HAL
open science

Analyse de la sécurité de transactions à puce avec le framework WinSCard Tools

Benoît Vibert, Vincent Alimi, Sylvain Vernois

To cite this version:

Benoît Vibert, Vincent Alimi, Sylvain Vernois. Analyse de la sécurité de transactions à puce avec le framework WinSCard Tools. SAR-SSI 2012, 2012, Cabourg, France. pp.8. hal-00697465

HAL Id: hal-00697465

https://hal.science/hal-00697465v1

Submitted on 15 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analyse de la sécurité de transactions à puce avec le framework WinSCard Tools

Benoît Vibert, Vincent Alimi, Sylvain Vernois

Laboratoire Greyc, ENSICAEN - CNRS, UCBN, Caen, FRANCE
6 Boulevard Maréchal Juin, F-14050 Caen Cedex 4
Tel : 33 (0) 2 31 53 81 48 - Fax : 33 (0) 2 31 53 81 10
benoit.vibert@ensicaen.fr, vincent.alimi@ensicaen.fr, sylvain.vernois@ensicaen.fr

Résumé

Une carte à puce est souvent considérée comme l'un des éléments les plus puissants et sécurisés pour les systèmes de paiement. L'analyse du développement et de la sécurité des transactions utilisant une carte à puce est une tâche complexe car elle implique de nombreux standards. Dans cet article, nous présentons une API ouverte et orientée objet pour l'exploration des cartes à puce d'un point de vue recherche. Nous montrons comment créer facilement une suite d'outils permettant de simuler des transactions de paiement EMV et de reproduire des attaques sur cartes à puce présentes dans la littérature.

1. Introduction

En France en 2010, on enregistrait 7 milliards de paiements par carte bancaire pour un montant de 336 milliards d'euros, et 1,5 milliards de retraits aux distributeurs pour un montant de 115 milliards d'euros [1]. Le système bancaire étant un domaine sécurisé, bon nombre de chercheurs tentent de trouver des failles dans le système. Nous nous intéresserons plus spécialement à la carte à puce et aux outils qui permettent de communiquer avec celle-ci.

Lors du développement d'une application exploitant une carte à puce, la première chose requise est une interface de programmation (*Application Programming Interface*, API) permettant d'accéder aux ressources du lecteur et à la puce elle-même. Nos besoins ne sont ni le développement d'une application embarquée ni la validation de la plateforme, mais de disposer d'outils facilitant l'accès aux ressources des cartes à puce à partir d'un ordinateur, afin de développer des applications permettant l'étude de multiples aspects d'une application embarquée sur une carte à puce.

Pour ce besoin, nous aurions pu utiliser une API propriétaire, fournie par le fabricant du lecteur de carte, avec pour défaut majeur le fait de ne pas être réutilisable si le lecteur de carte vient à changer. Une autre solution consiste à utiliser une API publique et reconnue telle que PC/SC [2], l'abstraction du lecteur étant fournie.

Dans un premier temps, nous ferons un état des lieux des outils logiciels et frameworks permettant de manipuler une carte à puce, ainsi que des attaques logicielles pouvant être menées. Nous présenterons ensuite WinSCard Tools et montrerons les opportunités que celui-ci offre aux chercheurs du domaine. Enfin, nous illustrerons l'usage de WinSCard Tools par la reproduction de l'attaque dite de "Cambridge" consistant à intercepter le code PIN lors d'une transaction EMV.

2. État des lieux

Dans cette section, nous présentons tout d'abord les logiciels et API disponibles publiquement qui nous semblent apporter une plus value pour des besoins de recherche par rapport à l'API de communication PC/SC. Cette liste ne se veut pas exhaustive et les présentations sont succinctes. Nous décrivons ensuite les différentes attaques présentes dans la littérature, ainsi que les outils avec lesquels elles ont été réalisées.

2.1. Environnement logiciel

Trois frameworks ont particulièrement attiré notre attention. Tout d'abord, celui développé par l'entreprise SpringCard [3] dont l'API est documentée et le code source est propriétaire, et celui nommé

SmartCard Framework proposé par Orouit [4] dont le code source est disponible sous licence CPOL [5] et proposé en guise de tutoriel de programmation. Tous deux sont fonctionnels et apportent un ensemble d'objets intéressants pour faciliter la communication entre l'application et la carte. Cependant ils possèdent deux défauts importants pour nos objectifs : le premier est de ne pas proposer de mécanisme d'extension des fonctionnalités, leur rôle originel étant essentiellement de faciliter l'accès aux couches PC/SC ; le second est de ne pas permettre la gestion complète de l'ensemble des protocoles de transmission prévus par la norme.

Le dernier est une librairie récente nommée *pcsc-sharp* et développée par Daniel Müller [6]. L'auteur a pris soin de mettre en place une architecture modulaire et évolutive, tout en proposant des objets de programmation permettant d'implémenter la norme sans restriction. Elle se démarque fortement par ces aspects des autres API disponibles.

Si développer nous même une application pour étudier le comportement des cartes à puce était une option qui nous intéressait, nous nous sommes aussi intéressés aux logiciels déjà disponibles dont l'utilisation aurait pu répondre au moins à une partie de nos besoin. Les trois logiciels suivants ont attiré notre attention, car ils permettent d'aller plus loin que le simple envoi de commande à la carte ou l'exécution d'un script statique.

- Gem_PCSC [7] de Gemalto permet d'agir au niveau des paramètres de connexion PC/SC ainsi que sur le protocole de transmission utilisé avec la carte (T=0 ou T=1). Il propose de plus la sauvegarde des commandes exécutées sous forme de script afin de les rejouer ultérieurement. L'interface est intuitive et permet de voir toutes les informations nécessaires et de suivre l'état de la communication.
- CardPeek [8] permet d'avoir accès à toutes les informations personnelles stockées sur une carte à puce telles que l'historique des transactions sur une carte de paiement. L'application est actuellement capable de lire les cartes bancaires EMV, les cartes Navigo, Monéo, Vitale 2, les passeports électroniques/biométriques avec une sécurité BAC, les cartes SIM GSM ainsi que les cartes sans-contact Mifare. L'interface permet de consulter l'ensemble des données extraites de la carte sous une forme arborescente. Enfin, élément intéressant, il propose un mécanisme de script (en langage LUA) afin d'explorer des cartes non prévues nativement.
- Enfin SmartCard ToolSetPro [9] de la société SCardSoft est certainement la solution la plus complète actuellement disponible. En effet, la suite logicielle propose un excellent contrôle sur le lecteur, la carte et les échanges d'APDU ayant été réalisés, ainsi qu'un mécanisme d'extension afin de s'adapter aux besoins.

2.2. Attaques sur les cartes à puces dans la littérature

Depuis sa création à la fin des années 1970, la carte à puce a connu une grande évolution. Simple carte à mémoire à ses débuts, puis carte à microprocesseur et aujourd'hui assistée d'un coprocesseur cryptographique et d'un générateur de nombres aléatoires ; la carte s'est complexifiée tant au niveau matériel que logiciel. Aujourd'hui une carte à puce embarque des plateformes logicielles complètes capables d'accueillir plusieurs applications écrites dans des langages publics et accessibles.

Mais cette complexification s'est également traduite par un besoin accru de sécurité. La sécurité de la carte à puce est assurée par quatre composants [10] :

- le corps de la carte,
- les composants matériels de la puce,
- le système d'exploitation (le masque),
- l'application.

Chacun de ses composants est sensible à certaines attaques expérimentées ces deux dernières décennies. Les premières attaques avaient principalement trait au corps de la carte et aux composants de la puce. Parmi ces attaques, que nous appellerons attaques classiques, on référence trois principaux types d'attaques [11] :

- les attaques *invasives*, dans lesquelles le microprocesseur est retiré de la puce et directement attaqué ;
- les attaques *semi-invasives*, où la surface de la puce est exposée et soumise à des contraintes ;
- les attaques *non-invasives*, dans lesquelles on cherche à obtenir des informations sans apporter de modifications à la carte (par exemple en observant les signaux électriques et émanations électro-magnétiques).

Puis d'autres attaques ont vu le jour grâce notamment à l'apparition des plateformes multi-applicatives ouvertes. Ces plateformes permettent de développer aisément des applications et de les installer sur une carte. Un attaquant peut alors charger une application malicieuse qui produira des attaques dites *internes*. Parmi ces attaques on compte :

- l'identification de service,
- la collecte d'information,

– les attaques contre les mécanismes de la plateforme.

De *nouvelles classes d'attaques* apparurent ensuite. Ces attaques tirent à la fois partie des attaques classiques et internes, *i.e.* elles consistent en l'observation des signaux au moment de l'injection de fautes logicielles. Toutes ces attaques ont bien entendu des contre-mesures. Nous étudierons ici les différents types d'attaques et les contre-mesures connues. On pourra trouver dans la littérature des classifications des attaques différentes de celles présentées ici. Par exemple, dans [12] les attaques sont catégorisées comme suit : attaques par canaux cachés, attaques par injection de fautes et attaques sur les plateformes multi-applicatives.

2.3. Discussion

Les environnements logiciels présentés, sous forme d'API de programmation ou de logiciels prêts à l'emploi, ne parvienne pas à répondre à l'ensemble de nos besoins tels que la généricité pour s'adapter à tout type de puce ou l'analyse précise des résultats selon une méthodologie. A l'exception notable de la librairie pcsc_sharp [13] qui, si elle été apparue quelques années plus tôt, aurait été un candidat très sérieux. Concernant les attaques, elles sont nombreuses (SCA (Side Channel Analysis) [14], Temps d'exécution, consommation de courant, SPA (Simple Power Analysis), DPA (Differential Power Analysis), injection de fautes, attaques sur les applications, sur le bytecode, identification de service, attaque sur le firewall de la carte), et la reproductibilité n'est souvent pas aisée, du fait d'environnements généralement montés spécifiquement pour chacune d'elle. C'est la raison pour laquelle nous avons développé depuis plusieurs années notre ensemble d'API et d'outils [15], qui sont présentés par la suite.

3. Framework WinSCard Tools

Voici une première contribution pour répondre aux problèmes décris dans la section précédente et permet la reproductibilité des résultats de recherche.

Le logiciel développé est divisé en deux parties, chacune réalisant une fonction spécifique. La première est une interface de programmation orientée objet encapsulant les bibliothèques dynamiques natives PC/SC sous Windows et Linux. La seconde est une interface utilisateur, visant à faciliter l'utilisation du lecteur de carte à puce et proposant un mécanisme d'extension modulaire qui sert de support au développement d'applications de test et d'analyse.

3.1. L'API pour cartes à puce

En 2004, une première implémentation en Java de la spécification ISO FDIS/IEC 24727-2 [16] a été développée à l'ENSICAEN, connue sous le nom de `Cardstack` [15]. La nouvelle API développée a bénéficié de cette expérience et acquis une pile conforme à la norme. Concrètement, l'implémentation de cette norme permet d'ajouter des couches logicielles d'abstraction entre l'application et la carte à puce. Ces couches étant indépendantes à la fois du module réalisant l'envoi des commandes et de la puce, elles permettent divers activités transparentes telles que la surveillance des échanges réalisés (activité passive) ou la modification des échanges eux mêmes (activité active) à des fins de tests de résistance aux erreurs par exemple. Une autre utilisation type de ces couches est la mise en communication d'un logiciel et d'une carte a priori incompatibles (commandes non supportées fréquemment) de façon transparente par modification au vol des APDU échangées. La figure 1 résume l'apport fonctionnel de l'API ainsi mise en place. De plus, la pile de protocoles (un ensemble de couches) implémentée présente elle même les caractéristiques d'une couche, apportant ainsi une modularité importante.

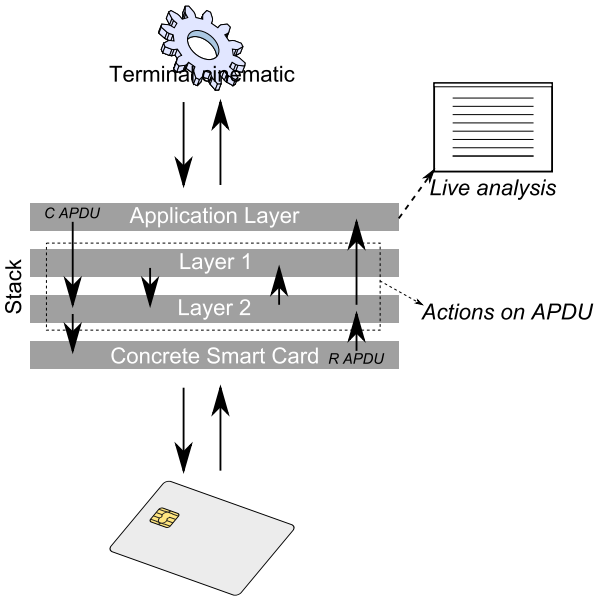

FIGURE 1. Plus value fonctionnelle apportée par l'API de *WinSCard Tools*

3.2. Une interface utilisateur

Un outil d'accompagnement, servant de preuve de concept, a été réalisé simultanément à la pile de pro-

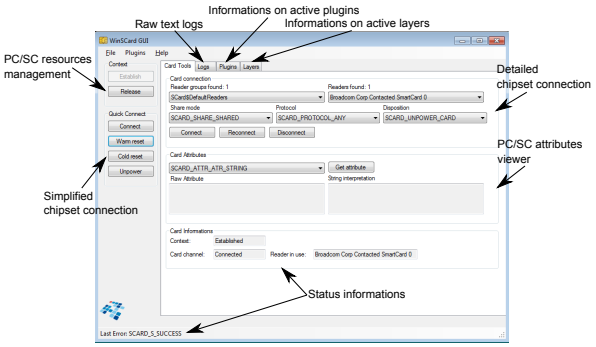

FIGURE 2. Fenêtre principale de l'interface graphique de *WinSCard Tools*

tocole. Il a évolué vers une interface graphique (GUI) exploitant toutes les capacités de l'API et permettant une extensibilité grâce à un système de plugin, comme illustré à la figure 3.

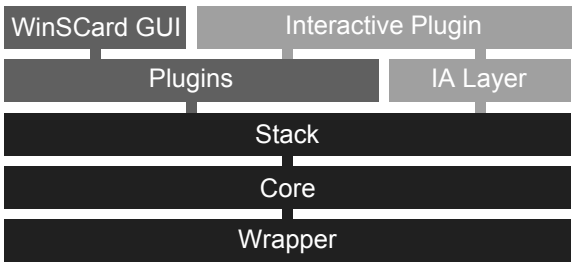

FIGURE 3. Architecture de l'interface graphique

3.2.1. L'interface graphique principale. L'interface graphique principale (*cf.* figure 2) est responsable de la gestion des ressources : accès aux pilotes PC/SC sous-jacents et gestion du canal établi entre l'utilisateur et la carte elle-même.

3.2.2. Le système de plugins. Un système d'extension est utilisé pour étendre les fonctionnalités de l'interface graphique, afin d'implémenter facilement des petites applications indépendantes (les plugins) répondant à un besoin bien précis.

Tous les plugins et l'interface graphique principale partagent un contexte et un canal uniques offerts par l'API. En utilisant activement les mécanismes de communication par évènement, plusieurs plugins peuvent simultanément accéder aux données échangées entre l'application cliente et la carte de manière transparente, offrant ainsi la possibilité d' analyser "à la volée".

3.2.3. Enregistrement et rejeu d'échanges APDU. En utilisant l'interface graphique pour des projets éducatifs et de recherche, nous avons rencontré un besoin récurrent : pouvoir lire et rejouer un ensemble de réponses provenant d'une carte pour permettre des analyses multiples. Le même jeu de commandes peut être envoyé à plusieurs reprises à la même carte, mais dans notre cas cela ne suffit pas car une transaction met en jeu des nombres aléatoires générés et des compteurs incrémentés par la carte. Le même scénario ne peut donc être reproduit à des fins d'études plus poussées. C'est pourquoi une nouvelle couche, désignée comme "Interactive layer" sur la figure 3 a été ajoutée. En utilisant le mécanisme de pile, cette couche est introduite entre l'application et la carte à puce et propose trois modes (*cf.* figure 4) :

- Le mode *record* permet de mémoriser et d'enregistrer les données échangées.
- Le mode *replay* permet de rejouer les données précédemment mémorisées. Dans ce mode, la carte n'est plus physiquement accéder, mais la couche Interactive renvoie les données directement à l'application, qui n'a pas connaissance de la "fausse" carte.
- Le mode *transparent* agit en "pass through", sans action sur la communication.

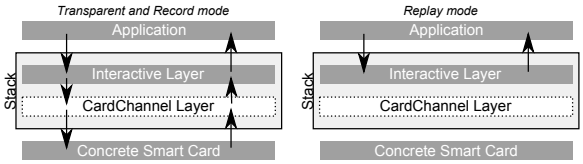

FIGURE 4. Illustration des modes de fonctionnement de la couche Interactive

Dans cette section, nous avons présenté l'API et les outils graphiques créés pour simplifier l'accès aux ressources de la carte à puce. L'architecture ouverte, basée sur des modèles de conception simple, permet non seulement la communication entre une application (ou un plugin) et la carte, mais fournit également des moyens pratiques pour analyser et modifier les données.

4. Application à l'analyse des transactions

4.1. Une librairie EMV à des fins de recherche

EMV (Europay MasterCard Visa)[17] est une spécification fréquemment utilisée pour les transactions de paiement par cartes à puce, particulièrement parce qu'elle est imposée par les principaux réseaux de paiement dans le monde entier : Mastercard, Visa

et JCB. C'est pourquoi nous avons mis en place une bibliothèque EMV, basée sur l'API développée, avec un plugin dédié pour gérer la transaction. Cette bibliothèque a été réalisée avec les mêmes objectifs que l'API : en particulier elle expose plusieurs *événements* pour surveiller les fonctionnements internes.

Le plugin associé permet de piloter une transaction EMV, étape par étape, et d'analyser avec précision toutes les APDU conformes aux spécifications EMV, comme le contrôle des cryptogrammes générés par la carte. Aussi, certains composants ont été adaptés pour permettre à certaines cartes sans contact basées sur EMV d'être utilisées. Un exemple d'utilisation à des fins de recherche de la bibliothèque EMV est détaillé dans la section suivante.

4.2. Rejeu de l'attaque de "Cambridge"

4.2.1. Présentation. L'attaque de "Cambridge" a été réalisée dans le *Computer Laboratory* de l'*Université de Cambridge* et exposée dans [18]. Dans cet article, Murdoch *et al.* démontrent la faisabilité d'une attaque sur une carte à puce EMV. Il s'agit d'une attaque de type *man-in-the-middle* sur la vérification du code PIN (Personal Identification Number). Elle consiste à intercepter la commande VERIFY envoyée par le TPE (Terminal de Paiement Electronique) à la carte à puce afin de vérifier le code PIN saisi par le porteur de la carte. Elle s'appuie sur le fait qu'une carte à puce EMV peut supporter d'autres méthodes de vérification du porteur (CVM) que le code PIN *"offline"* - *i.e.* vérifié par la carte à puce -, tels que la vérification de la signature du porteur et la vérification du code PIN *"online"* par la banque émettrice. Donc, si l'on réussit à intercepter la commande et envoyer une réponse positive au TPE à la place de la carte, à la fois le TPE et la carte "pensent" que la vérification du porteur a été un succès et que la transaction peut suivre son cours. Murdoch *et al.* ont mis en œuvre cette attaque de manière matérielle. Elle est composée d'une fausse carte à puce connectée à une carte FPGA. La carte FPGA est connectée à un ordinateur portable, qui est lui-même connecté par un lecteur de carte à puce à une carte à puce EMV volée (*cf.* Figure 5).

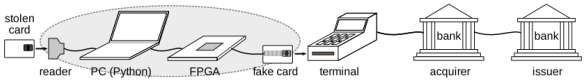

FIGURE 5. Matériel utilisé pour mener l'attaque (source : [18] courtoisie de Murdoch *et al.*)

Un script python s'exécutant sur l'ordinateur portable relaie toutes les commandes envoyées par le TPE à la carte. Il attend une commande VERIFY et, au lieu de la relayer à la carte, renvoie le code 0x9000 à la carte. À ce stade, le point de vente estime que le code PIN a été vérifié avec succès par la carte et poursuit la transaction de paiement.

4.2.2. Implémentation logicielle de l'attaque. A des fins de recherche et d'enseignement, nous avons reproduit cette attaque de manière logicielle grâce à *WinSCard Tools*. En effet l'architecture, de *WinSCard Tools* permet de définir une couche "man-in-the-middle" (MITM) pour implémenter cette attaque.

Dispositif expérimental. La couche MITM est ajoutée à la pile de communication et relaie toutes les commandes envoyées par *WinSCard Tools* à la carte, à l'exception de VERIFY, d'une (*cf.* figure 6). Au lieu de cela, il renvoie 0x9000 à *WinSCard Tools*. Ensuite, la transaction continue normalement et on peut observer que la carte valide la transaction.

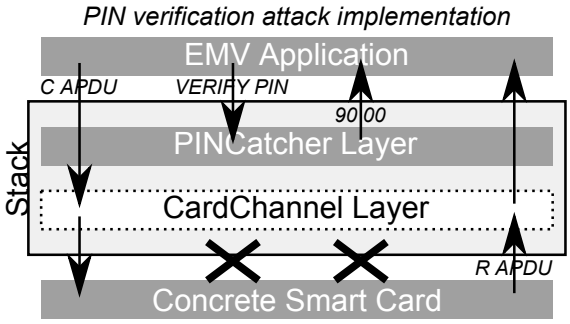

FIGURE 6. Implémentation logicielle de l'attaque sur le code PIN

Grâce à *WinSCard Tools* , il est possible d'observer ce qui se passe du côté du TPE lors d'une transaction. En effet, dans une transaction EMV, il est possible d'espionner le lien entre la carte et le TPE, mais il n'est pas possible de connaître la façon dont le POS traite les données, par exemple la façon dont le TPE vérifie la validité des certificats de la carte à puce et les données qu'il en extrait. Comme *WinSCard Tools* émule complètement un TPE et embarque la librairie EMV vue précédemment, ces traitements sont totalement accessibles à l'utilisateur. Une interface utilisateur dédiée présente les différents certificats, le résultat du processus de vérification de signature et les données contenues dans les certificats.

Une autre fonction de *WinSCard Tools* est particulièrement utile pour rejouer les attaques : la couche Interactive. Cette couche permet d'enregistrer et rejouer les transactions. Elle enregistre les données envoyées par la carte à la réception des commandes et peut les

rejouer au nom de la carte en mode de lecture. Dans la mise en œuvre de ce genre d'attaque, cette fonctionnalité est intéressante car elle permet de conserver, par exemple, le compteur du nombre de transactions inchangée. En effet, pour chaque transaction initiée, un compteur est incrémenté. Ce compteur est utilisé, avec d'autres éléments, par la carte pour décider si le terminal doit interroger la banque pour autoriser la transaction. Il est alors intéressant de maintenir inchangé les données d'une carte à puce EMV véritable lorsque l'on effectue une certaine quantité de tests et d'attaques.

Mode opératoire. Une fois l'environnement expérimental établi, une carte à puce EMV est insérée dans le lecteur et le plugin "EMV Explorer" est lancé. EMV Explorer (*cf.* Figure 7) simule le terminal de paiement du point de vente et permet de traiter une transaction EMV étape par étape. Il affiche toutes les commandes APDU échangées entre la carte et le terminal et les interprète grâce à la bibliothèque EMV.

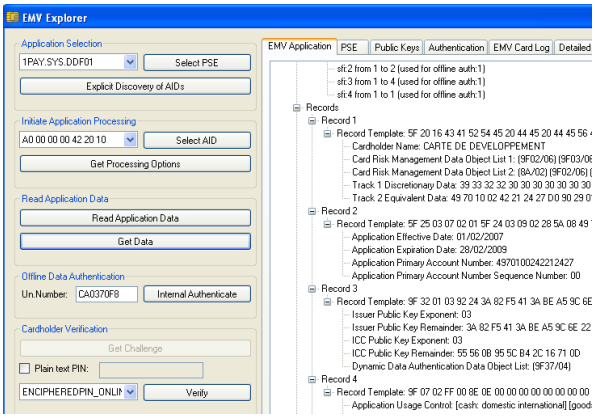

FIGURE 7. Le plugin "EMV Explorer"

A l'étape de la vérification du porteur, un code PIN est saisi et on peut observer que le POS reçoit le mot 0x9000 (traitement normal) puis procède à la commande suivante (Generate AC).

Résultats et discussion. L'attaque a été effectuée avec succès sur toutes les cartes à puce EMV en notre possession. Aucune d'elles n'a été capable de détecter l'attaque (les certificats générés sont valides), car elles permettent d'autres CVM que la vérification du code PIN. Ceci est également dû au fait que les émetteurs n'ont pas implémenté de contre-mesure dans les versions des applications que nous avons sur nos cartes à puce. Mais, depuis quelques années l'implémentation d'une contre-mesure est prévue dans les nouvelles spécifications d'application - de Visa et MasterCard, par exemple. Elle consiste souvent à passer l'élément de données *CVM Results* à la carte lors de l'envoi de la commande GENERATE AC. Pour protéger la transmission du CVM Results d'une autre attaque man-in-the-middle nous recommandons d'utiliser CDA (Combined DDA/Application Cryptogram Generation) pour signer de manière dynamique les données envoyées à la carte.

Nous avons vu dans cette section comment nous avons été en mesure de reproduire l'attaque dite "de Cambridge" de manière logicielle. Bien sûr, ceci ne peut pas être mis en œuvre dans la vie réelle avec de réels points de vente tels que les gens de Cambridge l'ont fait, mais nous avons démontré que *WinSCard Tools* est un outil parfaitement adapté à un travail de recherche sur les cartes à puce en général et sur les transactions EMV en particulier.

4.3. Interception et modification d'une transaction de paiement

4.3.1. Présentation. L'attaque de Cambridge a démontré qu'une attaque de type *man-in-the-middle* sur une transaction de paiement EMV – qui était connu depuis quelques temps déjà – n'était pas que le fruit de recherches théoriques mais avait maintenant une implémentation réelle sur le terrain. En effet, l'équipe de Ross Anderson, après avoir publié ses résultats, a fait l'objet de nombreux articles de presse [19], [20] et d'un reportage vidéo [21] dans lequel elle fait la démonstration de son attaque dans un magasin.

Grâce à *WinSCard Tools* nous faisons également la démonstration de la faisabilité de telles attaques mais avec un dispositif beaucoup plus réduit. En effet, le matériel constitué d'une carte FPGA et le logiciel en python pour l'analyse des APDUs sont regroupés en un seul outil logiciel.

Dans les sections suivantes nous étudions le dispositif expérimental, discutons les résultats obtenus et concluons en proposant une amélioration significative à ce dispositif. Nous commençons par donner au lecteur les informations sur EMV nécessaires à la bonne compréhension de cette attaque.

4.4. Mécanismes d'authentification dans EMV

Le standard EMV permet de réaliser trois types d'authentification : authentification de la carte, authentification du porteur et authentification de la banque. Pour l'authentification de la carte, il existe trois mécanismes. Le premier mécanisme est statique, appelé *Static Data Authentication* (SDA), et le principe est que la carte est personnalisée à la fois avec des données

en clair et ces même données mais signées par la banque. Ce mécanisme présente l'inconvénient majeur de permettre le clonage des cartes. Pour y palier, un mécanisme dynamique appelé *Dynamic Data Authentication* (DDA) a vu le jour. DDA consiste en une génération dynamique des données signées à chaque transaction et en utilisant un challenge envoyé par le terminal de paiement. Un dernier mécanisme appelé *Combined Data Authentication* (CDA) consuste a générer une signature dynamique comme pour DDA mais appliquée aux données d'une commande bien particulière, *GENERATE APPLICATION CRYPTOGRAM*. La commande GENERATE AC est utilisée dans EMV pour faire générer un cryptogramme par la carte attestant de la décision prise (communément par la carte et le terminal) pour la transaction en cours (abandon, acceptée *offline* ou aller *online*).

4.4.1. Dispositif expérimental. L'attaque que nous avons choisi d'implémenter est une interception et une modification des données d'une transaction de paiement sans-contact. Le dispositif est constitué de l'outil *WinSCard Tools* auquel sont raccordés un lecteur et une sonde sans contact. Une vraie carte de paiement sans-contact est posée sur le lecteur tandis que la sonde est présentée à un terminal de paiement sans contact (*cf.* figure 8). Un plugin spécialement développé relaie les commandes du terminal à la carte et vice versa. Nous utilisons ce dispositif non plus pour faire du simple relai de commande mais pour les intercepter et les modifier.

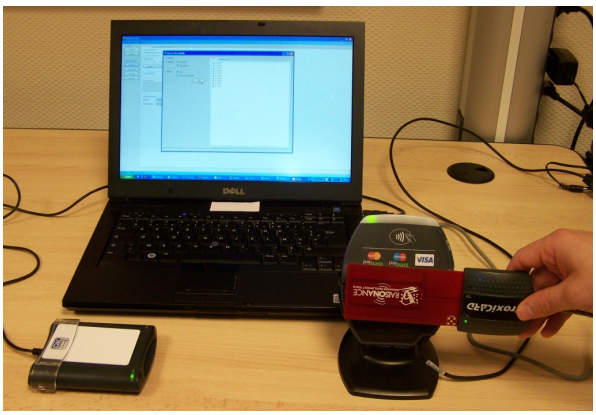

FIGURE 8. Dispositif d'attaque

4.4.2. Mode opératoire. Une fois le dispositif mis en place, on pose sur le lecteur une carte de paiement sans contact et la sonde sur le terminal de paiement. La carte implémente le mécanisme d'authentification SDA. Pour vérifier une signature SDA, le terminal récupère de la carte le certificat émetteur (*i.e.* la clé publique de l'émetteur signée par l'autorité de certification) et les données signées. Il vérifie alors le certificat, en extrait la clef publique et vérifie la signature des données. Cette vérification statique des données est sensible au rejeu et au clonage de la carte car ce sont toujours les mêmes données qui sont échangées entre la carte et le terminal.

Nous avons réussi à mettre en œuvre cette menace grâce au plugin *Man in the middle*. Nous avons réalisé un certain nombre de transactions jusqu'à ce que le compteur du nombre de transactions consécutives réalisées *offline* arrive à son plafond. A présent la carte refuse toute transaction. Grâce au plugin, nous interceptons les données renvoyées par la carte et les modifions afin de faire "croire" au terminal que la transaction est acceptée. Si cette attaque était réalisée chez un marchand, ce dernier délivrerait le bien acheté alors que la carte avait l'intention de refuser la transaction.

Pour obtenir l'autorisation de paiement le terminal demande à la carte de générer un cryptogramme accompagné d'un élément d'information indiquant la décision prise par la carte. Trois décisions sont possibles : refus de la transaction (*Application Authentication Cryptogram*, AAC), acceptation *offline* de la transaction (*Transaction Certificate*, TC) ou demande d'autorisation à la banque (*Authorisation Request Cryptogram*, ARQC). Le plugin intercepte les éléments retournés par la carte et modifie l'information sur le cryptogramme afin de modifier le type de cryptogramme retourné d'AAC à TC (figure 9).

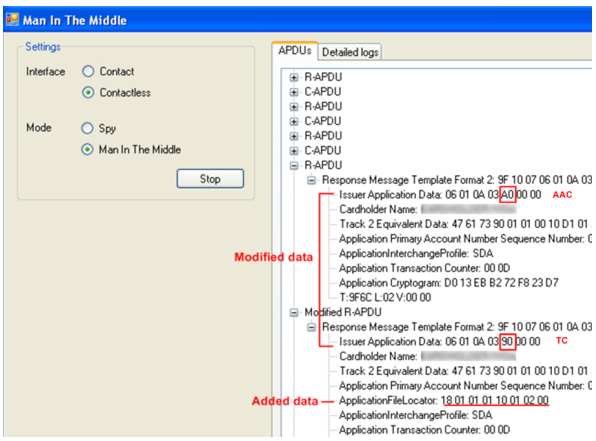

FIGURE 9. Modification du type de cryptogramme retourné par la carte

4.4.3. Résultats et discussion. Cette attaque s'avère efficace sur les cartes de type SDA. Pour lutter contre le rejeu des transactions, depuis le 1er janvier 2010

toutes les cartes bancaires émises en Europe doivent implémenter DDA. Contrairement à SDA, les données signées ne sont pas personnalisées dans la carte mais générées de manière dynamique à chaque transaction. La signature dynamique est générée à l'aide de la clef privée de la carte ce qui induit que celle-ci implémente l'algorithme RSA, par le biais d'un processeur cryptographique par exemple. Nous pourrions toujours modifier les données échangées mais la vérification de la signature dynamique des données permettrait de détecter l'attaque.

Dans le plugin *Man in the middle*, l'implémentation de l'attaque est codée "en dur" dans le code source et de manière spécifique au type de carte visée. Nous comptons améliorer le plugin de manière significative en décrivant dans un fichier XML des heuristiques d'attaque sur les cartes à puce afin que le logiciel s'adapte automatiquement à l'application avec laquelle se réalise la transaction et qu'il implémente les attaques applicables en fonction du contexte.

Dans de futurs travaux, nous envisageons également de porter *WinSCard Tools* sur un équipement portatif de type assistant personnel numérique équipé d'une plateforme Windows pour mobiles. En effet, comme ce système d'exploitation embarque la machine virtuelle .net, ceci nous permettrait de réduire de manière considérable l'encombrement du dispositif d'attaque. Le lecteur de carte sans contact pourrait également être remplacé par l'intégration de la technologie *NFC* (*Near Field Communication*) [22] car cette dernière permet à un équipement d'agir comme un lecteur de tags ou de cartes sans-contact RFID [23].

5. Conclusion

Nous avons présenté dans cet article l'intérêt de développer une API ouverte et orientée objet pour la recherche dans le domaine des cartes à puce. En effet, en proposant une encapsulation de la couche PC/SC et du protocole des cartes à puce ISO 7816, il est très facile de développer un outil répondant exactement aux fonctionnalités nécessaires. A titre d'illustration, il a été très facile d'utiliser l'API pour développer une bibliothèque EMV et un outil de mise en œuvre d'une attaque man-in-the-midle sur la vérification du PIN code et sur l'usurpation de l'autorisation de paiement donnée par une carte. WinSCard Tools est ainsi un outil permettant la reproductibilité des recherches présentes dans la littérature. Afin de continuer dans ce sens, nous travaillons actuellement activement sur l'attaque par Fuzzing [24] pour la reproduire avec l'aide de WinSCard Tools et valider de nouveaux scénarios.

Remerciements

Les auteurs souhaitent remercier Mr René Lozach, important contributeur aux standards ISO 7816 & 24727, pour ses précieux commentaires et Steven J. Murdoch *et al.* d'avoir accepté la reproduction de la figure 5 dans cet article.

Références

[1] *Groupement Carte Bancaire. Rapport d'activité cartes bancaire 2010*. http ://www.cartes-bancaires.com/IMG/pdf/Rapport_d_Activite_CB_2010.pdf.

[2] PC/SC Workgroup, http ://www.pcscworkgroup.com/. *PC/SC Workgroup Specifications 2*, 2005.

[3] *SmartCard framework API by SpringCard*. http ://www.springcard.com/download/sdks.html. vu en novembre 2011.

[4] *SmartCard framework API in .net by Orouit*. http ://www.codeproject.com/KB/smart/smartcardapi.aspx. vu en novembre 2011.

[5] Licence Code Project Open Licence. http ://www.codeproject.com/info/cpol10.aspx.

[6] *SmartCard Library in .net by Daniel Müller*. http ://danm.de/index.php ?action=source.

[7] *Gem_PCSC Tool*. http ://support.gemalto.com/ ?id=199. vu en novembre 2011.

[8] *CardPeek*. http ://code.google.com/p/cardpeek/. vu en novembre 2011.

[9] *SmartCard ToolSetPro V3.4 by ScardSoft*. http ://www.scardsoft.com/main.php3 ?Theme=Soft_v3Server. vu en novembre 2011.

[10] W. Rankl and W. Effing. *Smart Card Handbook*. 2003.

[11] Michael Tunstall. *Smart card security*. Springer, 2008.

[12] Constantinos Markantonakis, Keith Mayes, Michael Tunstall, Damien Sauveron, and Fred Piper. Smart card security. In Nadia Nedjah, Ajith Abraham, and Luiza de Macedo Mourelle, editors, *Computational Intelligence in Information Assurance and Security*, volume 57 of *Studies in Computational Intelligence*, pages 201–233. Springer, 2007.

[13] *PCSC_sharp*. http ://code.google.com/p/pcsc-sharp/. vu en novembre 2011.

[14] Paul C. Kocher. *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, pages 104Ú113. springer-verlag edition, 1996.

[15] *Contribution de Mr René Lozach*.

[16] ISO/IEC, http ://www.iso.org. *24727 : Identification cards - Integrated circuit card programming interfaces*.

[17] EMVCo. EMV integrated circuit card specifications for payment systems. Technical report, EMVCo, 2008.

[18] Steven Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and PIN is broken. In David Evans and Giovanni Vigna, editors, *SSP 2010, 31st IEEE Symposium on Security & Privacy*, Piscataway, NJ, USA, May 2010. IEEE Computer Society Technical Committee on Security and Privacy/The International Association for Cryptologic Research, IEEE Computer Society.

[19] Richard Alleyne. Chip and pin card readers fundamentally flawed. http ://www.telegraph.co.uk/science/science-news/7215920/Chip-and-pin-card-readers-fundamentally-flawed.html, 2010.

[20] Susan Watts. New flaws in chip and pin system revealed. http ://www.bbc.co.uk/blogs/newsnight/-susanwatts/2010/02/new_flaws_in_chip_and _pin_syst.html, 2010.

[21] YouTube. Uk chip and pin credit / debit cards are insecure. http ://www.youtube.com/-watch ?v=JPAX32lgkrw, 2010.

[22] Gerald Madlmayr, Josef Langer, Christian Kantner, and Josef Scharinger. Nfc devices : Security and privacy. *Availability, Reliability and Security, International Conference on*, 0 :642–647, 2008.

[23] Dr K. Finkenzeller. *RFID Handbook : Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, Third Edition*. John Wiley and Sons, 2010.

[24] Julien Lancia. Un framework de fuzzing pour cartes à puce : application aux protocoles emv. *SSTIC*, 2011.