantique du langage [16]. Lors de son conception, l'innovation majeure qu'apportait Java Card était la capacité de stockage et de gestion de multiples applications. Compte tenu des faibles capacités mémoire, pour introduire Java dans les cartes à puce la solution choisie fut de supporter seulement un sous-ensemble des caractéristiques du langage Java et de découper la machine virtuelle Java (JCVM, *Java Card Virtual Machine*) en deux parties. Une partie qui s'exécute en dehors de la carte (le convertisseur) et une partie qui s'exécute sur la carte (principalement l'interpréteur de *bytecodes*). Ainsi beaucoup de tâches ne sont plus réalisées à l'exécution mais sont déportées vers la partie de la machine virtuelle hors carte (e.g. le chargement de classes, la vérification de *bytecode*, la résolution de liens, l'optimisation).

Afin de palier aux problèmes de sécurité dû à l'absence de vérificateur embarqué, la technologie Java Card a ajouté à l'environnement d'exécution (JCRE, *Java Card Runtime Environment*) un mécanisme sécuritaire qui permet une séparation claire entre le système de la carte et les applications : le pare-feu. Il isole chaque applet des autres applets à l'intérieur de son propre espace appelé « contexte ». Ainsi, l'existence et les opérations d'une applet n'ont pas d'effet sur les autres applets présentes sur la carte. Ce pare-feu est renforcé par la JCVM puisque c'est elle qui exécute les *bytecodes*. Dans les situations où les applets ont besoin de partager des données (*i.e.* de collaborer) ou d'accéder à des services du JCRE, la machine virtuelle permet de telles possibilités à travers des mécanismes sécurisés de partage accessible via des méthodes APIs.

Une des raisons du succès de Java Card est la disponibilité gratuite du kit de développement de Sun microsystems et de ses spécifications.

2.2 MULTOS

Le standard MULTOS, soutenu par le consortium industriel MAOSCO, est le principal concurrent de la technologie Java Card. Comme pour elle, les concepteurs de MULTOS ont privilégié l'interopérabilité, la gestion d'applications multiples et la sécurité.

Pour le développement des applications, le développeur utilise :

- le langage optimisé pour les cartes à puce appelé MEL (Multos Executable Language), basé sur les P-codes Pascal et la machine virtuelle décrite par David A. WATT [17] ;
- des spécifications bien définies pour les APIs MULTOS. En d'autres termes, les fabricants de cartes MULTOS ne sont pas libres d'ajouter leur propres extensions pour des raisons d'interopérabilité. Ainsi, tous proposent les mêmes services avec des interfaces identiques et homogènes. Sur ce point, MULTOS diffère beaucoup de Java Card puisque par exemple, avant Java Card 2.2, les quelques constructeurs qui proposaient des mécanismes de ramasse-miettes utilisaient des interfaces différentes. Ceci avait pour conséquence des problèmes d'interopérabilité pour les applications Java Card destinées à être installées sur plusieurs plates-formes concurrentes.

Les applications pour MULTOS sont en général programmées soit en C soit en Java, mais, comme il existe une multitude de compilateurs, on peut aussi les programmer en Basic ou en Modula-2. Il faut noter que MULTOS est la seule technologie de cartes multi-applicatives qui permet d'inclure du code assembleur dans une application programmée dans un langage de haut niveau. Pour résumer, il existe plusieurs environnements de développement pour la plate-forme MULTOS, mais tous sont payants.

À l'installation d'une nouvelle application, la carte à puce MULTOS vérifie la validité de l'application qui a été envoyée, alloue au programme un espace mémoire protégé grâce au pare-feu. Chaque nouveau service ou application reste rigoureusement séparé des autres programmes déjà sur la carte afin qu'aucun d'eux ne puisse interférer avec un autre programme en cas de dysfonctionnement. Avec la technologie MULTOS, les applications peuvent communiquer entre elles et s'échanger des données au moyen des APDUs simplifiant considérablement l'écriture d'une application par rapport à Java Card. C'est un système simple et efficace qui permet la migration d'une application du terminal vers la carte sans nécessiter beaucoup de développement puisque le protocole de communication reste le même.

2.3 Windows for Smart Cards de Microsoft

La WfSC, proposée par Microsoft, est une carte multi-applicative orientée authentification compatible ISO 7816. Elle a été développée en 1998 (par Scott B. Guthery [18]) et elle n'est aujourd'hui quasiment plus utilisée.

La philosophie de cette carte est d'adapter le niveau de compétence et le modèle de programmation de la carte à puce pour la rendre accessible aux développeurs d'applications plutôt que de les forcer à acquérir de nouvelles compétences et à apprendre de nouveaux modèles de programmation pour utiliser la carte à puce. Évidemment, il y a eu tout de même des petites adaptations à cause des ressources limitées de la carte.

Cette carte embarque un vrai système de fichiers FAT avec des noms de fichiers représentés en Unicode. Le contrôle d'accès proposé est très fin puisqu'elle utilise les ACLs (*i.e.* Access Control List) présents sur beaucoup de systèmes Unix mais aussi sous Windows NT. La

machine virtuelle embarquée est une version compacte de l'Intel 8048 et l'API proposée est une version considérablement réduite de l'API Windows (Win32). Cette carte peut être utilisée selon deux modes, un mode système de fichiers tout simple avec un accès par le jeu de commandes standard ISO7816 ou un mode multi-applicatif qui est celui qui nous intéresse ici.

Un programme compilé pour WfSC consiste en deux fichiers, un fichier de *bytecodes* et un fichier pour les données. Cette séparation du code et des données en plus d'être une pratique courante de bonne programmation permet de mettre à jour les applications facilement tout en conservant leurs données. Ceci n'est, par exemple, pas aussi évident avec Java Card. Sur cette plate-forme le partage de données se fait avec les ACL au travers du système de fichiers.

Les applications se programme en Visual Basic ou Visual C++. Toutefois Microsoft ayant abandonné le support de WfSC dans Visual Studio et il faut utiliser d'autres environnements de développement commerciaux.

2.4 The ZeitControl BasicCard

Même si les BasicCard existent depuis 1996 c'est seulement en avril 2004 que ZeitControl a sorti sa première carte multi-applicative la *MultiApplication BasicCard ZC6.5* [15].

À l'instar des cartes multi-applicatives déjà présentées, les ZeitControl BasicCard possèdent une machine virtuelle, mais celle-ci se différencie des autres en étant la seule à supporter les nombres flottants.

Cette carte se programme en ZeitControl Basic qui contient la plupart des constructions habituelles du langage Basic telles que les chaînes et les fonctions pour les manipuler, les tableaux, les données définies par l'utilisateur, etc. Elle possède aussi un système de fichiers un peu similaire à celui de la FAT et propose une API Basic classique pour son utilisation. De façon similaire à WfSC, le partage de données se fait au travers du système de fichiers avec un contrôle d'accès par groupe d'applications. Les applications ne respectant pas les règles associées aux fichiers ne peuvent accéder qu'à leurs propres données.

L'environnement de développement proposé par ZeitControl est complet, bien pensé et disponible gratuitement sur leur site web. Enfin, un énorme avantage de ces cartes est leur très faible coût à l'unité.

2.5 Smartcard.NET

Cette plate-forme ne marque pas le retour de Microsoft dans le monde de la carte. C'est Hive-Minded, une pousse issue de Berkeley, qui a lancé Smartcard.NET [13,14] avec l'idée que si quelqu'un peut écrire un programme pour Windows, il peut le faire pour une carte à puce.

Si l'architecture proposée est très proche de celle de Java Card, la philosophie est assez différente. Par exemple, l'API Java Card n'est pas un sous-ensemble de l'API Java mais quelque chose de vraiment distinct. Au contraire, l'objectif des concepteurs de Smartcard.NET a été de fournir un vrai sous-ensemble de l'API .NET pour les cartes à puce. Ainsi, leur machine virtuelle supporte tous les types exceptés les nombres flottants, la vérification de code, les entiers sur 64 bits et le *Remoting* (*i.e.* les RPC dans .NET). Leur API inclut tous les types et toutes les méthodes requis par le langage pour supporter .NET (y compris les chaînes).

Elle gère également les entrées/sorties, la cryptographie, les synchronisations entre *threads*, les collections, etc.

Un des gros avantages de cette carte est que .NET supporte beaucoup de langages de programmation. Par conséquent, il est possible d'écrire des applications en C#, VB.NET, Jscript.NET, J#, etc.

Bien que très récent, Smartcard.NET pourrait très rapidement rejoindre Java Card et MULTOS parmi les poids-lourds de la multi-application.

3 Techniques d'assurance pour la sécurité des cartes multi-applicatives

Les techniques d'assurances que nous entendons présenter dans cette section sont celles destinées à donner confiance aux clients des fabricants de cartes multi-applicatives (*i.e.* des banques, des organismes de défense, etc.). Elles ont pour objectif « d'assurer » que la sécurité est bien présente et de minimiser les risques pour la carte.

Les trois grandes techniques d'assurance actuelles pour les cartes multi-applicatives sont :

- l'évaluation sécuritaire et la certification ;
- l'utilisation de GlobalPlatform ;
- les méthodes formelles.

Ces techniques d'assurance ne sont pas exclusives mais complémentaires. C'est donc en les associant qu'on arrive à obtenir une bonne assurance du niveau de sécurité d'une carte multi-applicative. Par conséquent, nous allons exposer chacune de ces méthodes et présenter comment elles permettent de renforcer le niveau d'assurance de la sécurité d'une carte multi-applicative.

3.1 L'évaluation sécuritaire et la certification

Le processus de certification consiste en une procédure complexe mettant en jeu plusieurs acteurs et dont l'action finale est d'émettre un certificat pour un produit ou un système issus des Technologies de l'Information (TI). La délivrance de ce certificat a lieu à l'issue de l'évaluation sécuritaire du produit réalisée par un tiers indépendant. En conséquence, le certificat d'un produit TI a pour vocation d'établir le niveau de confiance minimum qu'un utilisateur final pourra avoir dans la sécurité du produit.

Comme illustré Fig. 2, le processus de certification fait intervenir trois tiers indépendants :

- le commanditaire de l'évaluation qui souhaite obtenir un certificat pour son produit ou son système TI ;
- le centre d'évaluation qui conduit les tâches d'évaluation de la sécurité du produit en fonction de la méthodologie choisie par le commanditaire (e.g. les Critères Communs [19] ou ITSEC [20]), du niveau d'assurance choisi par le commanditaire, des éléments de preuve apportés par le commanditaire et de sa propre expertise technique ;
- l'organisme de certification qui revoit la pertinence technique des rapports émis par le centre d'évaluation et qui au vu du Rapport Technique d'Évaluation (RTE) décide ou non de la délivrance du certificat.

On notera que pour garantir le bon déroulement de l'évaluation, à chaque étape des éléments de preuve sont produits à partir de l'analyse des fournitures provenant du commanditaire (produits, documentation de conception, code des applications, preuves formelles de

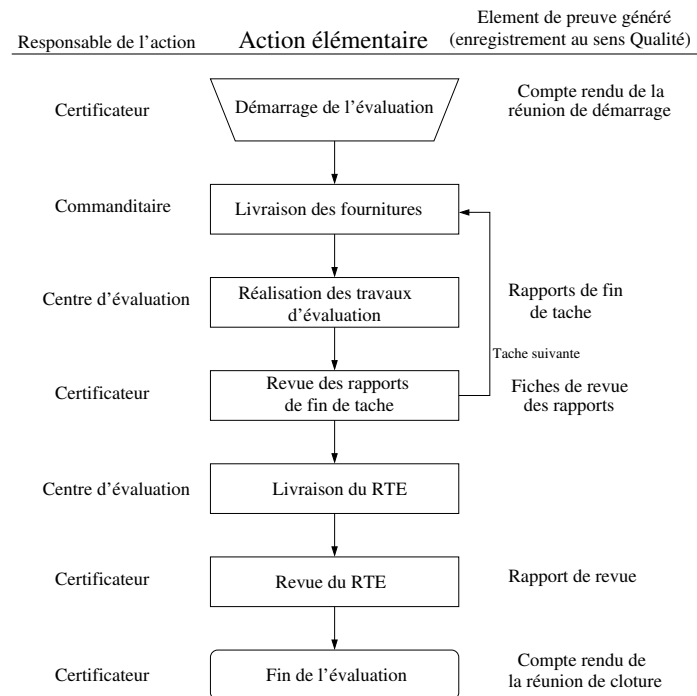


FIG. 2. Déroulement de l'évaluation sécuritaire. / Course of a security evaluation.

conception, etc.) et de l'expertise technique du centre d'évaluation. C'est la présence d'un tiers indépendant (*i.e.* l'organisme de certification), chargé de vérifier l'exhaustivité et la qualité des travaux menés par le centre d'évaluation grâce aux éléments de preuve, qui garantit le niveau de confiance que l'on peut avoir dans le produit TI. En effet, l'activité d'évaluation de la sécurité est une activité commerciale et à ce titre c'est le commanditaire qui choisit le centre d'évaluation qui va conduire l'évaluation de son produit. Dès lors, s'il n'y avait pas le contrôle extérieur de l'organisme certificateur, on pourrait tout à fait imaginer qu'un commanditaire « achète » son certificat avec la complaisance du centre d'évaluation désireux de ne pas froisser son client. Heureusement, ceci n'est pas possible au sein des schémas de certification tripartite mis en place dans les différents pays. Afin de développer la certification des produits TI mais aussi de renforcer la confiance dans leur système de certification, les gestionnaires de schémas de certification de plusieurs pays ont passé des accords de reconnaissance mutuelle des certificats émis par eux au niveau international.

Les avantages de la certification sont donc de garantir un certain niveau de confiance dans la sécurité d'un produit TI (confidentialité, intégrité, disponibilité). Ainsi, les utilisateurs finaux peuvent comparer les différents produits sur le marché de façon objective. Cela permet également aux commanditaires de prouver leur compétence, d'en faire un argument marketing et ainsi d'étendre leurs marchés. Enfin, cela permet aux organismes de certification – qui sont pratiquement toujours des organismes gouvernementaux – d'être sûrs que les produits sur le marché sont sécurisés et que leur pays ne court pas de risque majeur lors d'une attaque contre ses systèmes TI.

Si les processus de l'évaluation sécuritaire et de la certification ne sont pas spécifiques aux cartes multi-applicatives ni même d'ailleurs aux cartes à puce, ils contribuent de façon considérable, aux yeux des clients industriels (les banques, etc.), à fournir l'assurance d'un fort niveau de sécurité dans les cartes évaluées.

3.2 GlobalPlatform

GlobalPlatform [8] est un consortium d'industriels qui propose une spécification pour gérer les cartes de façon indépendante du matériel, des vendeurs et des applications. L'objectif de ces spécification est de résoudre les problèmes que l'on rencontre lorsqu'on veut utiliser des cartes multi-applicatives sur une grande échelle.

La position de GlobalPlatform a été de considérer, à juste titre, que l'on ne pouvait pas assurer la sécurité dans une carte multi-applicative tant qu'un vérifieur embarqué ne serait pas présent pour vérifier l'innocuité d'un code chargé et qu'à ce titre, il fallait interdire le chargement d'applications, sauf sous certaines conditions. Ainsi, les spécifications ne traitent pas directement des problèmes liés aux attaques internes et les problématiques présentées ci-dessous concernent seulement la gestion des applications au sein de la carte et leurs interactions vis-à-vis de l'extérieur. Par conséquent, nous aborderons le problème des attaques internes de façon spécifique dans la section 4.1 et nous présenterons également de nouveaux problèmes dans la section 4.2.

Afin de pouvoir utiliser industriellement les possibilités multi-applicatives des nouvelles cartes, GlobalPlatform a défini sous quelles conditions il était possible de charger et d'installer une application sur une telle carte. En d'autres termes, les spécifications déterminent qui est autorisé à charger des applications et dans quel contexte (*i.e.* quand et comment? – dans quels états du cycle de vie de telles opérations sont possibles?, etc.). Par exemple, selon l'environnement physique dans lequel se trouve la carte au moment de l'opération et le niveau de confidentialité du code de l'application, il faudra utiliser des procédures de chargement sécurisé.

Pour renforcer la politique de sécurité définie ci-dessus, mais aussi pour permettre une plus grande granularité, GlobalPlatform propose un mécanisme de signature des applications qui est garant que l'application est de confiance et que, par conséquent, elle ne devrait pas tenter de nuire aux autres applications présentes sur la carte si on la charge. En effet, si l'application a été signée c'est par un partenaire commercial de l'émetteur de carte ou par lui-même et on peut donc raisonnablement supposer qu'il aura fait les vérifications qui s'imposent pour s'assurer que la carte et les applications embarquées ne risquent rien (soit par l'utilisation d'un programme automatique de vérification du code de l'application, soit par un audit de son code).

GlobalPlatform définit également la manière dont une application peut accéder à des services globaux de la carte au travers de l'API GlobalPlatform (e.g. vérification de l'identité du porteur de carte, modification des octets d'historiques de l'ATR, etc.).

Enfin, GlobalPlatform met en place des procédures standards pour l'authentification mutuelle des applications embarquées sur la carte et celles hors carte. En effet, dans le cadre de l'utilisation de cartes multi-applicatives, il est nécessaire qu'une application client hors-carte puisse être sûre de dialoguer avec la bonne application sur la carte et vice-versa (*i.e.* il

faut des procédures pour assurer que chacun est celui qu'il prétend être). C'est un problème qui n'existait pas dans le cadre des cartes mono-applicatives puisque lorsqu'une application hors-carte dialoguait avec une carte, c'était forcément avec la bonne application.

Grâce aux nombreuses fonctionnalités qu'il propose le standard GlobalPlatform a réussi à lever le verrou psychologique quant à une utilisation industrielle des cartes multi-applicatives. En revanche le verrou technologique existe toujours puisque pour pouvoir charger du code sur sa propre carte multi-applicative il faut être un partenaire commercial de l'émetteur de carte.

Ainsi, tant que les vérificateurs de code ne seront pas embarqués sur les cartes ou tant qu'il n'aura pas été prouvé à 100% que les attaques ne sont pas possibles sur telle ou telle technologie de cartes multi-applicatives, il y a fort à parier que les industriels choisiront de ne pas ouvrir davantage leur cartes.

Aujourd'hui, l'infrastructure GlobalPlatform possède de nombreux atouts qui lui ont permis de prouver son efficacité puisqu'elle a déjà été déployée dans le secteur des cartes SIM et des cartes bancaires.

3.3 Les méthodes formelles

Les différents travaux de recherches exposés ici sont classés de façon thématique et se présentent sous une forme assez synthétique. Ils se basent presque tous sur l'utilisation de méthodes formelles et concernent principalement Java Card. La prédominance de cette technologie, dans les recherches académiques est liée à plusieurs facteurs :

- son succès dans l'industrie ;
- ses spécifications ouvertes et disponibles facilement ;
- aux travaux déjà menés sur Java et qui peuvent être facilement étendus à un tel environnement.

Les cartes à puce se prêtent idéalement aux travaux de formalisation car la taille du système est assez réduite. Ceci permet de mettre en œuvre des outils preuve, des générateurs de tests, etc.

À l'inverse, les résultats de ces recherches sont fondamentaux pour assurer et vérifier la sécurité du système d'une carte à puce et constituent donc une technique d'assurance incontournable pour apprécier le niveau de sécurité des cartes multi-applicatives développées.

Les travaux autour des APIs Java Card concernent pour l'essentiel leur modélisation formelle ou semi-formelle [21,22,23,24]. Ils utilisent des langages tels que JML [25,21,26] ou ESC/Java [27,26] pour traduire les spécifications informelles, puis ils vérifient la correction d'une implantation à l'aide d'un outil de preuve (e.g. PVS [28]). Bien évidemment, cette vérification n'est possible que grâce à une traduction préalable avec des outils comme LOOP [29] des spécifications JML et du code Java en un langage compréhensible par un outil de preuve. Dans ces travaux différentes propriétés commencent à être introduites pour exprimer des contraintes temporelles [30] et matérielles [31]. La prise en compte de aspects matérielles des cartes dans les modélisations nous semble essentielle pour obtenir des résultats fidèles à la réalité. D'ailleurs, on notera que dans les modélisations des APIs – que cela soit en JML ou ESC/Java –, la nature transiente ou persistante des objets n'est pas prise en compte [16].

Pour la JCVM, les travaux consistent principalement à en établir une description formelle, et à modéliser sa sémantique opérationnelle dans divers langages de preuve (*i.e.* Coq,

Isabelle/HOL) pour vérifier la correction de certaines propriétés [32,33]. Par ailleurs, afin de faciliter les différentes étapes de description, de modélisation et de vérification, un ensemble d'outils, appelé Jakarta, a été développé [34].

Concernant le pare-feu, les travaux portent essentiellement sur l'analyse statique du code pour vérifier si le partage d'objets est bien fait, mal fait, ou indéterminé [35]. Il existe aussi des travaux établissant une spécification formelle [36] et définissant une sémantique opérationnelle pour le pare-feu [37]. Enfin, le reste des travaux se concentre sur le développement de kits de programmation assurant une bonne gestion par les applications du partage d'objets [38].

Dans le domaine des transactions, il existe des travaux concernant le traitement à apporter aux d'objets créés lors des transactions [39]. D'autres travaux présentent une façon d'introduire les notions d'arrachage de carte et de transaction dans des spécifications JML [31].

Pour le vérifieur, nous évoquerons les travaux dans la section 4. Si dans ce domaine des travaux ont conclu à l'équivalence de la machine virtuelle défensive et de la machine virtuelle offensive à laquelle on adjoint un vérifieur [40], d'autres ont permis de construire à partir d'une machine virtuelle défensive, une machine virtuelle offensive et le vérifieur associé.

Des travaux ont montré à l'aide de Coq et d'une sémantique opérationnelle abstraite définie pour le fichier CAP et les fichiers `class` correspondants, qu'il y avait une relation logique permettant de prouver la correction des optimisations faites lors de la construction d'un fichier CAP à partir des `class` initiales [41,42].

Dans le domaine de la multi-application, nous présenterons les travaux dans la section 4.3.

Dans le domaine de la vérification d'*applets*, on trouve des méthodologies [43] et des outils (e.g. JIVE [44]) permettant de vérifier qu'une *applet* respecte une spécification informelle en utilisant une fois encore des langages comme JML. Par ailleurs, des travaux et des outils (e.g. JCAVE [45]) sont également disponibles afin d'établir les interactions entre des *applets* au niveau *bytecode* en utilisant pour modéliser le comportement des *applets* une logique temporelle linéaire [46]. Enfin, d'autres travaux proposent de réaliser des tests de conformité en générant un jeu de tests à partir d'un modèle UML des *applets* et de leurs communications avec les autres composants grâce aux outils UMLAUT et TGV [47]. Évidemment, il est également possible de tester une *applet* dans un générateur de preuve en convertissant les fichiers CAPs à l'aide d'outils de Jakarta.

Très récemment, une implantation de référence définissant de façon formelle les spécifications GlobalPlatform v2.1.1 est parue [48]. Elle utilise la logique formelle du langage B.

S'il est possible de citer encore beaucoup d'autres travaux, nous avons décrit ce que nous pensons être les principales recherches qui ont été menées sur les technologies de cartes multi-applicatives lors de ces dernières années.

4 Les cartes multi-applicatives ouvertes

Jusqu'à aujourd'hui aucune technologie de cartes multi-applicatives n'a été prouvée complètement sécurisée à l'aide de méthodes formelles. Par conséquent il ne serait pas sûr d'accepter des applications non certifiées qu'un attaquant pourrait charger pour tenter d'accéder aux biens de la plate-forme ou à ceux des applications déjà installées. Pour éviter ces problèmes les constructeurs de cartes implantent des standards tels que GlobalPlatform [8] afin de spécifier comment charger, installer et gérer de manière sécurisée plusieurs applications sur

la même carte. En effet, nous l'avons vu, ce standard permet de subordonner le chargement d'une application à la réalisation de certaines conditions. Par exemple, la carte peut accepter une application si elle est signée par une entité de confiance (e.g. l'émetteur de carte.). Pour cela, une fois l'application chargée, la carte vérifiera sa signature et l'acceptera ou la rejettera. L'inconvénient majeur de cette solution est le modèle centralisé dû au tiers de confiance signant les applications, qui réduit la flexibilité.

Toutefois, bien que les concepteurs des technologies multi-applicatives pensaient impossible d'embarquer sur les cartes un vérifieur de *bytecodes* à cause de leurs faibles ressources, plusieurs propositions de vérifieurs embarqués ont été avancées [49,50,51,52,53,54,9]. Parmi les différentes solutions proposées, trois (*i.e.* la machine virtuelle défensive, le vérifieur basé sur la transformation de code et le vérifieur autonome) permettent de se passer de l'étape de signature de l'application sans pour autant mettre en danger la sécurité de la carte. L'implantation de ces solutions permettrait donc à n'importe qui de charger librement son application sur sa carte – *i.e.* sans authentification préalable. Toute tentative de chargement d'une application malicieuse sera rejetée soit au chargement par le vérifieur autonome, soit à l'exécution par la machine virtuelle défensive. On notera cependant que le vérifieur basé sur la transformation de code [51,52] demande l'utilisation d'un programme hors carte pour normaliser l'application alors que les deux autres (*i.e.* la machine virtuelle défensive [53] et le vérifieur autonome [9]) sont réellement autonomes. Nous définissons donc **les cartes multi-applicatives ouvertes** comme les cartes utilisant une de ces deux solutions autonomes. Les deux sont équivalentes [54] mais la machine virtuelle défensive fait les vérifications dynamiquement à chaque *bytecode* exécuté alors que le vérifieur autonome fait une vérification statique seulement une seule fois au chargement. Pour des raisons de performance, ce dernier est en effet associé avec une machine virtuelle offensive qui ne fait donc plus de vérification sur les *bytecodes* à l'exécution.

Dans la suite, nous présenterons les attaques internes existantes spécifiques aux cartes multi-applicatives et celles, nouvelles, que nous proposons dans le cadre des cartes à puce multi-applicatives ouvertes [16,55].

4.1 Les attaques internes

De façon évidente, un des principaux problèmes des cartes multi-applicatives ouvertes est la possibilité de créer des attaques internes grâce à la possibilité de charger des applications contenant du code malicieux. De telles applications pourraient tenter :

- d'identifier les services présents sur la carte et de déduire le comportement probable d'une *applet* officielle¹ (en effet elle ne peut utiliser que les services présents sur la carte ce qui réduit le champ d'investigation de l'attaquant) ;
- de collecter des informations sur la carte pour préparer des attaques matérielles et logicielles ;
- d'attaquer la machine virtuelle et le pare-feu.

Dans la suite, le propos sera illustré au moyen d'exemples de code en Java Card.

¹ une *applet* officielle est une *applet* installée par un organisme officiel, e.g. une *applet* bancaire.

L'identification de service

Le listing 1.1 présente une application qui essaye d'utiliser tous les services cryptographiques définis dans les spécifications Java Card [11] afin de déterminer s'ils sont disponibles ou pas sur la carte.

Listing 1.1. La recherche des services cryptographiques. / Discovering cryptographic services.

```
public class TestCryptoAlgo extends Applet {
    ...
    public void process(APDU apdu) throws ISOException {
        byte[] buffer = apdu.getBuffer();
        // Essaye d'obtenir une instance de tous les algorithmes disponibles dans les spécifications.
        switch ( buffer[ISO7816.OFFSET_INS] ) {
            case 0x02: isValidAlgorithm( Cipher.ALG_DES_CBC_NOPAD );
            case 0x04: isValidAlgorithm( Cipher.ALG_DES_ECB_NOPAD );
            case 0x06: isValidAlgorithm( Cipher.ALG_RSA_PKCS1 );
            case ...
        }
    }

    public void isValidAlgorithm(byte bAlgo) throws ISOException {
        try {
            // Essaye de créer une instance d'un algorithme de type bAlgo.
            Cipher cipherInst = Cipher.getInstance(bAlgo, true);
        } catch ( CryptoException e ) {
            // Si le type n'est pas implémenté, le Status Word = 0x6FFF est envoyé sur la ligne d'I/O
            // sinon le SW = 0x9000 signifiant réussi est envoyé par la carte elle-même à la fin.
            if ( e.getReason() == CryptoException.NO_SUCH_ALGORITHM )
                ISOException.throwIt((short) 0x6FFF);
        }
    }
    ...
}
```

La collecte d'information

Le listing 1.2 montre comment il est possible de détecter toutes les applications présentes sur une Java Card. Si elles proposent des services, il montre également comment essayer d'obtenir une interface `Shareable` pour les utiliser. Ce code présuppose que les *applets* cibles ne réalisent qu'une authentification basée sur la valeur de `parameter` passée à la méthode `getAppletShareableInterfaceObject(AID serverAID, byte parameter)`. C'est une hypothèse forte et peu réaliste. En revanche, si les *applets* cibles utilisent une authentification plus réaliste mais basée seulement sur l'AID de l'*applet* client (*i.e.* `ScannerApplet` ici) il est encore possible sur certaines implantations de Java Card d'obtenir un AID valide. Cette attaque connue sous le nom de « *AID Impersonation* » [56] est rendue possible à cause de certaines contradictions sur les règles de nommage des *applets* dans les standards Java Card et GlobalPlatform. En effet, GlobalPlatform permet d'installer une *applet* avec l'AID que l'on désire alors que Java Card restreint cela à un AID basé sur le RID du paquetage. Donc, selon les choix d'implantation, il sera possible de mener à bien cette attaque ou non.

Il est également possible d'ajouter un glitch (voir la méthode décrite dans la section 4.2) juste avant d'accéder aux données des *applets* cibles afin de synchroniser une attaque physique (e.g. injection de fautes [57,58,59]) dans l'espoir de passer outre les vérifications du pare-feu grâce aux perturbations créées sur le composant matériel.

Listing 1.2. Le code pour scanner toutes les applications et essayer de récupérer une interface shareable. / To scan all the applications and to try to get a shareable interface.

```
public class ScannerApplet extends Applet {
    ...
    public void process(APDU apdu) throws IOException {
        byte[] tab = new byte[16];
        ...
        // Mettre ici le code pour tester toutes les valeurs possibles pour tab
        ...
        for (byte i=5; i <= 16; i++) {
            AID aid = JCSystem.lookup(tab, (short) 0, i);
            if (aid != null) tryToGetServices(aid);
        }
    }

    void tryToGetServices(AID aid) {
        for (byte i=0; i <= 255; i++) {
            Shareable sio = JCSystem.getAppletShareableInterfaceObject(aid, i);
            if (sio != null) callMethodOfSIO(sio);
        }
    }

    void callMethodOfSIO(Shareable sio) {
        // Mettre ici le code pour appeler les méthodes sur cette interface
    }
    ...
}
```

Les attaques contre la machine virtuelle, le pare-feu, les APIs

Évidemment, il existe beaucoup d'attaques contre la VM et le pare-feu, mais nous ne les décrivons pas en détails car beaucoup d'autres travaux les ont déjà évoquées. Par exemple, deux attaques contre le mécanisme de pare-feu (*AID impersonation* et le *cast* illégal de référence pour avoir accès à toutes les méthodes d'interface d'une classe) sont décrites dans [56]. Il y a également des attaques issues de problèmes dans les spécifications comme celles présentées dans [60]. Des attaques contre la VM existent aussi et elles peuvent être dues à une mauvaise spécification ou à une mauvaise implantation.

Par exemple, le programme du listing 1.3 qui fabrique un pointeur devrait être rejeté par une carte multi-applicative ouverte soit au chargement pour celles utilisant le vérifieur autonome soit à l'exécution pour celles utilisant une machine virtuelle défensive. Malgré tout, un attaquant peut toujours essayer de charger de tels codes malicieux afin de déterminer s'il y a des problèmes d'implantation ou non.

Le morceau de code du listing 1.3 (présenté dans un langage intermédiaire) parcourt toute la table d'allocation et essaie de lire un à un les octets de l'objet en le considérant comme un tableau. Il utilise l'hypothèse forte que la pile n'est pas typée (e.g. c'est le cas d'une machine virtuelle offensive ou d'une mauvaise implantation d'une machine virtuelle défensive) afin de faire de l'arithmétique sur les références.

On notera que si la référence est représentée par une adresse mémoire (*i.e.* l'équivalent d'un pointeur) dans l'implantation de la machine virtuelle, alors ce code parcourra toute la mémoire.

4.2 De nouvelles classes d'attaques

Une fois que les services offerts par une carte ouverte ont été identifiés par l'une des méthodes exposées précédemment, il peut être intéressant d'observer leur signature physique issus des canaux cachés : consommation en courant, émission électromagnétique ou durée du

Listing 1.3. Le code pour balayer toutes les références. / To scan all the references

```
sconst_0 // From reference 0
:loop
dup
sconst_0
baload // Essaye d'obtenir un octet et le pose sur la pile (mais on peut imaginer faire d'autres opérations)
invoke do_something // Méthode qui retire l'octet lu et ne retourne rien
sconst_1
sadd // Incrémente la valeur de la référence par pas de 1
goto loop

void do_something(byte value@address) {
// Récupère une opérande de type byte sur la pile (i.e. lit l'octet) et la sauvegarde quelque part
}
```

temps d'exécution. Par exemple, on peut utiliser un service déterminé comme présent sur la carte afin d'essayer de rechercher les caractéristiques physiques des signaux émis durant son exécution (e.g. durée dans le temps, localisation spatiale des meilleures émissions électromagnétiques, consommation en courant, etc.). Cette technique de caractérisation pourra se focaliser sur l'utilisation d'un service complet mais également sur des opérations élémentaires, e.g. l'interprétation d'un seul *bytecode* ou d'une séquence de *bytecodes*.

Les méthodes d'aide à l'identification physique

Nous avons décrit dans [16,55] les trois méthodes permettant de localiser un motif choisi dans la trace d'exécution d'une application chargée sur une carte multi-applicative.

La méthode la plus simple, se fait par l'utilisation de *glitches* sur le canal d'entrée/sortie de la carte et consiste à entourer le motif à observer par des événements visibles pour un observateur situé à l'extérieur de la carte. La Fig. 3 montre la trace de l'exécution normale d'une *applet* contenant le motif que l'on veut observer. On peut constater dans un tel cas la difficulté d'isoler le motif à observer de tous ceux composant la trace. En revanche, avec une

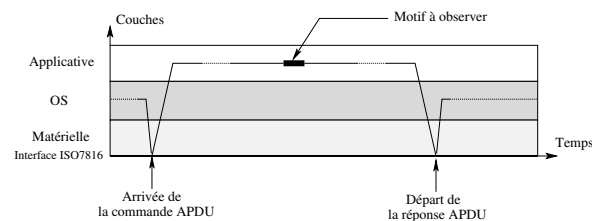


FIG. 3. Trace d'une exécution normale vue au travers des différentes couches. / Trace of a normal execution in the layers.

l'exécution glitchée, *i.e.* une exécution où l'on a entouré le motif à observer avec des octets de synchronisation, il sera alors facile de retrouver le motif dans la trace (*cf.* Fig. 4). Tous les détails sur les méthodes d'aide à l'identification physique ainsi que les contre-mesures que nous avons proposés sont décrites dans [16,55].

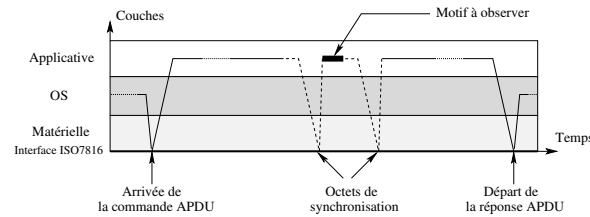


FIG. 4. Trace d'une exécution *glitchée* vue au travers des différentes couches. / Trace of a glitched execution in the layers.

Les applications des méthodes d'aide à l'identification physique

On peut imaginer au moins deux applications possibles des méthodes d'aide à l'identification physique expliquées précédemment :

- les attaques par couplage qui permettent de faire de la rétro-conception de code d'une *applet* officielle ;
- les attaques physiques qui permettent de déterminer rapidement la faisabilité d'une attaque contre une *applet* officielle.

Les attaques par couplage : Une telle attaque consiste à coupler le signal identifié pour un motif dans une *applet* malicieuse avec le signal d'une *applet* officielle afin d'en faire la rétro-conception. La première étape nécessaire pour mettre en place cette attaque est de construire un dictionnaire qui fait correspondre les motifs d'un signal avec les *bytecodes*. La seconde étape est d'identifier les motifs du dictionnaire dans une trace d'exécution d'une *applet* officielle pour connaître les opérations qu'elle réalise.

C'est évidemment une tâche très difficile mais des recherches sont en cours pour permettre la reconnaissance automatique de motifs dans un signal [61]. Il est important de remarquer que si cette attaque peut mettre en péril la confidentialité du code de l'*applet* officielle (*i.e.* le code peut être connu), elle n'assure pas qu'une attaque sera possible contre l'exécution de ce code.

Cependant, dans les évaluations Critères Communs [19], le code d'une *applet* officielle est souvent un bien à protéger et une telle attaque sera donc un problème à prendre en compte. De plus, la connaissance du code permet de mener des analyses détaillées pour rechercher d'éventuelles vulnérabilités afin de déterminer si une attaque est possible et à quel moment dans l'exécution du code. Enfin, il est aussi possible de coupler la connaissance du code avec les attaques que nous décrirons ci-dessous.

Les attaques physiques : Grâce aux méthodes d'identifications que nous avons présentées, un attaquant peut mettre en place des attaques physiques [57,58,59] sur le motif identifié dans sa propre *applet*. En effet, il peut se placer dans les meilleures conditions pour attaquer facilement et rapidement l'implantation d'une carte (e.g. essayer de passer outre les conditions d'accès ou perturber un algorithme cryptographique) évitant ainsi de mener beaucoup d'essais inutiles et donc par la même, gagner beaucoup de temps.

Si son attaque contre sa propre *applet* réussit, il peut alors ensuite attaquer l'*applet* officielle ou même la plate-forme.

Par exemple, s'il connaît une attaque contre un algorithme cryptographique utilisé par une *applet* officielle, il pourra attaquer directement cet algorithme en l'appelant depuis sa propre *applet*. Ainsi, en entourant l'appel de *glitches*, il pourra tester rapidement et efficacement la résistance de son implantation.

4.3 Travaux connexes

Des travaux visant à détailler les problèmes des cartes à puce multi-applicatives ont également été menés en 1999 par les équipes de Gemplus [62,63,64]. Ils soulevaient plusieurs problèmes relatifs au chargement, à la machine virtuelle, au partage d'objets et au flux d'information entre *applets* mais aucun n'identifiait les attaques physiques et les attaques par couplage que nous avons présentées en section 4.2.

Plus récemment, d'autres travaux [65,66] sur les cartes multi-applicatives n'ont pas non plus identifié les problèmes que nous décrivons. Les travaux de Pierre Bieber *et al.* [65] proposent de permettre à l'émetteur de cartes de vérifier qu'une nouvelle *applet* interagira de façon sécurisée avec les autres *applets* déjà présentes sur la carte. Ceux de Gerhard Schellhorn *et al.* [66] présentent un modèle de sécurité générique pour les systèmes d'exploitation de cartes multi-applicatives qui formalise les principaux aspects de sécurité : confidentialité, intégrité, communication sécurisée inter-applications et chargement sécurisé de nouvelles applications.

Par ailleurs, nous n'avons trouvé aucun travaux connexes basés sur les méthodes d'identification physique pour caractériser une plate-forme et il nous a semblé judicieux de les présenter dans le cadre des cartes multi-applicatives ouvertes.

5 Conclusion

Dans cet article nous avons présenté une synthèse des différentes techniques d'assurance garantissant le haut niveau de sécurité des cartes multi-applicatives aujourd'hui présentes sur le marché. Utilisées de manière concomitante, ces méthodes procurent aux clients (banques, etc.) la confiance qu'ils recherchent pour investir dans les cartes à base de technologie multi-applicative. Ainsi, ces techniques d'assurance permettent de conserver un minimum de contrôle sur la carte (notamment via les mécanismes mis en œuvre par GlobalPlatform après la phase d'émission de la carte), tout en fournissant le dynamisme nécessaire au développement de nouveaux services (e.g. application de fidélité). Toutefois, pour atteindre une flexibilité totale, l'issue inéluctable sera de fournir des cartes multi-applicatives ouvertes. Par conséquent les méthodes formelles devraient maintenant prendre le pas sur le standard GlobalPlatform pour assurer la confiance des décideurs et promouvoir l'avenir et la sécurité des cartes multi-applicatives ouvertes. Dans ce cadre, les problèmes que nous avons évoqués lors de la section 4.2 devront être pris en compte (car inhérents à toute carte ouverte) afin de garantir la sécurité de ces nouvelles cartes.

Références

1. Kocher, P. : Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In : Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1996) 104–113

2. Kocher, P., Jaffe, J., Jun, B. : Differential Power Analysis. In : Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1999) 388–397
3. Coron, J.S., Kocher, P., Naccache, D. : Statistics and Secret Leakage. In : Proceedings of Financial Cryptography (FC2000). Volume 1962 of Lecture Notes in Computer Science., Springer-Verlag (2001) 157–173
4. Quisquater, J.J., Samyde, D. : ElectroMagnetic Analysis (EMA) : Measures and Countermeasures for Smart Cards. In : Proceedings of E-smart 2001. Volume 2140 of Lecture Notes in Computer Science., Springer-Verlag (2001) 200–210
5. Gandol, K., Mourtel, C., Olivier, F. : ElectroMagnetic Analysis : Concrete Results. In : Proceedings of CHES'2001. Volume 2162 of Lecture Notes in Computer Science., Springer-Verlag (2001) 251–261
6. Kömmerling, O., Kuhn, M.G. : Design Principles for Tamper-Resistant Smartcard Processors. In : Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 9–20
7. Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G. : Improving Smart Card Security using Self-timed Circuits. In : Proceedings of ASYNC'02. (2002) 211–218
8. GlobalPlatform : GlobalPlatform. (<http://www.globalplatform.org/>)
9. Deville, D., Grimaud, G. : Building an “impossible” verifier on a Java Card. In : 2nd USENIX Workshop on Industrial Experiences with Systems Software, Boston, USA (2002)
10. Chen, Z. : Java CardTM Technology for Smart Cards : Architecture and Programmer's Guide. Addison-Wesley (2000)
11. Sun microsystems : Java CardTM 2.2.1 Specifications. Sun microsystems (2003)
12. MULTOS : The MULTOSTM Specification. (<http://www.multos.com/>)
13. Hive-Minded : Smartcard.NET. (<http://www.hiveminded.com/>)
14. : .NET Brings Web Services to Smart cards. SmartCards Trends 1 (2004) 12
15. ZeitControl : BasicCard. (<http://www.basiccard.com/>)
16. Sauveron, D. : Étude et réalisation d'un environnement d'expérimentation et de modélisation pour les nouvelles technologies embarquées : application à la sécurité dans un environnement Java Card. PhD thesis, Université Bordeaux 1 (2004)
17. Watt, D.A., Brown, D.F. : Programming Language Processors in Java : Compilers and Interpreters. Prentice Hall (2000)
18. Jurgensen, T.M., Guthery, S.B. : Smart Cards : The Developer's Toolkit. Prentice Hall (2002)
19. CCIMB : International Common Criteria home page. (<http://www.commoncriteriaportal.org/>)
20. : ITSEC. (http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-fr.pdf)
21. Poll, E., van den Berg, J., Jacobs, B. : Specification of the JavaCard API in JML. In Domingo-Ferrer, J., Chan, D., Watson, A., eds. : Fourth Smart Card Research and Advanced Application Conference (CARDIS'2000), Kluwer Acad. Publ. (2000) 135–154
22. Poll, E., van den Berg, J., Jacobs, B. : Formal specification of the Java Card API in JML : the APDU class. Computer Networks 36 (2001) 407–421
23. Berg, J.v.d., Jacobs, B., Poll, E. : Formal Specification and Verification of JavaCard's Application Identifier Class. In Attali, I., Jensen, T., eds. : Java on Smart Cards : Programming and Security. Volume 2041 of Lecture Notes in Computer Science., Springer-Verlag (2001) 137–150
24. Meijer, H., Poll, E. : Towards a full formal specification of the Java Card. In Attali, I., Jensen, T., eds. : Smart Card Programming and Security. Number 2140 of Lecture Notes in Computer Science, Springer-Verlag (2001) 165–178
25. : The Java Modeling Language (JML) Home Page. (<http://www.jmlspecs.org/>)
26. : Formal interface specifications for the Java Card API 2.1.1 in JML and ESC/Java. (http://www.cs.kun.nl/~erikpoll/publications/jc211_specs.html)

27. : Extended Static Checking for Java. (<http://research.compaq.com/SRC/esc/>)
28. : PVS Specification and Verification System. (<http://pvs.csl.sri.com/>)
29. van den Berg, J., Jacobs, B. : The LOOP Compiler for Java and JML. In : Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer-Verlag (2001) 299–312
30. Trentelman, K., Huisman, M. : Extending JML specifications with temporal logic. In : Algebraic Methodology And Software Technology (AMAST '02). Volume 2422 of Lecture Notes in Computer Science., Springer-Verlag (2002) 334–348
31. Hubbers, E., Poll, E. : Reasoning about Card Tears and Transactions in Java Card. In : Fundamental Approaches to Software Engineering (FASE'2004). Volume 2984 of Lecture Notes in Computer Science., Barcelona, Spain, Springer-Verlag (2004) 114–128
32. Barthe, G., Dufay, G., Jakubiec, L., de Sousa, S.M., Serpette, B. : A Formal Executable Semantics of the JavaCard Platform. In Sands, D., ed. : Proceedings of ESOP'01. Volume 2028 of Lecture Notes in Computer Science., Springer Verlag (2001) 302–319
33. Siveroni, I., Hankin, C. : A Proposal for the JCVMLe Operational Semantics. Technical report, SecSafe (2001)
34. Barthe, G., Dufay, G., Huisman, M., Sousa, S. : Jakarta : a toolset to reason about the JavaCard platform. In Attali, I., Jensen, T., eds. : Smart Card Programming and Security (Proceedings of e-Smart'01). Volume 2140 of Lecture Notes in Computer Science., Springer Verlag (2001) 2–18
35. Caromel, D., Henrio, L., Serpette, B.P. : Context Inference for Static Analysis of Java Card Object Sharing. In : Proceedings of the International Conference on Research in Smart Cards, Springer-Verlag (2001) 43–57
36. Siveroni, I., Jensen, T., Eluard, M. : A Formal Specification of the Java Card Firewall. Technical report, SecSafe (2001)
37. Éluard, M., Jensen, T.P., Denney, E. : An Operational Semantics of the Java Card Firewall. In : Proceedings of E-smart 2001, Springer-Verlag (2001) 95–110
38. Perovich, D. : Secure Object Sharing Development Kit for JavaCard. Presented at the Verificard project workshop (2002)
39. Oestreicher, M. : Transactions in Java Card. In : Proceedings of 15th Annual Computer Security Applications Conference (ACSAC), Phoenix, Arizona, USA (1999)
40. Barthe, G., Dufay, G., Jakubiec, L., Melo de Sousa, S. : A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines. In : Proceedings of VMCAI'02. Volume 2294 of Lecture Notes in Computer Science., Venice, Italy, Springer-Verlag (2002) 32–45
41. Denney, E. : Correctness of Java Card Tokenisation. Technical report (1999) Projet Lande, IRISA.
42. Denney, E., Jensen, T. : Correctness of Java Card method lookup via logical relations. Theoretical Computer Science **283** (2002) 305–331
43. Schultz, A. : Verification of Java Card Applets. Master's thesis, Universität Kaiserslautern (2003)
44. : JIVE - Java Interactive Verification Environment. (<http://www.sct.inf.ethz.ch/projects/#jive>)
45. : Jcave - javacard applet verification environment. (<http://www.sics.se/fdt/vericode/jcave.html>)
46. Chugunov, G., Fredlund, L., Gurov, D. : Model Checking of Multi-Applet JavaCard Applications. In : Fifth Smart Card Research and Advanced Application Conf. (CARDIS'2002), USENIX (2002) 87–96
47. Martin, H., du Bousquet, L. : Automatic Test Generation for Java-Card Applets. In : Revised Papers from the First International Workshop on Java on Smart Cards : Programming and Security, Springer-Verlag (2001) 121–136
48. Team, E. : Environments for Verification and Security of Software. (<http://www-sop.inria.fr/everest/index.php>)
49. Rose, E., Rose, K. : Lightweight bytecode verification. In : In Workshop on Fundamental Underpinnings of Java, OOPSLA '98 Workshop., Vancouver, Canada (1998)

50. Casset, L., Burdy, L., Requet, A. : Formal Development of an embedded verifier for Java Card Byte Code. In : Proceedings of the IEEE International Conference on Dependable Systems & Networks, Washington, D.C., USA (2002)
51. Leroy, X. : On-Card Bytecode Verification for Java Card. In : Proceedings of the International Conference on Research in Smart Cards, E-Smart 2001, Springer-Verlag (2001) 150–164
52. Leroy, X. : Bytecode verification on Java smart cards. *Software-Practice & Experience* **32** (2002) 319–340
53. Cohen, R.M. : Defensive Java Virtual Machine Version 0.5 alpha. (1997)
54. Barthe, G., Dufay, G., Jakubiec, L., Melo de Sousa, S. : A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines. In : Proceedings of VMCAI'02. Volume 2294 of Lecture Notes in Computer Science., Venice, Italy, Springer-Verlag (2002) 32–45
55. Chaumette, S., Sauveron, D. : New security problems raised by open multiapplication smart cards. Technical report, LaBRI (2004)
56. Montgomery, M., Krishna, K. : Secure Object Sharing in Java Card. In : Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999)
57. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C. : The Sorcerer's Apprentice Guide to Fault Attacks. In : Proceedings of Workshop on Fault Detection and Tolerance in Cryptography, Italy (2004)
58. Giraud, C., Thiebaud, H. : A survey on fault attacks. In : Proceedings of CARDIS'04, Smart Card Research and Advanced Applications VI, Toulouse, France, Kluwer academic publisher (2004) 159–176
59. Skorobogatov, S., Anderson, R. : Optical Fault Induction Attacks. In : Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), San Francisco Bay (Redwood City), USA (2002)
60. Betarte, G., Giménez, E., Chetali, B., Loiseaux, C. : FORMAVIE : Formal Modelling and Verification of Java Card 2.1.1 Security Architecture. In : Proceedings of E-Smart 2002, Nice, France (2002) 215–229
61. Quisquater, J.J., Samyde, D. : Automatic Code Recognition for smart cards using a Kohonen neural network. In : Proceedings of the 5th Smart Card Research and Advanced Application Conference (CARDIS'02). (2002)
62. Girard, P., Lanet, J.L. : Java Card or How to Cope with the New Security Issues Raised by Open Cards ? In : Proceedings of Gemplus Developer Conference, Paris, France (1999)
63. Girard, P., Lanet, J.L. : New Security Issues raised by Open Cards. In : Information Security Technical Report. Volume 4. (1999) 19–27
64. Girard, P. : Which security policy for multiapplication smart cards ? In : Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 21–28
65. Bieber, P., Cazin, J., Girard, P., Lanet, J.L., Wiels, V., Zanon, G. : Checking Secure Interactions of Smart Card : Applets Extended Version. *Computer Security* **10** (2002) 369–398 Special issue on ESORICS 2000.
66. Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V., Toll, D. : Verification of a Formal Security Model for Multiapplicative Smart Cards. In : Proceedings of ESORICS 2000. Volume 1895 of Lecture Notes in Computer Science. (2000) 17–36