Normandie Université

THESE

Pour obtenir le diplôme de doctorat

Spécialité 4200040

Préparée au sein de « Université de Caen / ENSICAEN »

Evaluation d'applications de paiement sur carte à puce

Présentée et soutenue par
Germain JOLLY

Thèse soutenue publiquement le 08 Juillet 2016 devant le jury composé de		
M. Benjamin NGUYEN	Professeur à l'INSA centre Val de Loire	Rapporteur
M. Lionel BRIAND	Professeur à l'université de Luxembourg	Rapporteur
M. Pierre PARADINAS	Professeur au CNAM Paris	Examinateur
M. Christophe DOLABDJIAN	Professeur à l'Université de Caen Normandie	Examinateur
M. Sylvain VERNOIS	Ingénieur de recherche ENSICAEN	Co-encadrant
M. Christophe ROSENBERGER	Professeur à l'ENSICAEN	Directeur de thèse

Thèse dirigée par Christophe ROSENBERGER, laboratoire GREYC

ENSICAEN
École Nationale Supérieure d'Ingénieurs de Caen

ED 181 SIMEM

GREYC

Pour Snowy

Remerciements

Je tiens à remercier en vers de douze pieds
Toutes ces personnes avec qui j'ai échangé,
Sans qui je n'aurai pas pu créer au fil des années.

Je tente de résumer en quelques vers, ces années
Sous la forme d'un sonnet, peut être un peu léger,
Avec lequel j'espère ne pas vous assommer :

Cette aventure prend fin, après tant de questions,
Tant de péripéties mais tant de découvertes,
Et tournant ainsi la page d'une période experte
De recherche, d'enseignement et de transmission.

Cette thèse s'achève, grâce à l'écoute et au soutien
De ma direction et de mon équipe entière,
Et m'ayant ainsi permis d'apporter ma pierre
A l'édifice du savoir, pour le bien des miens.

Je dédie cette étude, fabuleuse expérience,
Désirant partager et faire avancer la science,
A toutes les personnes qui m'ont tant éveillé.

A tout ces bons instants passés dans cette ville,
A tout ce que je dois à mes parents, ma famille,
Aux collègues et amis qui m'ont tant partagé.

Résumé

Le sujet s'intitule : **"Evaluation d'applications de paiement sur carte à puce."**

Cette thèse traite des méthodes d'évaluation de haut niveau, proche des applications. Une méthode d'évaluation d'application sur cartes à puces, complémentaire aux méthodes existantes, a été définie durant de la thèse. La méthode sur laquelle je travaille est une méthode qui mêle observation de la communication et détection de violation de propriétés. Le but est de détecter les anomalies présentes sur cartes à puce (et plus précisemment sur l'application de la carte à puce) et de fournir une documentation accrue sur cette erreur ainsi que les raisons déclencheurs de cette erreur. Avec l'utilisation de propriétés, nous ne stressons pas l'application et nous n'avons pas la difficulté de devoir ordonner les tests pour constituer les suites de tests. Nous allons donc savoir à la volée si une application possède une erreur d'implémentation. L'utilisateur de l'outil va configurer un ensemble de propriétés correspondant aux comportements attendus de l'application qu'il va vérifier, localement ou globalement, durant la phase de test par exemple.

Approches :

La première approche consiste à observer l'évolution de l'application carte par rapport à la machine d'état fournie par les spécifications. Nous observons alors le comportement global de l'application et pouvons détecter la présence d'une erreur. La seconde approche consiste à étudier l'évolution de l'application localement. Un observateur prend en entrée une collection de propriétés et une communication APDU et nous obtenons en sortie une liste de propriétés violées ou non. L'apport de la méthode par rapport au test est donc une documentation accrue et une vérification partielle du comportement sans devoir construire entièrement le comportement global.

Génération de l'oracle :

Afin de connaître la conformité du comportement de l'application carte vis à vis de la théorie (spécifications), la première étape est la génération de l'oracle, élément de référence lors d'une activité de vérification et validation. Dans un premier temps, une approche triviale a été traitée par la génération de la totalité des comportements possibles de l'application. Cependant, nous nous sommes vite orientés vers une technique plus intelligente permettant de cibler des comportements à vérifier plus intéressants pour notre étude. Pour ce faire, nous avons travaillé sur une méthode de génération basée sur un algorithme génétique prenant en entrée un ensemble de logs de transaction qui permet de générer automatiquement un ensemble de propriétés (c'est à dire un ensemble de comportements locaux et attendus de l'application carte).

Résultats :

La méthode de vérification est developpée grâce au framework WSCT. Deux plugins ont été créés et permettent de communiquer avec l'application carte mais aussi d'observer et détecter une anomalie dans le comportement de l'application carte. Nous avons utilisé une applet JavaCard developpée au laboratoire afin de tester la faisabilité de la méthode. De plus, l'interface graphique a été travaillée afin d'être accessible et compréhensible.

Usages :

Cette méthodologie d'évaluation d'application carte permet une utilisation dans différents contextes :

- lors de la phase de test, la méthodologie peut être utilisée en parallèle de la méthode utilisée par l'entreprise de certification (par exemple, du test) afin d'obtenir plus d'informations concernant une anomalie et avoir un rapport d'évaluation de meilleure qualité (plus complet et plus sûr) ;
- lors du développement de l'application, l'utilisation de cette méthodologie permet un guidage du développement. On peut ainsi l'utiliser dans le cadre d'un enseignement du développement JavaCard par un enseignant. Cet outil permet alors de traiter le développement JavaCard, la sécurité des applications carte mais aussi leur conformité vis à vis des spécifications.

mots clés : Validation, carte à puce, évaluation, sécurité, javacard, propriétés temporelles, observation, détection, génération.

Summary

This thesis is entitled : **"Evaluation of payment applications on smart cards."**

This thesis deals with high-level evaluation of applications in smartcards. An evaluation method of applications on smart card, complementary to existing methods, was defined during this PhD thesis. The proposed method combines observation of the communication and detection of properties' violation. The goal is to detect anomalies on smart cards (and more precisely on its implementation) and provide a better documentation on this error and on the reasons that triggered this error. With the use of properties, we do not put application under stress and also we do not have the difficulty of ordering the tests to establish the test suites. We can know on the fly if an application has an error of implementation. The user of the tool configures a set of properties corresponding to the expected behavior of the application will check, locally or globally, during the test phase for example.

Approaches :

The first approach is to observe the evolution of the application from the state machine provided by the specification. We then verify the global behavior of the application and can detect the presence of an error. The second approach is to study locally the evolution of the application. An observer takes as input a collection of properties and APDU communications and we get as output a list of violated properties. The contribution of the method compared to the test is a increased documentation and the verification of a partial behavior without having to fully build the global behavior.

Generation of the oracle

To ascertain compliance of the behavior of the card application with the theory (specifications), the first step is the generation of the oracle, reference used during

verification and validation activity. Initially, a trivial approach has been processed by the generation of all possible behaviors of the application. However, we quickly directed to a smarter technique to target the most interesting behaviors to check for our study. To do this, we worked on a generation method based on a genetic algorithm taking as input a set of transaction logs to automatically generate a set of properties (i.e. a set of local and expected behaviors of the applications).

Results :

The evaluation methodology is developped through the WSCT framework. Two plugins were created and used to communicate with the smart card application, but also to observe and detect an abnormality in the behavior of the application. We used a JavaCard applet developped in the laboratory to test the feasibility of the method. In addition, the GUI has been worked to be accessible and understandable.

Use cases :

This methodology can be used in different contexts :

- During the test phase, the methodology can be used in parallel by the certification firm (e.g., testing) to obtain more information about an anomaly and have a better report of the evaluation (more complete and more precise) ;
- During the development of the application, the use of this methodology allows to guide the development. It can also be used as part of a JavaCard development teaching with an interdisciplinary approach (dealing with development, security and evaluation during the life cycle of a smart card).

Keywords : Validation, Smart card, Evaluation, Safety, Javacard, Temporal properties, Observation, Detection, Oracle.

Table des matières

Introduction

> « Un homme digne de ce nom ne fuit jamais. Fuir, c'est bon pour les robinets. »
>
> Boris Vian

Contexte de réalisation de la thèse

Cette thèse est réalisée grâce à un financement ministériel. J'ai effectué cette étude à Caen au sein de l'Ecole doctorale Structures, informations, matière et matériaux (SIMEM), et le laboratoire Groupe de recherche en informatique, image, automatique et instrumentation (GREYC) dans l'équipe Monétique et Biométrie depuis le 27-09-2012, financée par le Ministère de l'Enseignement supérieur et de la Recherche (MESR). La sécurité des transactions sécurisées fait partie des domaines de recherche de cette équipe. Alors que les transactions sécurisées sont présentes partout dans la vie de tous les jours (transport, paiement, identification), il est nécessaire d'étudier et d'enrichir la sécurité de ces transactions. Ma thèse s'intitule "Evaluation d'applications de paiement sur carte à puce".

Motivation

En 1974, un premier brevet est déposé par Roland Moreno concernant un "procédé pour la mémorisation de données et la commande correspondante de machines électroniques", plus couramment appelé carte à puce [1]. Depuis, ce système a été largement déployé et a évolué, notamment avec l'apparition de standards. Une carte à puce est actuellement composée de deux parties : électronique et logicielle, chacune responsable de la sécurité et de la mise en oeuvre des fonctionnalités d'usage. C'est en 2004 que les spécifications EMV (Europay Mastercard Visa) sont déployées en France,

ayant pour objectif de garantir un haut niveau de sécurité ainsi que l'intéropérabilité des systèmes pour le paiement. Selon EMVCo, le nombre de cartes à puce EMV utilisées dans le monde s'élevait à 3,4 milliard en 2015 et ce chiffre augmente encore [2]. Le déploiement de masse et la large disponibilité impliquent l'apparition de risques sécuritaires et donc le besoin pour les fabricants d'avoir une longueur d'avance sur les attaquants. Avant d'être utilisée dans la vie de tous les jours, une carte à puce doit être vérifiée vis à vis d'exigences fonctionnelles et sécuritaires. Le problème est qu'il est difficile pour une campagne de tests intensifs de retrouver la raison d'une anomalie de fonctionnement. En effet, le déclenchement de l'erreur peut être générée avant la détection du dysfonctionnement. Différentes solutions de validation et vérification d'applications embarquées sur une carte existent d'un point de vue opérationnel et académique mais la complexité des spécifications et les différentes implémentations possibles proposées par les développeurs peuvent être la source d'erreurs.

Objectifs

Durant le cycle de vie de la carte à puce, le système, à la fois logiciel et matériel, va être testé, vérifié et validé. Il peut s'agir de techniques de vérification durant la conception afin d'apporter une aide au prototypage, durant la phase de certification afin de garantir le respect fonctionnel ou encore durant l'utilisation de la carte dans la vie de tous les jours pour garantir la sécurité de la transaction à un utilisateur. Nous verrons par la suite que ces techniques peuvent être caractérisées de méthodes dîtes de boîte blanche (code source connu) ou boîte noire (aucune connaissance hormis les entrées à fournir). Dans cette thèse, nous nous concentrons sur la communication entre la carte à puce et le terminal et nous adoptons alors une analyse en boîte noire permettant de détecter et d'analyser des failles fonctionnelles et sécuritaires de l'application carte à partir de l'observation de cette communication. L'objectif principal est de s'assurer de la validation d'une application de paiement par rapport aux spécifications la définissant et ceci en détectant les possibles erreurs d'implémentation. L'originalité est d'utiliser des outils d'analyse formelle sur une méthode de détection à la volée, c'est à dire que cette méthode utilise une communication existante entre un terminal et une carte à puce, correspondant par exemple à une transaction nominale ou une transaction de test.

Principales contributions de la thèse

Cette thèse se situe dans la problématique de l'évaluation des applications sur carte à puce permettant des transactions électroniques sécurisées comme le paiement, la fidélité, le transport ou le contrôle d'accès. Le marché mondial des cartes à puces est conséquent avec plus de 8 milliards de cartes à puces dans le monde en 2015 [3]. L'enjeu est pour les fournisseurs de proposer des cartes à puces de qualité avec des applications sécurisées. L'usage de cartes à puces dans le cadre de transactions sécurisées fait intervenir de nombreux acteurs directs (émetteur, porteur, accepteur et acquéreur) et indirects (entreprise de certification, laboratoire de test, chercheurs académiques).

Au cours de cette thèse, nous nous attachons à bien positionner la problèmatique de l'évaluation d'applications en termes de méthodes de détection d'anomalie. Après avoir traité l'état de l'art des méthodes d'analyses d'une application carte, une méthodologie est proposée pour détecter un comportement anormal d'une application à partir de la communication entrante et sortante de l'application cible. Nous définissons un langage pour définir les propriétés, comportements théoriques de l'application cible, développons une plateforme d'observation développée avec le framework WSCT afin de prouver l'efficacité de notre méthode. L'observation globale permet de savoir si l'application est correcte d'un point de vue fonctionnel. L'observation de comportements locaux nous permet d'avoir une documentation accrue sur la présence d'une anomalie mais aussi sur les raisons qui ont déclenchées cette anomalie. Cette preuve de concept est réalisée pour une application de paiement mais le principe peut être appliqué à toute application carte.

Nous traitons dans un second temps, la question de la génération de l'oracle d'une méthode d'évaluation. Afin de détecter un mauvais comportement, il va falloir connaître tous les comportements possibles d'une application. Nous définissons donc un language permettant de représenter les comportements globaux mais aussi locaux afin de savoir si cette application suit son évolution attendue. Nous définissons alors une plateforme de génération de propriétés automatisée et optimisée par l'usage d'algorithmes génétiques.

Nous avons ensuite réfléchi à l'usage de cette méthode à différentes phases du cycle de vie de la carte à puce (phase de développement, phase de test ou phase d'utilisation). La preuve de concept principale est mise au point sur une application de type Porte Monnaie Electronique (PME), dans un premier temps, dans le contexte

de développement et de sécurisation de cette application. Dans un second temps, son intérêt est illustré dans le cadre de l'enseignement du développement d'applications JavaCard et de la sécurité associée.

Nous proposons une approche pragmatique originale au vu de la littérature et applicable dans le cadre industriel. C'est pourquoi cette étude est pensée selon trois axes : l'intégration dans une plateforme existante, afin de disposer d'un outil déployable en milieu industriel, la définition d'un langage de propriétés simple pour les non-initiés et une focalisation sur la souplesse d'utilisation plutôt que sur la complétude formelle.

Organisation du manuscrit

Le manuscrit est organisé de la façon suivante :

- **Le chapitre 2** positionne le contexte en donnant les notions les plus importantes pour appréhender les contributions de la thèse.
- **Le chapitre 3** présente la méthodologie générale proposée de vérification d'une application carte vis à vis d'exigences fonctionnelles et sécuritaires. Dans ce chapitre, nous supposons connu l'oracle, c'est à dire l'ensemble des propriétés nécessaires à l'évaluation d'une application sur carte.
- **Le chapitre 4** présente la génération automatique de l'oracle d'une application carte et différents cas d'usage de la méthodologie proposée..

Chapitre 1

Positionnement du problème

Ce chapitre présente les définitions permet d'appréhender la problématique ainsi que le périmètre de l'étude. Suite à ces rappels, nous définissons clairement les objectifs de cette thèse.

Sommaire

Cette partie a pour objectif de donner au lecteur les notions les plus importantes pour appréhender la problématique de la thèse. Nous allons donc définir les termes majeurs et introduire les spécifications sur lesquelles est basée l'étude. Nous définissons le cadre de l'étude et les enjeux de cette thèse en considérant à la fois les aspects académiques et industriels. Plus les exigences sécuritaires sont importantes, plus les attaques sont élaborées. Nous verrons à quels moments du cycle de vie de la carte à puce et sous quelles formes les processus de sécurité sont mis en place afin d'obtenir un système le plus sécurisé possible. Ensuite, nous traitons la question de l'analyse d'applications en vue de la certification du système.

1.1 Contexte de l'étude

1.1.1 Une transaction électronique

Une transaction électronique est un échange d'informations dématérialisées entre deux entités (individus ou organisations) via des systèmes informatiques [4]. La figure 1.1 illustre une transaction électronique dans les usages quotidiens.

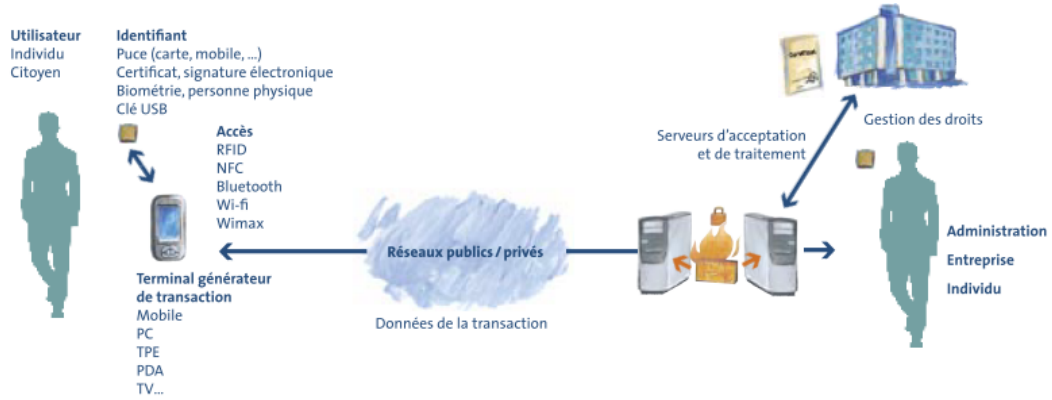

FIGURE 1.1 – Le périmètre de la chaîne d'une transaction électronique sécurisée (source : Pôle Transactions Électroniques Sécurisées [4])

Une transaction électronique peut se décomposer en plusieurs transactions qui sont chacune un ensemble cohérent d'informations au travers d'un réseau. Une transaction peut être la validation d'une information sur une carte à puce (contrôle d'accès), le demande du solde (crédit mobile, solde bancaire, ...) ou encore un paiement [5]. Les technologies employées pour sécuriser la transaction sont nombreuses (cryptographie, réseau, biométrie, firewalls, ...) dont un objet important est l'élément sécurisé.

1.1.2 La monétique

Apparu dès les années 1980, le mot monétique vient de la contraction des termes moné(taire) et (informa)tique et est défini dans l'ouvrage [6]. Il désigne l'ensemble des techniques électroniques, informatiques et télématiques permettant d'effectuer des transactions et des transferts de fonds. A l'origine, ce terme a été créé pour désigner l'ensemble des activités liées à la carte de paiement bancaire. Avec l'introduction de la carte à puce, le terme monétique a été utilisé pour désigner les activités de paiement par carte et les technologies comparables à base de carte (cartes téléphoniques, cartes billettiques, cartes prépayées, ...). La dématérialisation a permis d'inclure la notion de carte virtuelle (recharge téléphonique, e-carte Bleue). Aujourd'hui, l'ensemble du domaine est désigné par l'appellation Transactions Electroniques Sécurisées (TES).

Il recouvre les technologies liées à la carte, aux moyens de paiement, à l'identification numérique, à l'e-santé et à l'e-administration [7].

Depuis 1990, on observe une évolution du terme Monétique et du métier de monéticien. Une définition de son périmètre basée sur la notion de carte à puce et de traitement des transactions électroniques associées est plus juste et en adéquation avec l'évolution de ce domaine. Nous nous focalisons dans cette étude sur le paiement par carte à puce.

La carte de paiement est un moyen de paiement présenté sous forme de carte plastique, équipée d'une bande magnétique et éventuellement d'une puce électronique. Il existe plusieurs sortes de cartes, en fonction de leur vocation :

- La carte de débit qui permet l'utilisation d'argent présent sur le compte du porteur
- La carte de crédit pour des paiements impliquant un taux d'intérêt (débit en plusieurs fois moyennant un crédit financier automatique)
- Le Porte Monnaie Electronique (PME) permettant de charger une carte afin d'effectuer des paiements

Le paiement par carte de paiement met donc en relation plusieurs acteurs comme le montre la figure 1.2 :

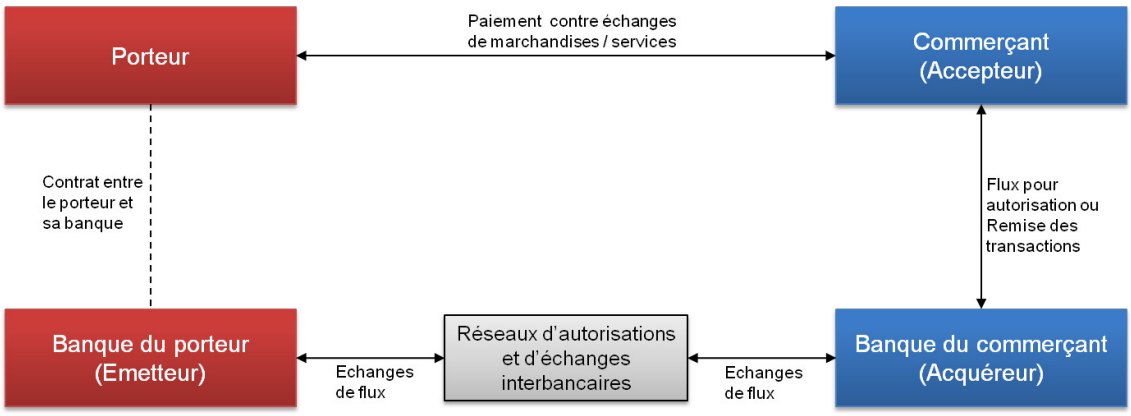

FIGURE 1.2 – Schéma à quatre coins représentant les acteurs pour le paiement (source : Comprendrelespaiements.com [8])

- Un client (porteur) qui possède une carte de paiement
- Un commerçant (accepteur) équipé d'un terminal de paiement
- Etablissement financier du commerçant (accepteur) sur lequel est connecté le terminal

– Etablissement financier du client (émetteur) qui va répondre à la demande d'autorisation online

La transaction de paiement peut s'effectuer soit en ligne soit hors ligne. Dans le premier cas, le réseau interbancaire est utilisé, ce qui permet la communication entre les différentes entités et l'acceptation du paiement (si la transaction est légitime et autorisée). Cependant, la demande d'autorisation n'est pas obligatoire. Il se peut que la transaction se fasse hors ligne. Dans ce cas, il est nécessaire d'avoir une confiance totale dans l'implémentation des spécifications qui régissent la transaction, c'est-à-dire l'application de paiement. Un distributeur de billet est un appareil capable d'accepter une carte de paiement et de distribuer de l'argent. Dans ce cas, la transaction est obligatoirement en ligne. De plus en plus, les transactions de type paiement sont possibles à distance. On parle donc de transactions avec ou sans contact. Dans le cas du paiement mobile, le paiement est réalisé avec un élément sécurisé sans contact.

1.1.3 Les spécifications et certifications

Des spécifications définissent les aspects fonctionnels d'une application et garantissent des exigences de sécurité. Durant les étapes de création de la carte à puce, la notion de sécurité est traitée. Un produit dit certifié a généralement été testé non seulement durant la phase de développement par le constructeur mais aussi par un laboratoire de certification afin de garantir son bon fonctionnement vis à vis des spécifications. Ce processus de certification permet de reconnaître un niveau de sécurité par un laboratoire externe à l'entreprise à l'origine du produit. Pour le paiement de type Europay Mastercard Visa (EMV), EMVCo fournit le document appelé "Security Evaluation Process" à ses membres [9]. Il assure une base de sécurité robuste pour les cartes à puces et produits associés. Pour résumer, EMVCo fournit des documents afin de guider le test et l'évaluation de cartes à puces mais aussi une liste des laboratoires approuvés par EMVCo [10].

Plus généralement, l'évaluation de plus haut niveau est permise par la certification dite tierce partie, "qui permet à un client de s'assurer par l'intervention d'un professionnel indépendant, compétent et contrôlé, appelé organisme certificateur, de la conformité d'un produit à un cahier des charges ou à une spécification technique. La certification tierce partie apporte au client la confirmation indépendante et impartiale qu'un produit répond à un cahier des charges ou à des spécifications techniques publiées" [11]. En France, la certification s'appuie sur des travaux effectués par des centres d'Evaluation de la Sécurité des Technologies de l'Information (CESTI)

acrédités par le Comité Français d'Acréditation (COFRAC) selon la norme ISO/IEC 17025 [12]. Les CESTI mènent ces évaluations selon les spécifications de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) afin d'émettre des certificats attestant que le produit certifé est conforme à une spécification technique appelée cible de sécurité.

L'évaluation selon les Critères Communs [13] permet quant à elle de certifier, et ainsi délivrer un certificat, un produit selon des niveaux d'assurance de sécurité Evaluation Assurance Level (EAL) très variés, allant de EAL1 (niveau d'attaquant faible) à EAL7 (niveau d'attaquant élevé). Les certificats émis par l'ANSSI et basés sur les Critères Communs peuvent bénéficier d'une reconnaissance internationale. On peut aussi citer le CCRA (Common Criteria Recognition Agreement) signé par 25 membres permettant la reconnaissance au niveau EAL2 (niveau d'attaquant basique) et l'accord européen SOG-I5 [14] regroupant actuellement 10 membres dont les CESTI et permettant la reconnaissance des certificats jusqu'à EAL4, voire pour certains domaines techniques particuliers jusqu'à EAL7.

1.2 La sécurité des applications sur carte

FIGURE 1.3 – Le contexte de l'étude

Dans cette partie, nous présentons toutes les notions liées à la sécurité des applications sur carte en décrivant les aspects matériels, les standards et les spécifications afin de positionner les travaux de cette thèse (illustration dans la figure 1.3).

1.2.1 Elément sécurisé

Un élément sécurisé ou "Secure Element"(SE) est une plate-forme "inviolable" capable d'héberger une ou plusieurs applications en toute sécurité ainsi que leurs

données confidentielles et cryptographiques (par exemple, gestion des clés). L'authentification, l'identification, la signature et la gestion du PIN sont au coeur du système. Le SE contrôle les interactions entre les sources de confiance (un établissement financier), l'application de confiance (une application de paiement) stockée sur la SE et des tiers (un commerçant). Le domaine sécurisé protège les informations d'identification de l'utilisateur et traite la transaction de paiement dans un environnement de confiance afin d'assurer la sécurité des données de l'utilisateur. La figure 1.4 fournit un zoom sur le SE de la carte à puce.

FIGURE 1.4 – Zoom sur l'élément sécurisé (Source : cartes-america [15])

Il y a trois différents types de SE :
- le SE pour le mobile : Universal Integrated Circuit Card (UICC)
- le SE embarqué
- le SE sous forme de microSD

Chaque SE répond à des besoins différents du marché. La puce à microcircuit qui réside dans les cartes de paiement a été adaptée pour répondre aux besoins du monde mobile. Avec de multiples applications actuellement stockées et leurs processus exécutés dans le même dispositif, il est essentiel d'être en mesure d'accueillir des applications de confiance dans un environnement sécurisé. Au niveau logiciel, les SEs de type embarqué sont souvent basés sur les systèmes d'exploitation ouverts comme JavaCard[16] et MULTOS et répondent aux spécifications GlobalPlatform [17] pour le chargement sécurisé des applications.

De nos jours, l'utilisation de la carte à puce ne se réduit pas au paiement et les domaines sont variés (voir Figure 1.5). Les applications de la monétique sont nombreuses et nous avons la possibilité d'utiliser les suivantes :
- Application de paiement Europay Mastercard Visa (EMV)

– Application de porte monnaie électronique
– Application de ticketing
– Application d'identification
– ...

FIGURE 1.5 – Domaines d'utilisation des cartes à puce

1.2.2 Norme ISO/IEC 7816

Les données échangées au niveau applicatif à partir d'un terminal vers une carte à puce sont appelées commandes Application Protocol Data Unit (C-APDU). Les données échangées à partir d'une carte à puce vers un terminal sont les réponses Application Protocol Data Unit (R-APDU). La carte à puce va prendre le rôle esclave dans le dialogue. La communication est définie dans la norme ISO/IEC 7816 [18]. Dans le cas d'une transaction sans contact, on ajoute la norme dédiée ISO/IEC 14443 [19] pour les aspects physiques, les aspects applicatifs restent identiques. Comme présenté sur la figure 1.6, une commande envoyée par le terminal est obligatoirement suivie d'une réponse de la part de la carte.

FIGURE 1.6 – Un couple commande/réponse APDU

Sur la figure 1.7, nous pouvons observer la structure d'une commande et d'une réponse APDU d'après l'ISO/IEC 7816 [18]. Les champs de la commande sont les suivants : l'octet "Class" (CLA) indique le type de la commande, l'octet "Instruction" (INS) la commande spécifique, les paramètres (P1 et P2) précisent la commande, l'octet "Length of UDC" (LC) la longueur des données inclues dans la commande (UDC) et l'octet "Length of UDR" (LE) la longueur des données de la réponse

attendue. Ces trois derniers champs sont optionnels. Si le champs LE est présent, la réponse contiendra un UDR, données de longueur LE ainsi que deux champs obligatoires constituant le ”Status Word” (SW1 et SW2), il s'agit du statut de traitement de la commande par la carte.

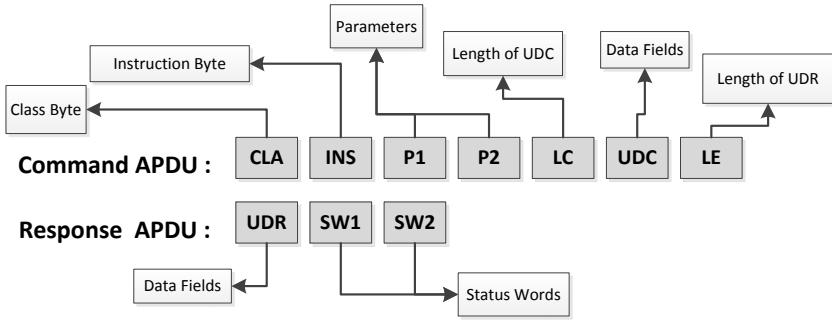

FIGURE 1.7 – Structure de la communication APDU

Chaque commande possède sa propre configuration (les valeurs des paramètres, l'utilisation de données ou l'attente de données dans la réponse). Les commandes principales définies par cette norme sont les suivants (l'octet INS est suivi de la description de la commande) :

- 0E : Erase Binary
- 20 : Verify
- 70 : Manage Channel
- 82 : External Authenticate
- 84 : Get Challenge
- 88 : Internal Authenticate
- A4 : Select File
- B0 : Read Binary
- B2 : Read Record(s)
- C0 : Get Response
- C2 : Envelope
- CA : Get Data
- D0 : Write Binary
- D2 : Write Record
- DA : Put Data
- DC : Update Record
- E2 : Append Record

Les réponses possibles sont visibles dans la figure 1.8.

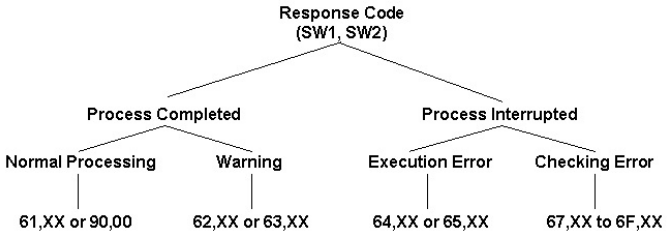

FIGURE 1.8 – SWs possibles (Source : Oracle [20])

1.2.3 Communication PC/SC :

Les spécifications "Personal Computer / Smart Smart" (PC/SC)[21] ont été créées afin de permettre l'interopérabilité des communications entre une carte à puce et un ordinateur. Le group PC/SC, établi en 1996, est à l'origine de ces spécifications (la dernière version datant de Juin 2013). Grâce à celles-ci, nous pouvons créer une application en communiquant avec n'importe quel lecteur PC/SC. Une bibliothèque de type "Dynamic Link Library" (DLL), ensemble d' "Application Programming Interface" (API) est fournie par Microsoft. Cette bibliothèque propose des fonctions utiles pour la création d'une application nécessitant la connexion à une carte à puce sur un PC. Lorsque qu'une application a besoin de ce fichier, l'ensemble de la bibliothèque est chargé dans la mémoire. Pour être exact, winscard.dll est l'implémentation standard pour les environnements Windows et pcsclite est l'implémentation open source pour les systèmes linux.

1.2.4 Les spécifications EMV

Europay Mastercard Visa (EMV) est un ensemble de spécifications pour la sécurité des paiements par cartes à puce. C'est l'organisme EMVco qui s'occupe de ce standard (MasterCard, Visa, JCB et American Express). Les points importants de ces spécifications sont l'interopérabilité internationale, la vérification et déchiffrement de la clé personnelle (PIN) par la puce ainsi que le multi-applicatif. Plusieurs applications peuvent alors cohabiter sur la même carte à puce. Précisons toutefois que EMV couvre la transaction à partir de l'émetteur juqu'au terminal en passant par l'acquéreur, mais nous nous focalisons sur la relation entre la carte et le terminal seulement dans la suite de l'étude.

Les spécifications EMV de base sont constituées de quatre livres disponibles sur internet librement :

– Livre 1 : définition des caractéristiques mécaniques et électriques, réponse à la

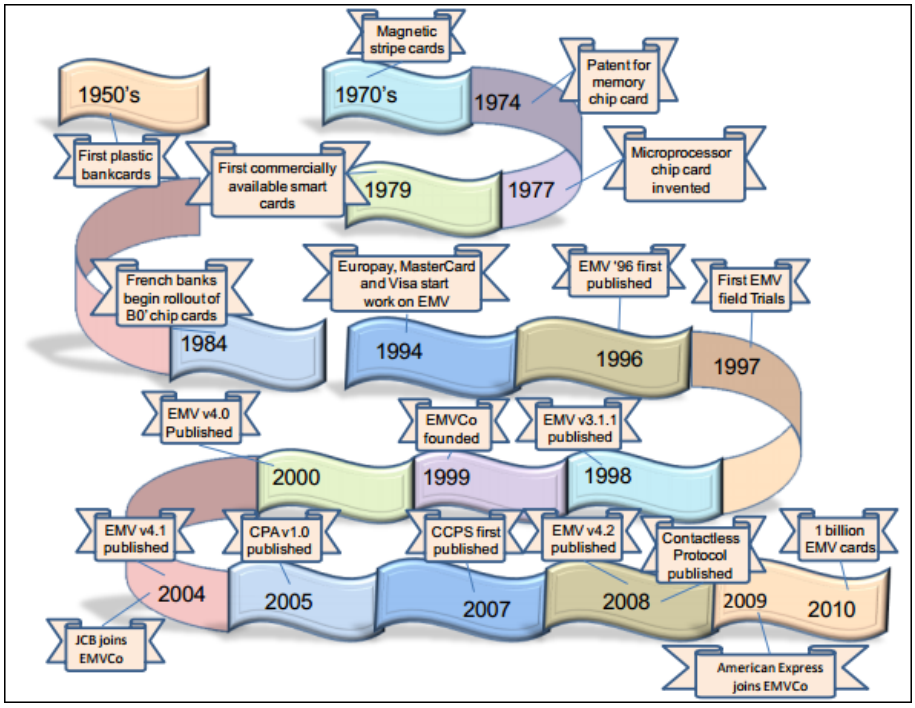

FIGURE 1.9 – Évolution de EMV (source : EMVCo [22])

remise à zéro, description du protocole de transaction et sélection d'applications [23],
- Livre 2 : authentification des données : statique et dynamique, chiffrement du code PIN, intégrité, confidentialité, mécanismes de sécurité et algorithmes cryptographiques [24],
- Livre 3 : une partie concerne les commandes et les données, une seconde définit le flux transactionnel [25],
- Livre 4 : besoins fonctionnels et caractéristiques physiques, gestion des données et du logiciel et interfaces utilisées [26].

La transaction est illustrée schématiquement par la figure 1.10. Voici une description concise de chaque étape régissant une transaction EMV (utilisée dans le contexte de paiement face à face avec une carte à puce à contact) :

1. **Initialisation**
   - **Application selection** : soit l'application est sélectionnée par l'"Application identifier" (AID) par une commande SELECT, soit le "Payment System Environnement" (PSE) est sélectionné et l'AID est retrouvé dans les données pointées par le "Short File Identifier" (SFI) retourné par la commande SELECT.

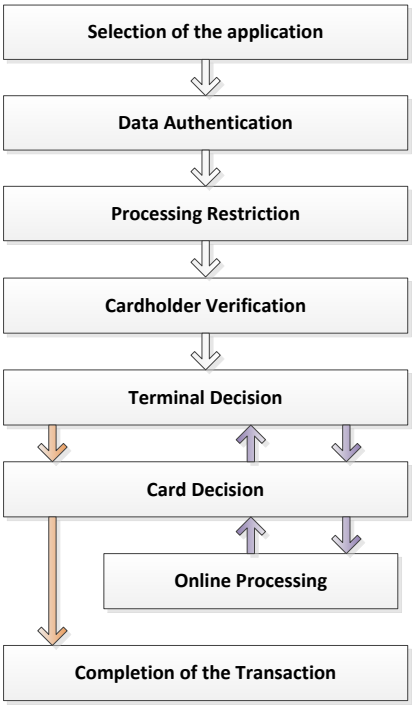

FIGURE 1.10 – Étapes EMV

- **Initiate application processing** : les valeurs des "Application Interchange Profile" (AIP) et "Application File Locator" AFL sont retrouvées grâce au "Processing Options Data Object List" (PDOL) ou bien par la valeur par défaut 83 00 dans la commande GET PROCESSING OPTION.
- **Read application data** : on peut alors lire les données dans les enregistrements pointés par l'AFL. Les données sont alors récupérées pour être utilisées plus tard dans la transaction.

2. **Transaction Analysis**
   - **Offline data authentication :** l'authentification la plus sécurisée est effectuée si supportée par la carte, "Combined Data Authentification" (CDA), avec un repli possible sur la "Dynamic Data Authentification" (DDA) ou enfin "Static Data Authentification" (SDA).
   - **Processing restrictions :** cette étape détermine le degré de comptabilité de l'application dans le terminal avec l'application sur la carte.
   - **Card Holder Verification :** le porteur est authentifié grâce à un code PIN en clair ou chiffré.
   - **Terminal Risk Management :** des mesures de sécurité sont effectuées, vérification du seuil limite, transaction pouvant être envoyée en ligne par le

terminal et vérification d'autres compteurs.

3. **Decision**
   - **Terminal Action Analysis :** en envoyant la commande GENERATE AC, le terminal demande la transaction en ligne ou non. Le TVR contient les données de sa décision. Réponse possible : AAC, TC ou ARQC.
   - **Card Action Analysis :** la carte répond alors avec sa propre décision.
   - **Script processing :** permet à l'émetteur de modifier des données sur la carte.
   - **Completion :** termine la transaction.

1.2.5 Spécifications propriétaires

Chaque émetteur de carte à puce complète les spécifications EMV par ses propres spécifications apportant des éléments supplémentaires pour développer une application. Mastercard est à l'origine des spécifications M/CHIP [27], définissant les applications cartes de paiement Mastercard conformes EMV. L'application M/CHIP peut prendre plusieurs états. L'état de l'application évolue par la réception de commandes APDU et l'émission des réponses associées.

L'application est illustrée par une machine d'états (que nous ne pouvons pas donner entièrement, les spécifications propriétaires étant confidentielles) dont les états possibles sont :

- "idle" : l'application n'est pas encore selectionnée
- "selected" : l'application est selectionnée
- "initiated" : la transaction est initialisée
- "online" : l'application demande une connexion à l'émetteur
- "script" : l'application est prête à recevoir une commande de script

Nous pouvons voir dans les spécifications MasterCard, conformes au standard EMV, que l'application de paiement ne peut évoluer qu'en recevant et répondant à une série de commandes APDU. Nous ne pouvons pas considérer l'acceptation d'une seule commande sans prendre en compte l'état courant théorique ainsi que les communications précédentes. Chaque constructeur définit son cahier des charges afin d'exposer le plus clairement possible les fonctionnalités désirées. Des commandes et des réponses spécifiques peuvent être ajoutées aux définitions données par la norme ISO/IEC 7816. Les développeurs vont, à partir de ces spécifications, standards et normes, implémenter l'application grâce à un ensemble de technologies.

1.2.6 JavaCard

La génération actuelle de carte à puce est basée sur le premier langage orienté objtet adapté aux cartes à puce. Il s'agit d'un langage créé par les ingénieurs de Schlumberger au Texas. En février 1997, Schlumberger et Gemplus fondent Java Card Forum [28], ils sont rejoints par les producteurs de cartes à puce comme Bull ou Giesecked & Devrient. En octobre 2000, plus de 40 entreprises ont acquis la licence d'exploitation Java Card. Actuellement, la plus récente version Java card 3.0.1 date de mai 2009.

Les principaux avantages de ce type de la technologie sont :

- Développement facile
- Interopérabilité des applications
- Isolation entre les applications
- Sécurité
- Indépendance au hardware
- La gestion de multiples applications

Ces avantages sont permis par le langage Java, l'environnement de développement Java et des mécanismes tels que l'impossibilité de construire des pointeurs, un firewall et l'encapsulation de la complexité fondamentale du système des cartes à puce ([29] et [30]).

JavaCard est un environnement d'exécution déstiné aux cartes à puce permettant d'écrire et d'exécuter des programmes appelés Applet avec l'approche orientée objet du langage Java. L'architecture est présentée dans la figure 1.11.

1.2.7 GlobalPlatform

GlobalPlatform [17] est un consortium créé en 1999 par les grandes entreprises des secteurs du paiement et des télécommunications ainsi que les structures gouvernementales. L'infrastructure globale est destinée à l'implémentation des cartes à puce commune à tous les secteurs. Les spécifications GlobalPlatform (anciennement Visa Open Plateform) visent à gérer les cartes de façon indépendante du matériel, des vendeurs et des applications. Elles répondent efficacement aux problèmatiques de la gestion du multi-applicatif : chargement sécurisé des applications, gestion du contenu et cycle de vie.

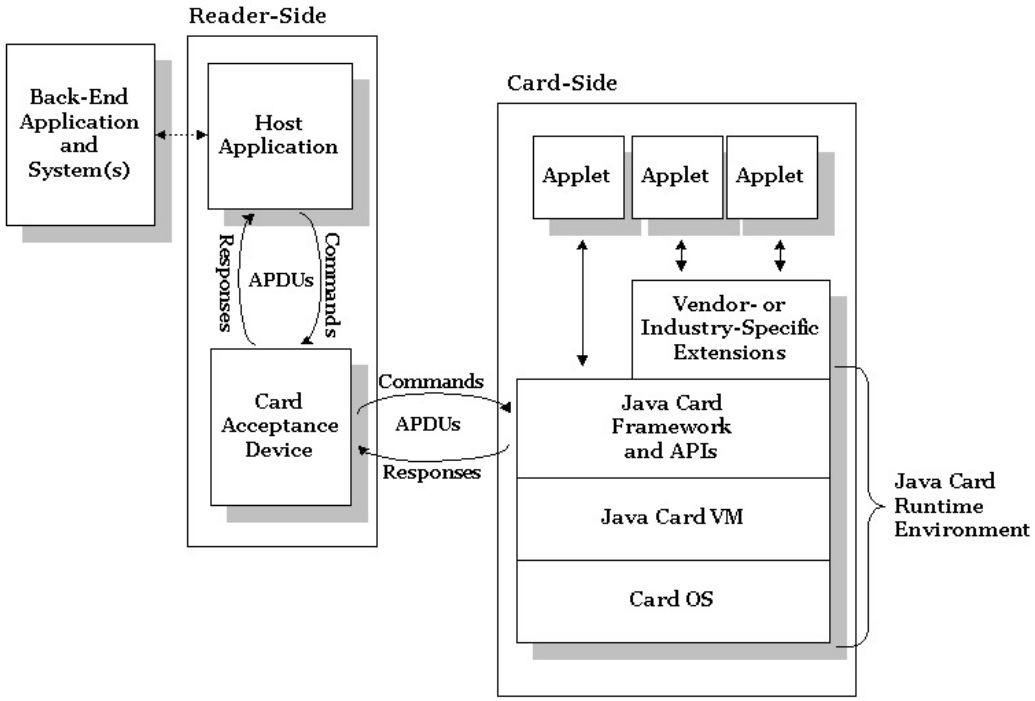

FIGURE 1.11 – Architecture d'une application JavaCard (source : Oracle [31])

1.2.8 Discussion

La transaction de paiement peut donc être vue selon deux angles. D'une part, le terminal de paiement initie et effectue la transaction en communiquant des informations à la carte et en analysant les réponses. La transaction est donc composée des étapes EMV vues précédement. D'autre part, nous pouvons voir la transaction de paiement du point de vue carte. Ce sont donc les spécifications propriétaires et la machine d'état définie par MasterCard pour l'application carte M/CHIP qui nous permet d'appréhender le sujet selon les deux points de vue. Ces documents permettent aussi à l'entreprise de certification d'effectuer une analyse sur le fonctionnement de l'application. Le paiement EMV étant très répandu, une étude sur ce type d'application est cohérente. Cependant, la simplicité de l'application Porte Monnaie Electronique (PME) par rapport au paiement EMV nous permet de travailler sur une preuve de concept plus accessible. Toutes ces applications sont basées sur ISO/IEC 7816 ainsi que des spécifications complémentaires.

1.3 Attaques sur une transaction de paiement

Dans cette partie, nous allons exposer les failles existantes sur l'usage de la carte à puce. Malgré une sécurité importante et un audit sécurité, nous pouvons voir que des attaques sur ce système existent [32]. Certaines attaques sont spécialisées pour une transaction avec ou sans contact physique. Nous pouvons aussi distinguer les attaques invasives (impliquant une destruction partielle ou complète de la carte) et les attaques non invasives.

1.3.1 Attaques invasives ou semi-invasives :

Ce type d'attaque met en péril l'intégrité physique de la puce à micro-circuit. Les attaques présentées ici sont présentées dans [33].

Par injection de fautes :

Cette attaque est une attaque semi-invasive car elle nécessite un accès direct à la carte à puce. Il s'agit de perturber le fonctionnement d'une carte à puce en modifiant une donnée précise. Par exemple, on peut faire varier la tension d'alimentation (power glitch), l'horloge ou encore la température. Une autre méthode consiste à imposer à la puce un laser, un faisceau d'ion, un rayon X, etc. Concrètement, plusieurs techniques ([34] ou [35]) sont connues :

- Application de lumière laser sur le processeur introduit par [36]
- Attaque de Bellcore sur le chiffrement RSA (nommé par les initiales de ses trois inventeurs) comme dans [35]
- Défaillance électronique (glitch) appliqué sur un calcul RSA, historiquement la première méthode [37]
- Attaque de type "Differential fault analysis" (DFA) [37]

La réalisation de cette attaque est donc variable d'un système à l'autre. Il faut effectuer de nombreux essais, de différentes sortes, afin d'obtenir des résultats concluants.

Par conditions anormales Cette attaque varie de l'attaque précédente par la durée de la modification apportée [38]. Ici, on fait fonctionner la carte à puce en dehors de ses conditions opérationnelles. L'attaquant peut modifier la tension ou la fréquence d'alimentation ainsi que la température facilement. Cependant, ces attaques ne semblent pas efficaces car des détecteurs de conditions anormales sont présents dans la carte à puce.

Par rayonnement : Afin de modifier l'exécution des applications d'une carte à puce, on peut aussi utiliser un rayonnement électromagnétique. Cette attaque est semi-invasive car l'attaquant doit accéder à la puce. On applique donc un rayonnement durant l'exécution de l'application (ce qui diffère de l'injection de faute) [39].

Par les composants : Ces attaques [40] sont complètement invasives car nécessitent l'étude d'un composant précis de la carte à puce. On doit donc désassembler la carte à puce et celle-ci ne sera plus utilisable par la suite. Cette première étape se déroule chimiquement ou physiquement. Il existe plusieurs outils possibles en vue de cette attaque [40] :

- Focused Ion Beam (FIB). Cet outil permet d'étudier un composant avec un microscope ainsi que de retirer de la matière à la carte à puce.
- Electron Beam Tester (EBT). Cet outil permet de lire la valeur du potentiel à un endroit donné de la carte.
- Microprobing : accès à la puce.

L'attaquant cherche à trouver les systèmes de sécurité (implémentations, informations contenues dans la puce) présents sur un composant en particulier. Par exemple, il est possible de lire les bits sur le bus mémoire lorsqu'on sait qu'une donnée intéressante y transite.

1.3.2 Attaques non invasives

Ce type d'attaque n'altère pas la puce à micro-circuit mais consiste à observer le fonctionnement d'une application.

Par canaux cachés :

Une attaque sur carte à puce consiste à étudier le fonctionnement du circuit électronique. Plusieurs données peuvent être utilisées telles que la consommation du courant, les émissions électromagnétiques ou encore le temps d'exécution. Les attaques par canaux cachés utilisent les failles matérielles de l'implémentation pour récupérer des informations sur le secret utilisé [41].

On peut ainsi lister plusieurs techniques majeures :

- consommation du courant : Simple Power Attack / Differential Power Attack / High Order Differential Power Analysis avec par exemple [42]
- émissions électromagnétiques : Simple Electromagnetic Attack / Differential Electromagnetic Attack [43]

– le temps d'exécution dans [44]

Skimming : L'attaque dite de " skimming " [45] est une attaque simple à mettre en place, d'où son utilisation importante. Il s'agit de récupérer des informations d'une carte à puce en plaçant un élément entre le lecteur de carte à puce et la carte à puce (à condition d'avoir la possibilité de créer un élément d'écoute d'épaisseur inférieure à 0,1 mm). Cet élément a pour but de capturer les données tel que le numéro de carte. Sur la figure 1.12, on voit cet élément sur la photo en haut à droite. La seconde partie de l'attaque est l'utilisation d'une caméra (voir la flèche noire visible sur la première image de la figure 1.12). On peut ainsi récupérer le code PIN (Personal Identification Number) du porteur et se faire passer pour le porteur de la carte, même si généralement il s'agit uniquement de la lecture et la récupération de la piste.

FIGURE 1.12 – La technique de skimming en images (source : Antiskimmingeye [46])

Yes Card La Yes Card est une carte vierge programmable à l'origine. L'attaquant copie toutes les données statiques d'une carte à puce valide, à une différence près. L'attaquant modifie le retour du test du code PIN pour accepter n'importe quel code PIN. Cependant, lors de l'envoi de la réponse "code PIN accepté", un faux cryptogramme est envoyé par la carte qui est détectable lors d'une transaction en ligne par vérification du cryptogramme par l'émetteur. Cette attaque n'est possible qu'en mode hors ligne.

Attaque en relais Son principe est qu'un attaquant intercepte les communications entre deux entités. Le but de l'attaque est d'usurper l'identité d'un individu et de récupérer la transaction initiale entre les deux entités initialement en communication et de modifier cette transaction, afin par exemple de valider une autre transaction [47].

Attaque de Cambridge Contrairement à l'attaque utilisant une YES CARD, cette attaque est typiquement un man-in-the-middle dans lequel l'intercepteur répond oui à la vérification du PIN sans transmission de la commande à la carte réelle [48]. Cette faille est connue par EMVCo depuis des années et peut être comblé par

l'authentification CDA en cours de déploiement.

FIGURE 1.13 – Illustration du matériel necessaire pour effectuer l'attaque de Cambridge initiale

1.3.3 Transactions sans contact

Les attaques sont tout aussi possibles voire pour certaines facilitées dans un contexte sans contact. En effet, l'attaque dite "écoute illégale" a pour principe l'observation de communication entre le terminal et le lecteur. Dans ce contexte, l'attaquant peut récupérer plus facilement les informations concernant la transaction ou la carte à puce. On peut aussi parler de l'attaque dite "scanning actif" qui permet d'activer une carte sans contact à distance. L'attaque en relais est bien entendu possible et facilitée par l'usage d'une carte sans contact [49]. La figure 1.14 expose cette attaque.

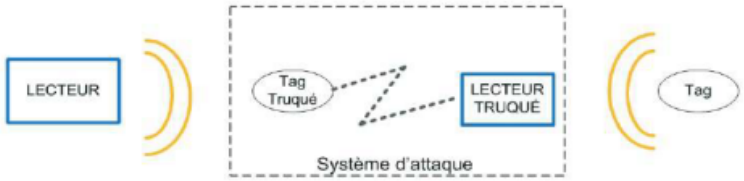

FIGURE 1.14 – Attaque en relais

1.3.4 Récapitulatif :

Le tableau 1.1 récapitule les attaques traitées dans ce document. On discerne les attaques invasives des attaques non invasives mais aussi les attaques avec ou sans contact. Certaines de ces attaques ont la particularité d'être facilement reproductibles comme l'attaque de Cambridge. L'étude de ces attaques est une étape importante dans la recherche de sécurité. En effet, il faut connaître les risques afin de proposer des mesures adaptées.

TABLE 1.1: Récapitulatif des attaques

Attaque	Invasive	Contact	Fonctionnement
Par injection de fautes	Oui	Oui	modification du fonctionnement de la carte à puce en injectant des erreurs
Par conditions anormales	Oui	Oui	modification du fonctionnement en modifiant une condition sur la durée
Par rayonnement	Oui	Oui	On applique un rayonnement sur la carte à puce en fonctionnement
Par les composants	Oui	Oui	isolation des composants pour isoler les éléments de sécurité
Par canaux cachés	Non	Oui	à partir de consommation, temps exécution ou émissions électromagnétiques
White Plastic	Non	Oui	clonage de la partie piste d'une carte, afin de l'utiliser à l'étranger
Skimming	Non	Oui	obtention des informations d'une carte grâce à des éléments extérieurs
Yes Card	Non	Oui	clonage d'une carte à puce avec modification de la vérification du code PIN
Attaque relais	Non	Oui	utilisation d'un terminal modifié et transfert d'une transaction pour la modifier
Cambridge	Non	Oui	éviter la vérification du code PIN pour accepter la transaction sans code PIN
Attaques en relais	Noni	Non	utilisation d'un terminal modifié et transfert d'une transaction pour la modifier
Ecoute illégale	Non	Non	obtention des informations par écoute de la transaction
Scanning Actif	Non	Non	activation d'une carte par l'envoi de commandes pour obtenir des données

1.4 Evaluation d'applications

Afin de garantir le bon fonctionnement et la sécurité d'applications, il est nécessaire les évaluer.

1.4.1 Introduction

Implémenter une application carte n'est pas une tâche facile à réaliser. Par exemple, dans le livre [50], certaines difficultés sont exprimées. Des spécifications imprécises ou des options trop vastes pourraient être à l'origine de problèmes. Par exemple, toutes les variantes possibles d'authentification des données et de la carte peuvent ne pas être entièrement gérées par un terminal.

L'existence d'attaques est une preuve évidente du besoin de sécurité. Cependant, la sécurité idéale n'existant pas, c'est une course perpétuelle entre les constructeurs qui tentent de sécuriser au mieux leurs produits et les attaquants qui conçoivent des attaques de plus en plus complexes. Nous allons voir comment l'évaluation de la carte à puce peut donner un certain niveau de garantie concernant les aspects fonctionnels et sécuritaires de la carte à puce. Les enjeux sécurité sont nombreux : confidentialité des informations, authentification du porteur, intégrité de la transaction, impossibilité de modifier des informations sur la puce, ...

1.4.2 Définitions

Selon ISO 8402[51], la qualité est définie comme l'ensemble des fonctions et caractéristiques d'un produit ou d'un service qui lui confèrent l'aptitude à satisfaire les besoins exprimés ou implicites. Les deux attributs les plus importants de la qualité d'une application sont la robustesse et l'absence de défauts et d'erreurs. On appelle l'évaluation logicielle le fait d'évaluer un logiciel. En ce qui concerne l'évaluation d'une carte à puce, le test est principalement utilisé. Le but du test est toujours de trouver des défauts et des erreurs, le processus de test doit être orienté vers ce but. Cependant, les tests ne peuvent être complets car il n'est jamais vraiment possible de travailler en prenant en compte toutes les combinaisons possibles. A l'instar d'une qualité parfaite et complète, nous obtenons un niveau de qualité acceptable par rapport au niveau de qualité souhaité. Comme les tests ne peuvent jamais tout couvrir, il est difficile de définir un critère d'achèvement pour le développement de tests. La règle de base est que le même effort devrait être consacré à l'élaboration de tests qu'au développement du produit. Le niveau minimum de couverture d'instructions

devrait être de 95% et la couverture des branches de 80% [50].

On peut observer dans certains articles, l'utilisation des termes vérification, validation ou évaluation de manière interchangeable comme ci ces termes avaient la même signification. La norme ISO 8402[51], maintenant obsolète, traitait de la vérification et la validation de systèmes de manière bien distinctes. Ce sont des processus d'évaluation d'un système ou d'un composant. La norme de l'Institute of Electrical and Electronics Engineers (IEEE) [52] nous donne deux définitions, citées dans leur langue originale :

- "verification. (1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Contrast with : validation. (2) Formal proof of program correctness."
- "validation. The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements."

Des documents développent donc ces notions, que ce soit dans le domaine général ou dans un domaine bien précis ([53], [54]). La vérification permet de contrôler la conformité d'un système vis à vis de sa définition, une description formelle. La validation est le processus consistant à déterminer si le système remplit effectivement le but pour lequel il a été conçu. L'évaluation reflète l'acceptation du système par les utilisateurs ainsi que ses performances. On peut résumer ces notions par trois questions :

- Vérification : Est-ce que le système fonctionne correctement ?
- Validation : Est-ce que le système réalise les fonctions attendues ?
- Evaluation : Dans quelle mesure le système peut-il être utilisé par tous ?

Concrètement, la première chose à faire sur un système est de le valider. Puis, il faut le vérifier de manière la plus compète possible (différentes techniques existent). Enfin, il s'agit de donner une évaluation du système. Un système correct du point de vue de la vérification mais inutilisable (trop lourd, pas adapté au public) ne peut pas avoir une réponse positive à l'évaluation [53]. Les méthodes de Vérification et Validation (V&V) permettent de découvrir des problèmes [54]. Cependant, il peut s'agir de différents types de problèmes. Une erreur est un problème de raisonnement de la part des personnes à l'origine de ce système. S'il est inclus dans le système, l'erreur devient une faute. Une défaillance se produit quand le système ne se comporte

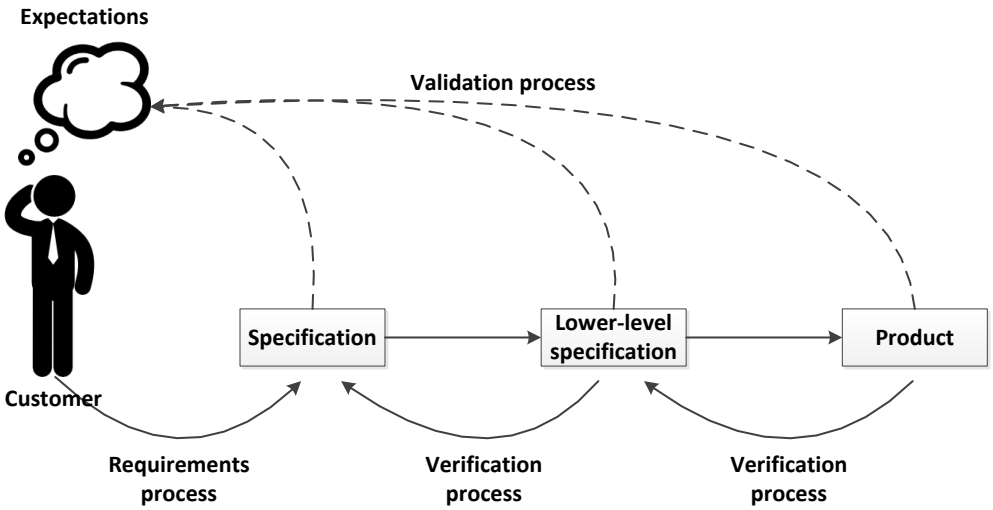

FIGURE 1.15 – Le processus de validation d'un système

pas comme il le devrait. Nous pouvons aussi parler d'anomalie, terme générique pouvant signifier une erreur, une faute ou une défaillance.

On peut récapituler ainsi :
- Erreur : raisonnement incorrect de la part d'une personne.
- Faute : erreur dans un système.
- Défaillance : comportement d'un système incorrect.
- Anomalie : terme générique indiquant la présence d'une erreur, faute ou défaillance.

En effet, il est nécessaire de valider chaque étape et chaque élément d'une carte à puce. Il est irréaliste d'utiliser un composant sécurisé et certifié avec un programme incorrect. Le programme doit également être validé et certifié. La création des critères d'évaluation a permis la certification de produits [55]. La reconnaissance de ces critères permet la reconnaissance mutuelle entre les différentes structures et les différents pays.

Quelques grands principes d'une évaluation sont les suivants :
- Répétabilité : une même évaluation sur un même produit doit donner le même résultat
- Reproductibilité : une même évaluation sur une même cible dans un autre laboratoire doit donner le même résultat
- Impartialité : il ne doit pas y avoir de biais
- Objectivité : l'interprétation du test ou des résultats doivent être sans équivoque

Il existe différents contextes d'évaluation :

- en boîte noire : Cette évaluation consiste à étudier le comportement de l'application en connaissant seulement les entrées et les sorties du système.
- en boîte blanche : Dans ce cas, nous avons accès au code de l'application. Cette connaissance complète nous permet de connaître de manière efficace le déroulement théorique de l'application ou encore d'ajouter des éléments pour le test.
- en boite grise : Nous disposons de quelques informations mais pas l'accès complet à l'application.

Des évaluations peuvent subvenir à différents moments de la mise en place d'une carte à puce.

- Il peut s'agir d'évaluation lors de la phase de développement. Par exemple, la mise en place d'assertions. Afin de continuer le programme, chaque assertion doit être vraie.
- Il peut aussi s'agir d'évaluation après la production de la carte. Une entreprise peut essayer de stresser l'application carte afin de vérifier si une attitude est incorrecte. On parle alors de phase de test.
- Durant la vie de la carte, nous pouvons aussi utiliser un système détectant les anomalies à la volée durant les transactions normales.

1.4.3 Le cas du logiciel

Les notions de vérification et validation logicielles sont développées dans les documents [56] et [57]. Le but principal de la V&V logicielle est de déterminer si le logiciel effectue les fonctions demandées correctement, de s'assurer qu'il n'effectue pas d'opérations non désirées et de fournir des informations sur la qualité et la fiabilité du logiciel. On peut aussi s'assurer que le logiciel est en accord avec les standards non seulement vis-à-vis de ses spécifications mais aussi par rapport aux systèmes en relation avec ce logiciel. On peut ainsi dire que la vérification logicielle apporte la preuve que les sorties de conception d'une phase de développement logiciel sont en accord avec les exigences spécifiées pour cette phase [56]. Ceci vise à l'exactitude du logiciel. Dans un environnement de développement, la vérification logicielle permet de s'assurer qu'une phase de développement est terminée.

La validation logicielle permet de s'assurer que le logiciel est conforme aux besoins des utilisateurs. Selon [57], la validation logicielle peut s'apparenter à une vérification

car elle inclut toutes les activités de vérification et de test menées tout au long du cycle de vie du logiciel. Elle permet une diminution des coûts du logiciel en diminuant les risques d'utilisation ainsi que le besoin de mises à jour. Finalement, dans le développement logiciel, la validation permet de dire que le bon logiciel est créé et la vérification permet de s'assurer qu'il a été créé correctement. Un niveau de confiance, basé sur le niveau de validation, le niveau de vérification et le niveau de test, pourrait alors être définit pour un logiciel. On peut ainsi savoir si ce niveau de confiance est acceptable ou non. EMVco effectue des contrôles sur les cartes et terminaux. Un niveau de sécurité est alors accordé à ces éléments.

1.4.4 Le cycle de vie de l'application carte

Sur la figure 1.16, le cycle de vie simplifié de la carte à puce basé sur [58] est présenté. Pour évaluer et valider le produit, plusieurs tests sont effectués au cours des différentes étapes de sa vie [59]. Tous les aspects de la carte (application, puce, données de personnalisation) doivent être validées, ce qui assure un produit sécurisé. Une fois conçu et validé, la carte est dite certifiée. La certification permet de reconnaître le niveau de sécurité d'un produit par un laboratoire externe.

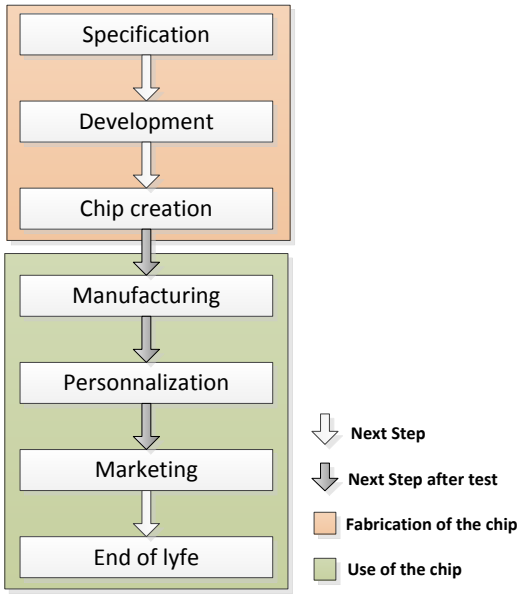

FIGURE 1.16 – Cycle de vie

Avant d'être utilisée en production, une carte à puce doit être vérifiée, validée, évaluée et certifiée. Elle possède alors un niveau acceptable et reconnu de sécurité

afin d'autoriser l'utilisateur à payer dans le cadre de la transaction de paiement. La figure 1.17 expose les acteurs principaux impliqués dans le cycle de vie de la carte à puce.

FIGURE 1.17 – Cycle de vie (Source : exploratheque.net [60])

1.5 Objectifs de la thèse

Cette étude, effectuée dans le laboratoire GREYC, et plus spécialement dans l'équipe Monétique et Biométrie, se concentre entre autres sur la sécurité logicielle d'applications embarquées sur une carte à puce. 5 membres de l'équipe travaillent sur le sujet : deux doctorant-ingénieurs, deux ingénieurs et un professeur. Notre vision est complétée par les discussions durant les conférences nationales et internationales et les échanges au sein de notre laboratoire. Ce sujet est donc un sujet appliqué à la carte à puce possédant deux visions complémentaires : une vision recherche académique mais aussi une vision en ingénierie. Ce point n'est pas négligeable et explique le déroulement de la thèse et les objectifs.

L'objectif est de traiter la carte à puce comme un système boite noire ou plus précisemment boite grise (nous devons avoir un minimum d'informations concernant le déroulement de la transaction par les spécifications afin de connaître le comportement

théorique de l'application carte mais aussi permettre de configurer nos outils d'analyse) en vue de son évaluation. Ayant une modélisation du comportement théorique de l'application, nous avons pour objectif de détecter un mauvais comportement. Nous modélisons cette application par une application de référence puis par une collection de propriétés, comportements locaux et attendus de l'application étudiée. Dans tous les cas, il s'agit de représenter un oracle de l'application utilisable en vue de détecter des anomalies sécuritaires ou fonctionnelles de l'application carte cible.

1.6 Conclusion

Nous avons présenté les bases des cartes à puce puis les attaques connues sur ce système, qu'elles soient invasives ou non. Il est nécessaire d'améliorer la sécurité de ces systèmes au vu de la large disponibilité d'applications dans de nombreux domaines. L'étude peut alors se résumer par la phrase suivante : De quelle manière pouvons nous représenter l'application cible afin de détecter et diagnostiquer une anomalie sécuritaire ou fonctionnelle de manière plus précise et automatique que les méthodes existantes (plan de tests ou un modèle formel) ?

Cependant, dans notre étude, nous n'avons pas accès au code source. Nous voulons étudier des applications du commerce telle que l'application de type EMV ou Porte Monnaie Electronique. Nous ne souhaitons pas formellement créer un modèle spécifique à l'application. Nous nous focalisons sur une étude basée sur l'ingénierie, possiblement sur la rétro-ingénierie, et les techniques de test. Il faudra bien sûr veiller à automatiser notre étude et la rendre la plus efficace possible tout comme ceci est fait dans la génération des cas de tests dans l'industrie ou le monde académique [50]. Ce que nous cherchons à faire est une solution open source, utilisable rapidement et simplement afin de modéliser l'oracle pour une évaluation d'un système de type boite-noire (à la rigueur boite-grise avec l'accès à une documentation nous permettant la configuration de la génération de l'oracle ou de l'évaluation finale notamment pour la gestion des données cryptées). A terme, un outil permettant l'analyse d'une application via l'observation d'une transaction et la détection d'anomalie sera notre principal objectif. Différents usages sont étudiés par la suite.

Nous dressons un état de l'art des techniques d'évaluation d'application dans une carte à puce et la présentation de la méthodologie que nous proposons dans la prochaine partie. Nous nous focalisons sur la partie logicielle dans la suite de l'étude.

Chapitre 2

Une méthodologie d'analyse d'applications sur carte à puce

Ce chapitre présente la méthodologie de vérification d'une application carte permettant d'évaluer le niveau de conformité et de sécurité vis à vis de spécifications. Cette méthodologie est basée sur la représentation du comportement de l'application, l'observation de la communication APDU et la détection de violation de propriété. Elle adopte une approche d'analyse en boîte noire inspirée du test, des méthodes formelles et de la rétro-ingénierie.

Sommaire

2.1 Introduction

Malgré une sécurité mise en avant pour ce genre de système à base de microcircuit, des attaques sont découvertes [32]. Il peut s'agir d'attaques invasives ou non, physiques ou logiques. Des soucis de clarté de spécifications ou de développement

incorrect ou incomplet peuvent être à l'origine de ces attaques. Après avoir exposé les méthodes existantes, nous proposons une méthode afin d'améliorer l'évaluation, et surtout la détection d'erreurs potentielles sur une application sur carte à puce. Nous illustrons l'approche proposée sur une application de paiement mais le principe serait le même dans un autre contexte.

2.2 Etat de l'art

Après avoir défini ce qu'est l'évaluation dans la partie précédente, nous étudions plus précisément les différentes méthodes connues de l'état de l'art. La carte à puce doit être considérée comme un mécanisme de confiance et le fournisseur doit garantir à l'usager entre autre la validité fonctionnelle et sécuritaire de l'application embarquée.

2.2.1 Méthodes d'analyse sur carte à puce

Nous listons des études majeures sur les applications cartes. Nous rappelons qu'il s'agit de différents types de méthodes : en boîte noire, blanche ou grise. Des évaluations peuvent survenir à différents moments de la mise en place d'une carte à puce comme vu précédement.

- Il peut s'agir d'évaluation lors de la phase de développement. Par exemple, la mise en place d'assertions. Afin de continuer le programme, chaque assertion doit être vraie.
- Il peut aussi s'agir d'évaluation après la production de la carte. Une entreprise peut essayer de stresser l'application carte afin de vérifier si une attitude est incorrecte. Nous pouvons voir des tests sur une transaction nominale ou encore des tests sur des transactions comportant des erreurs.

Méthodes formelles

Les méthodes formelles [61] sont des techniques informatiques d'une grande rigueur permettant, à l'aide de langages spécialisés et de règles logiques, de s'assurer (idéalement) de l'absence de tout défaut dans les programmes informatiques. Ces méthodes trouvent leurs racines dans les languages mathématiques appliqués à l'informatique, pour prévoir et anticiper, plutôt que pallier aux problèmes sur le terrain [62]. Généralement, ces méthodes reposent sur un langage (on discerne les languages généraux qui s'appuient sur l'utilisation d'une logique de premier ordre, de la théorie des ensembles et de l'algèbre universelle ou d'une logique d'ordre supérieur, des languages de descriptions ou de comportements). Après avoir défini

une description correspondant aux propriétés ou comportements du système, il est question de vérifier que des invariants représentant des propriétés attendues sont valides. On discerne principalement deux grandes familles ;

- Preuve de théorème : en entrée, une description mathématique du système afin d'obtenir les preuves de la conformité et de la validité du système [63]
- Model checking : représente et énumère l'ensemble des comportements du système (en limitant l'explosion combinatoire) [64]

Il est toutefois difficile de catégoriser les études suivantes car souvent les études mêlent différentes notions. La question de l'utilité des méthodes formelles dans le domaine de la carte à puce n'est pas nouvelle [65] et il est bon de connaître les grandes études même si cela ne constituera pas la pierre angulaire de notre étude, nous le verrons.

Les méthodes formelles consistent à définir des propriétés et à les vérifier formellement par des techniques mathématiquement rigoureuses. Les systèmes de preuve peuvent être utilisés pour une analyse statique de l'application carte. Il s'agit d'ajouter dans le code java des préconditions et des postconditions. En fait, ce sont des annotations qui doivent être vérifiées avant et après l'exécution d'une méthode.

Ces études concernent aussi bien le design que la validation d'application carte. Nous pouvons ainsi citer des méthodes telles que B ([66]), JML ([67], [68], [69]) ou encore OCL ([70]). Ces techniques sont aussi traitées dans [71] et [72].

Certaines études mêlent les différentes techniques [73]. D'autres méthodes plus récentes concernent la logique de séparation (prenant racine dans la logique de Hoare) ([74], [75], [76]) permettant de s'assurer que les règles de développement sont effectivement bien suivies.

Le Model Checking [64] fait tout aussi partie des méthodes de V&V et n'est pas un domaine très récent [77]. Il réunit des méthodes de vérification automatique des systèmes dynamiques. Il s'agit de vérifier si un modèle, un système ou une abstraction satisfait une spécification. L'utilisateur donne les entrées et attend sans intervenir les résultats de toutes les configurations possibles explorées dans un modèle du système préalablement défini. Des études ont été effectuées dans [78], [79], [80] et [81]. Des études plus récentes montrent que ce domaine reste très apprécié des chercheurs ([82], ([83], [84], ([85]).

Test and Fuzzing

La méthode de test permet de découvrir des erreurs à tous les niveaux et à chaque phase de la vie d'un système. C'est la méthode la plus utilisée dans le domaine de la carte [58]. Grâce à l'expérience, le testeur va pouvoir améliorer et alimenter son cahier de tests. En effet, il s'agit d'envoyer une entrée au système afin de vérifier la sortie. Contrairement au model checking, l'automatisation ne fait pas partie du coeur de la technique comme le montre la figure 2.1.

FIGURE 2.1 – Comparaison entre le test et le model checking (Source : babelfish.arc.nasa.gov[86])

La plupart des recherches dans le domaine concerne outre l'usage du test via un outil efficace, la génération des cas de test. De manière générale, des études sont effectuées sur la génération automatisée de test ([87], [88], [89], [90]), plus particulièrement, sur la carte à puce [91] et [92].

L'objectif est donc de générer le plus de cas de test possibles. Mais, il est clair que générer des cas de tests adaptés et pertinents est prioritaire. Certains vont utiliser des techniques de mutations [93] alors que plus récemment, ce sont des techniques catégorisées de Fuzzing qui vont être étudiés. De nombreux frameworks de fuzzing (ou fuzzer) sont connus et nous pouvons citer [94] ou encore [95]. Concernant le domaine qui nous intéresse, c'est à dire la carte à puce, ce sujet a été traité dans [96] puis dans [97]. Même si cette méthode coûte beaucoup en temps et en carte (car le nombre de transactions est limité pour une carte), elle a montré son utilité dans ce domaine.

Nous pouvons discerner plusieurs types de test. Les tests vont du test unitaire, c'est à dire la validation d'un élément spécifique [98] au test du système complet.

Selon l'accès au code source, il s'agira de test dit boite-noire ou boite-blanche. Voici un graphe représentant l'ampleur des possibilités du test dans la figure 2.2 :

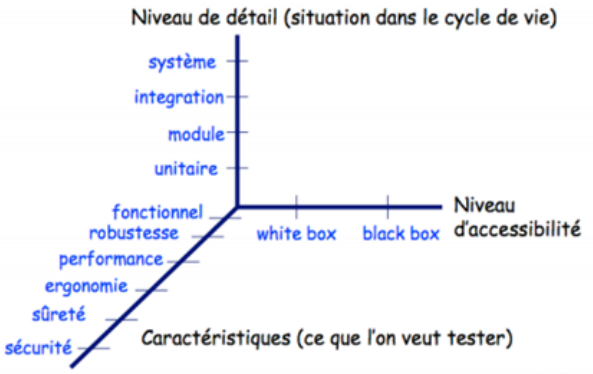

FIGURE 2.2 – Des tests nombreux et variés (Source : Polytech.unice.fr [99])

Enfin, comme écrit dans la partie sur les méthodes formelles, beaucoup de méthodes sont combinées afin d'obtenir de meilleurs résultats. Même si les études sont spécialisées, on observe un large panel de travaux dans ce domaine. Il n'est donc pas facile de catégoriser correctement toutes ces méthodes, les chercheurs n'utilisent généralement pas exactement la même terminologie. Par exemple, le model checking peut parfois être présenté comme du test formel comme dans [100]. On peut ainsi voir des techniques dites specification-based, behavioral-based ou encore scenario-based ([101], [102], [103] et [104]). Toutefois, les inspirations de ces méthodes se ressemblent.

Reverse engineering

Les techniques de Reverse Engineering [105] ou Retro-Ingénierie en français permettent d'analyser la carte à puce. Cette technique peut être utilisée au niveau logiciel afin de reconstruire le modèle par compréhension du fonctionnement de l'application (c'est à dire la machine d'états de l'application carte). Des études ont été effectuées sur l'applet javacard [106] afin de d'améliorer les données obtenus par les méthodes précédentes. Ces études sont assez récentes et des outils de plus en plus élaborés voient le jour ([107], [108], [84], [109]). Ces techniques consistent de partir d'un produit souvent fini et donc présent dans la vie de tous les jours. Les études suivent une approche attaquante et ceci afin de refondre la production du système, ce processus est appelé la ré-ingénierie comme exposé dans la figure 2.3.

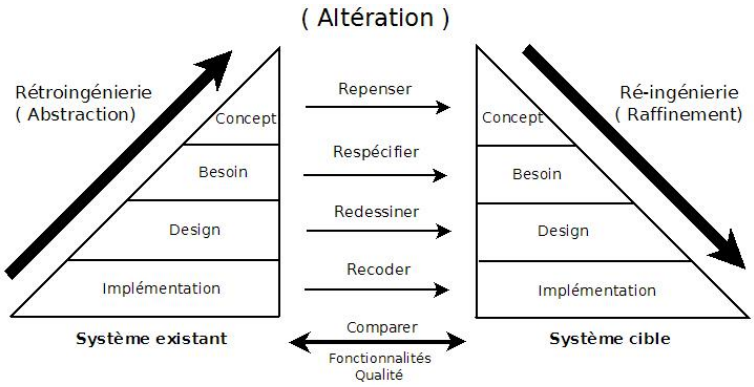

FIGURE 2.3 – La méthode de Reverse Engineering (Source : program-transformation.org [110])

2.2.2 Outils d'analyse de la carte à puce

Dans cette section, nous allons lister des outils utilisés pour étudier les cartes à puce.

Solutions propriétaires : outils de test

Dans le monde industriel, le secret est bien gardé sur les techniques utilisées. On peut toutefois lister un certain nombre de produits permettant le test de carte à puce. La plupart permet le test EMV de niveau 1 voire de niveau 2.

- FIS Open Test Solutions [111] est un expert des technologies de test dans le monde du paiement (anciennement, il s'agissait de Clear2pay [112] avec par exemple l'outil Card Test Tools)
- Galitt propose des services de test avec des outils tel que Kanest ICC ou VisuCard [113]
- Keolabs propose des solutions de validation basées sur Scriptis PICC [114]
- FIME [115] avec EMVericard est un fournisseur de confiance (laboratoire certifié par EMVco) d'outils de certification
- Elitt et l'organisme de certification PayCert assure une prestation d'évaluation et de certification Carte Bancaire (CB) et EMV (laboratoire approuvé EMV) [116]
- UL TS présente une solution de validation de carte EMV (anciennement connu sous le nom de Collis) [117]
- DayBreakSoftware propose des outils pour le développement et la personnalisation de carte EMV [118]
- COMPRION propose des solutions de test innovantes tel queSpectro 2 [119]

- Gemalto est un acteur majeur dans le domaine et fournit des outils comme Visual Tester [120]
- Riscure possède aussi sa solution de test [121]
- Barnes International fournit des outils divers de test et en particulier pour EMV [122]
- MicroPross propose son outil de test MP300 TC3 [123]
- SCardSoft dans un registre un peu plus restreint propose des logiciels tel que CardToolSetPro [124]

Solutions académiques : outils de test

Nous avons vu dans la section précédente que beaucoup de méthodes différentes existent. Il peut s'agir de contrôle (notice technique, review), d'analyse (vérification mathématique) ou encore du test (entrée/sortie). Pour ce faire, des frameworks ou des outils ont été développés dans le monde académique ou grand public. Nous en listons une partie, la plus intéressante pour notre étude :

- OpenSCDP, un ensemble d'outils pour le développement, le test et le déploiement de carte à puce [125]
- javaemvreader, un framework pour la lecture et l'interaction avec les cartes EMV [126]
- CardPeek, un outil permettant de vérifier le contenu d'une carte [127]
- The smartLogic Tool, un outil permettant le test [108]
- The Smart Card Detective, un intercepteur EMV [107]
- OpenSC, un ensemble de bibliothèques pour travailler sur les cartes [128]
- CardBrowser, un outil permettant l'étude du contenu d'une carte EMV [129]
- WSCT, un framework possédant des outils d'accès et de tests d'une carte développé au GREYC [130]

Les outils Verifast, Jstar ou Key tool cités précédemment sont des outils permettant l'utilisation de méthodes formelles sur les cartes à puce.

2.2.3 Solutions académiques : outils de développement

Voici une liste des outils permettant de developper facilement des outils pour la carte à puce.

- SmartCard Framework, un framework pour développer des applications pour les cartes [131]

TABLE 2.1: Comparaison des outils de développement

	Fonctionnalités	Documentation	Maintenance
SmartCard Framework	⋆	⋆⋆	⋆
OpenSCDP	⋆⋆	⋆⋆⋆	⋆⋆
PCSC_Sharp	⋆	⋆	⋆
OPAL	⋆⋆	⋆⋆	⋆⋆
WSCT	⋆⋆⋆	⋆	⋆⋆⋆

- Open SCDP, un ensemble d'outils pour le développement, le test et le déploiement de carte à puce [125]
- PCSC_sharp, un outil simple pour communiquer avec une carte à puce [132]
- OPAL, un framework pour travailler avec les cartes à puces [133]
- WSCT, permet le développement d'outils pour la carte [130]

Nous comparons les outils dans le tableau 2.1 Les fonctionnalités contiennent l'observation et les mécanismes de rejeu. Plus il y a d'étoiles comme ⋆, plus les outils proposent des fonctionnalités intéressantes, une documentation complète et accessible et une maintenance récente (mise à jour).

2.2.4 Construction de l'oracle

La technique de test principalement utilisée pour vérifier des applications carte est l'ensemble des activités ayant pour but de déterminer des différences entre le comportement réél et le comportement théorique d'une application. Afin de procéder au test, il faut deux éléments :

- Les données d'entrée : il s'agit de la génération des cas de tests à appliquer à la carte
- Le résultat : l'oracle qui détient la vérité

Alors que nous nous inspirons des méthodes de génération intelligente tel que le fuzzing pour la génération des données d'entrées dans des travaux de notre laboratoire, nous traitons la génération de l'oracle dans le chapitre suivant. L'idée est d'utiliser le système ou un modèle du système. Dans tous les cas, c'est un problème pour l'automatisation des tests [134].

2.2.5 Discussion

La figure 2.4 permet de comparer selon plusieurs critères les méthodes vues précédemment.

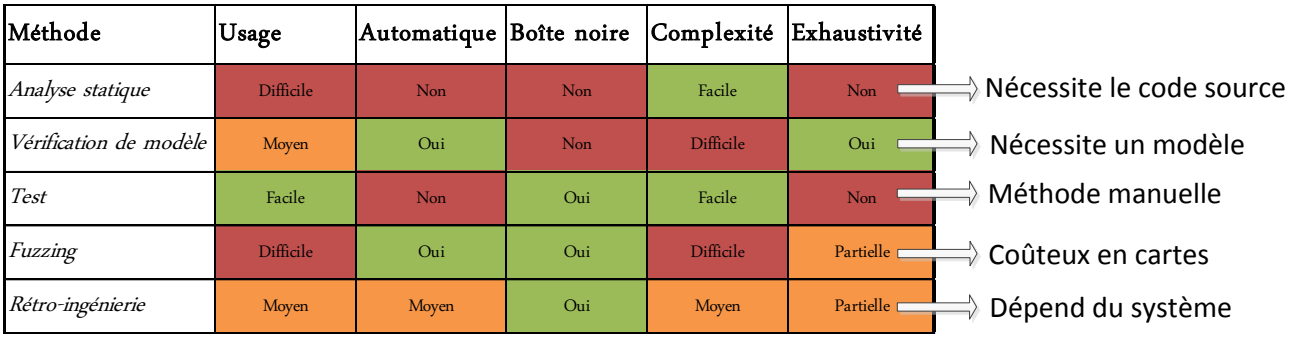

FIGURE 2.4 – Comparaison des méthodes principales

Le schéma 2.5 replace rapidement le périmètre de notre étude par rapport aux méthodes traitées dans cet état de l'art. Alors que l'analyse statique d'une application adopte une analyse boite blanche (par l'annotation de code source par exemple), nous nous plaçons dans un contexte boite noire. Nous ne sommes pas dans un cadre de vérification de modèle, notre ambition n'étant pas de représenter formellement un système mais de travailler sur un système qui est utilisé dans la vie de tous les jours. Nous nous rapprochons donc d'une méthode basée sur la rétroingénierie (ou observation sur la figure) au niveau de l'usage avec comme principale inspiration le test et le fuzzing. Nous utilisons tout de même un oracle tout comme le test afin de s'assurer de la conformité de notre système cible.

Pour notre étude, il nous faudra travailler à deux points principaux. La génération de l'oracle qui sera constitué d'un ensemble de propriétés (c'est-à-dire des comportements locaux et attendus de l'application cible). Très proche d'un comportement local de cas de test, ces comportements sont définissables manuellement via la documentation mais nous verrons qu'il est possible d'utiliser soit une méthode proche de la rétro-ingénierie soit une méthode à partir d'un modèle pour automatiser la génération d'un oracle complet. Nous nous focalisons sur une méthode de détection d'anomalie via une observation de la communication à la volée proche des méthodes de test et de rétro-ingénierie.

En effet, nous voulons une méthode permettant de savoir si une application carte est correcte ou non mais aussi réalisant une détection rapide et simple de l'anomalie en nous donnant plus d'informations sur les raisons qui ont conduit à cette erreur

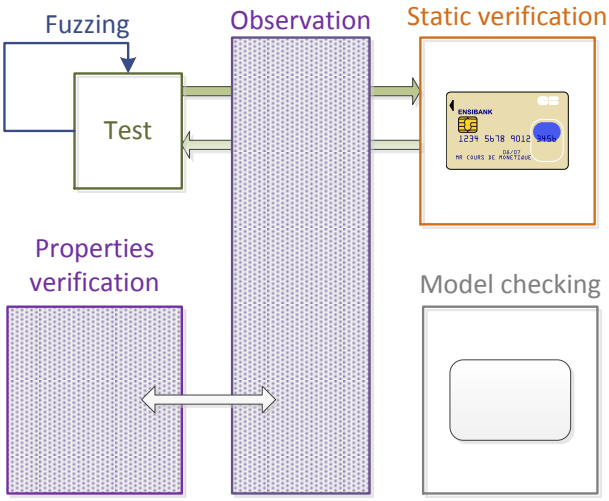

FIGURE 2.5 – Positionnement des différentes méthodes

fonctionnelle ou sécuritaire. Nous nous focalisons sur le logiciel contenu dans la carte en ayant comme biais la conformité du matériel utilisé dans la carte.

D'un côté, nous avons de nombreux outils de test industriels, assez confidentiels, auxquels nous n'avons pas accès gratuitement et encore moins au code source. C'est un problème majeur dans cette étude, le sujet est un domaine assez fermé et tenu par l'industrie carte, le thème étant assez critique. De l'autre côté, nous avons un certain nombre d'études, avec pour beaucoup l'utilisation de méthodes formelles (preuves ou étude basée sur un modèle formel) pas forcément faciles à utiliser et qui nécessitent pour la plupart l'accès au code source de l'application.

Cependant, dans notre étude, nous n'avons pas l'accès au code source. Nous voulons étudier des applications du commerce telle qu'une application de type EMV ou Porte Monnaie Electronique. Nous ne souhaitons pas formellement en créer un modèle. Nous nous focalisons sur une étude basée sur l'ingénierie, possiblement sur la rétro-ingénierie, et les techniques de test. Il faudra bien sûr veiller à automatiser notre étude et la rendre la plus efficace possible tout comme ceci est fait dans la génération des cas de tests dans l'industrie ou le monde académique [50]. Ce que nous cherchons à obtenir est une solution open source, utilisable rapidement et simplement afin de modéliser l'oracle pour l'évaluation d'un système de type boite-noire (à la rigueur boite-grise avec l'accès à une documentation nous permettant la configuration de la génération de l'oracle ou de l'évaluation finale notamment pour la gestion des données chiffrées). A terme, un outil permettant l'analyse d'une application via l'observation

d'une transaction et la détection d'anomalie sera notre principal objectif. Différents usages seront étudiés par la suite.

Le framework WSCT [130] développé par Sylvain Vernois et de nombreuses personnes (stagiaires, doctorants et ingénieurs ayant participé à son développement au fur et à mesure des années) est à notre disposition. Ce framework constitue une base de travail pour l'accès et la communication avec une carte à puce. Nous mettons l'accent sur l'utilisation de la méthodologie dans un contexte réel et pas seulement théorique. Ceci nous permet de livrer un ensemble d'outils utilisables dans un contexte concret.

Nous définissons dans un premier temps un langage pour manipuler des propriétés que doit respecter une application. Ce langage décrit en XML doit etre simple à utiliser, évolutif et efficace.

2.3 Représentation de l'application et de son comportement

Afin de représenter l'application et ses comportements théoriques et donner des informations par rapport aux raisons qui ont conduit à un mauvais comportement de la part de l'application carte, nous avons défini un langage.

2.3.1 Introduction

Contrairement aux méthodes formelles tels que la vérification de modèle ou le théorème de preuve, nous vérifions à la volée des séquences d'éléments APDU par rapport à un oracle, et ceci durant la transaction. En fait, dans les études [79] [66] [65] [68] or [78], la vérification nécessite un modèle formel ou l'accès au code source. Avec notre language, nous pouvons définir des comportements théoriques (locaux ou globaux) utilisant seulement les données transmises.

2.3.2 Définition

Le langage nous permet de définir à la fois des machines à états, permettant de suivre le comportement global d'une application, mais aussi de les partitionner, permettant de suivre le comportement local d'une application. Nous présentons les éléments de base, puis les éléments nous permettant de définir les machines à états et

enfin les éléments nous permettant de définir des comportements locaux. Ce language est décrit en langage de balisage extensible (XML) [135].

2.3.3 Eléments de base

Nous avons défini deux types d'éléments de base sans lesquels la représentation des données n'est pas possible. Dans un premier temps, nous avons les éléments représentant la communication APDU :

- **Instruction :** représente les octets CLA et INS d'une commande APDU
- **Parameters :** représente les octets P1 et P2 d'une commande APDU
- **Status :** représente les octets SW1 et SW2 d'une commande APDU

Le second type permet de représenter les relations entre plusieurs éléments :

- **And :** tous les éléments sont vrais
- **Or :** au moins un élément est vrai
- **Nor :** tous les éléments sont faux

Afin de compléter la représentation des commandes et réponses, nous pouvons compléter les éléments représentant les instructions, les paramètres et les status words, éléments obligatoires lors d'une communication unitaire commande et réponse APDU (un couple APDU). La représentation d'une partie ou de la totalité des données transmises (i.e. les données contenues dans UDC et UDR). Avec des données de type TLV, nous pouvons aller plus loin et traiter une partie de la donnée complète. De plus, l'appel à des fonctions externes est possible et permet une utilisation spécifique d'une partie des données ou d'une association de données (garder en mémoire une partie, effectuer des opérations sur des données transitant sur plusieurs couples APDU).

Le second type permet de représenter les relations entre plusieurs éléments :

- **Cdata :** correspond à l'UDR entier
- **Rdata :** correspond à l'UDC entier
- **Tag :** correspond à un élément de type Tag Longueur Valeur (TLV)
  - nom : nom du tag (exemple : 0x9F27)
  - valeur : valeur de l'élément (exemple : 0x40)
- **mask :** permet d'appliquer un masque sur un élément
- **call :** permet l'appel à une fonction de traitement de données supplémentaires

Ce dernier élément permet d'améliorer les vérifications par l'ajout de fonctionnalités (exemple : opérations sur des éléments chiffrés). Ceci permet d'avoir une

représentation évolutive et accessible.

2.3.4 Représenter le comportement global

En plus des éléments précédents, la définition d'une machine à états, c'est à dire un comportement global d'une application carte, nécessite les éléments suivants :

- **cardstate :** représente l'état
  - name : nom de l'état
  - x, y : position visuelle pour l'interface
  - default : indique l'état initial de la machine à états
- **transition :** représente une transition permettant de modifier l'état courant de l'application carte :
  - name : nom de la transition
  - from : état courant
  - to : état suivant
- **from / to :** permet de construire les transitions entre les états
  - nom : nom de l'état de départ
  - direction : haut, droite, bas ou gauche qui correspond à la position visuelle de la base de la fléche partant d'un état et arrivant à un second état représentant la transition

2.3.5 Représenter le comportement local

En plus des éléments de base, nous avons besoin de ces éléments pour définir les comportements locaux ou propriétés :

- **property :** comportement théorique et attendu de l'application carte
  - name : nom de la propriété
  - explanation : description de la propriété
- **step :** indique l'ordre des éléments
- **tag :** permet de vérifier une donnée particulière contenu dans l'APDU

2.3.6 Modèle de représentation

Ces modèles de représentation permettent d'illustrer la forme d'une machine à états dans le listing 2.1 ou d'un comportement local dans le listing 2.2 (que l'on présente dans la sous section suivante) nommé propriété.

Listing 2.1: Modèle pour la représentation d'une machine à états

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<machine>

<!-- Définition des états possibles et leurs coordonnées (affichage)
    -->
 <cardstate name="CardState1" x="100" y="120" />
 <cardstate name="CardState2" x="100" y="230" />

<!-- Les transitions sont définies de la sorte -->
 <transition text="Nom de la transition">
  <from name="State of departure" dir="Down" />
  <to name="state of arrival" dir="Up" />
  <and>
   <instruction instruction="Couple CLA/INS"/>
   <status status="SW"/>
  </and>
 </transition>

</machine>
```

Listing 2.2: Modèle pour la représentation de comportements locaux

```
<properties>

<!-- Les propriétés sont de la forme suivante -->
    <property name="Nom de la propriété">
    <step>
      <instruction instruction="INS" />
      <status status="SW" />
      <cdata cdata="UDC"/>
    </step>
  </property>


    <property name="Nom de la propriété" explanation="description">
    <step>
      <instruction instruction="INS"/>
      <status status="SW"/>
      <cdata cdata="UDR"/>
      <or>
        <parameters parameters="P1/P2" mask="Valeur du masque"/>
        <parameters parameters="P1/P2" mask="exemple : 0x0000"/>
      </or>
      <require>
        <nor>
          <tag name="TAG" value="Valeur" />
        </nor>
```

```
      </require>
    </step>
  </property>

  <property name="Nom de la propriété">
    <step>
      <instruction instruction="INS"/>
      <status status="SW" />
        <call method="Nom de la fonction à appeler" />
    </step>
  </property>
</properties>
```

2.4 Observation de comportements globaux et locaux d'une application

2.4.1 Contexte

Le but principal est d'améliorer les méthodes d'évaluation connues en créant un module complémentaire capable d'observer le comportement du système et de mesurer sa conformité avec le comportement théorique ou référence du système, c'est-à-dire les spécifications. Comme l'état du système cible qu'est la carte à puce ne peut changer que par la réception d'une commande APDU (l'état est alors indiqué par la réponse APDU), l'observation des entrées et des sorties du système semble pertinent afin de suivre l'évolution de l'application. Le perimètre est simplement exposé par la figure 2.6.

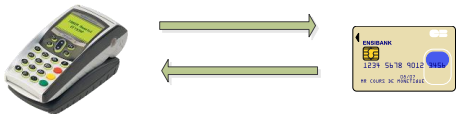

FIGURE 2.6 – Le périmètre de l'étude

2.4.2 Inspirations

Suite à l'étude de l'existant dans la partie précédente, nous avons vu un manque dans le panel des techniques utilisées pour l'évaluation d'applications sur carte en mode boîte noire, accessible et utilisable simplement dans le domaine de la carte à

puce. Les inspirations principales sont récapitulées ici. Dans un premier temps, la technique de vérification appelée "Assertion-based design" tout droit tirée du domaine électronique [136] nous a permis de réfléchir d'une autre manière l'évaluation d'une application sur carte. Cette technique consiste en la spécification de propriétés, l'extraction automatique de propriétés combinées à des méthodes formelles [137]. Nous avons étudié l'utilisation des propriétés temporelles dans des domaines proches de notre cas. Nous pouvons ainsi citer [138], [139], [140] et [141]. Pour terminer, ces études mêlant des modèles formels et l'utilisation de rétro-ingénierie ont été très inspirantes pour notre méthodologie comme dans [142], [143] ou [144]. Notre méthodologie ressemble à la stratégie de couverture de tests à haut niveau dans [103].

2.4.3 Analyse temporelle sur la communication APDU

L'état du système va évoluer au fur et à mesure que les commandes et réponses APDU sont échangées. Pour connaître le comportement d'une application carte, nous pouvons donc étudier la communication entre le terminal et la carte à puce et détecter un mauvais comportement de l'application carte. Une horloge, à l'image de ce qui existe dans le monde du semi-conducteur avec l'assertion based design [136], peut alors être définie grâce aux deux événements suivants : réception d'une commande, envoi d'une réponse. Des assertions, des propriétés et des séquences peuvent alors être définies. Prenons en exemple trois signaux A, B et C. La séquence $S : A$ *et* B *et* $\overline{C}$ est vraie sur le second front montant de l'horloge. Sur la figure 2.7, un exemple d'horloge, de signaux et de propriétés sont fournis.

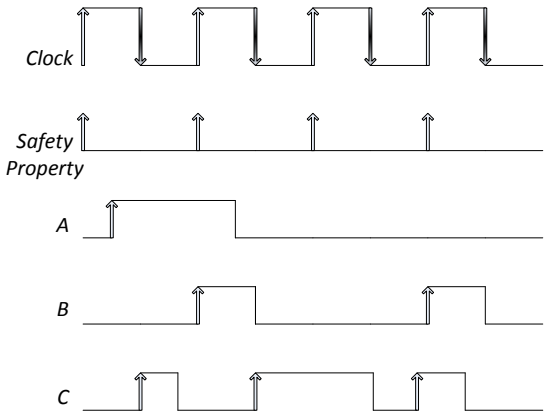

FIGURE 2.7 – Analyse temporelle et exemples

La figure 2.8 fournit une illustration des niveaux d'abstraction au fil du temps,

ce qui expose les bases de ce type d'analyse.

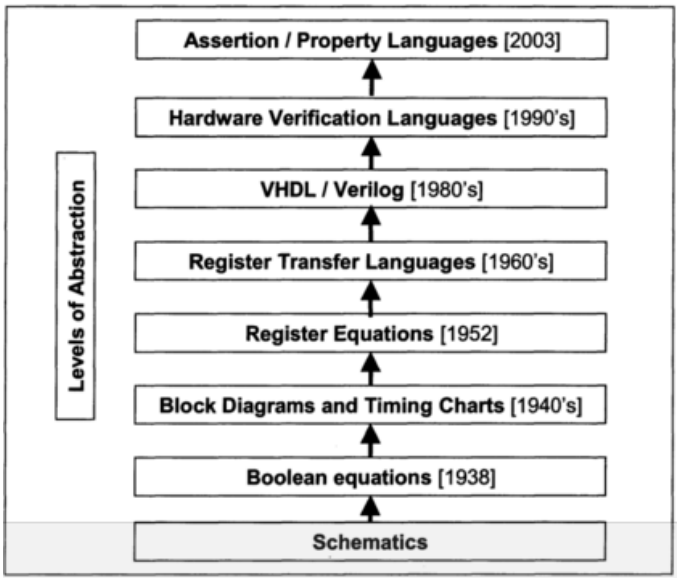

FIGURE 2.8 – Evolution du niveau d'abstraction (Source : Assertion-based design [136])

La structure des communications transitant entre le terminal et la carte à puce est décrite dans la 2.9. Une commande est automatiquement suivie d'une réponse. Différents types de propriétés peuvent alors être créés (sur un couple commande/réponse ou sur plusieurs couples commande/réponse).

FIGURE 2.9 – La structure des données APDU

Ces éléments sont accessibles via les fonctionnalités de WSCT et le langage de représentation préalablement défini permet de représenter une application de manière globale ou partielle. La prochaine section expose la méthode de détection de violation de propriété permise par l'osbervation de la communication APDU et la définition d'un oracle de l'application.

2.4.4 Détection d'anomalie grâce à l'observation de la communication

La méthode consiste en l'observation de la communication APDU entre une carte à puce et un terminal afin de détecter des anomalies sur l'application. Une fois la

bibliothèque complète de propriétés, conformes aux spécifications de l'application, il s'agit de détecter un comportement anormal de l'application à la volée durant la transaction. Nous proposons trois approches par la suite. La figure 2.10 expose le principe général des solutions proposées.

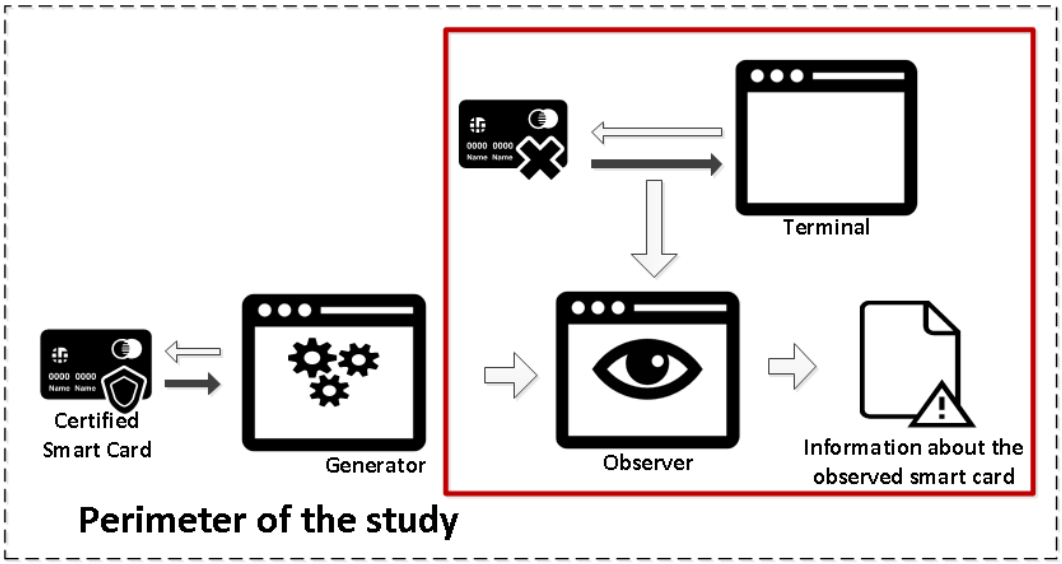

FIGURE 2.10 – Schéma général de la méthodologie

Approche A : basée sur la machine à états

La première approche consiste à vérifier si une application est conforme avec la machine à états définies par les spécifications de l'application. Un fichier de configuration définit les états possibles de l'application et les transitions qui peuvent faire évoluer l'application d'un état à un autre. Cette approche est un moteur de vérification de la conformité de la machine à états. Une simple comparaison entre l'état théorique et l'état réél est possible par la capture de la communication APDU (notamment le Status Word de la réponse APDU). La figure 2.11 illustre cette première approche.

Approche B : Observation et détection de violation de propriété

Comme définie dans l'article [145], cette approche est basée sur l'assertion-based design [136]. Nous pouvons appliquer les principes de cette méthode au monde de la carte à puce. En effet, les commandes et réponses APDU définissent implicitement une horloge sur laquelle des séquences peuvent être définies. Une propriété implique un ou plusieurs couples APDU, c'est à dire un ou plusieurs cycles d'horloge. Cette approche

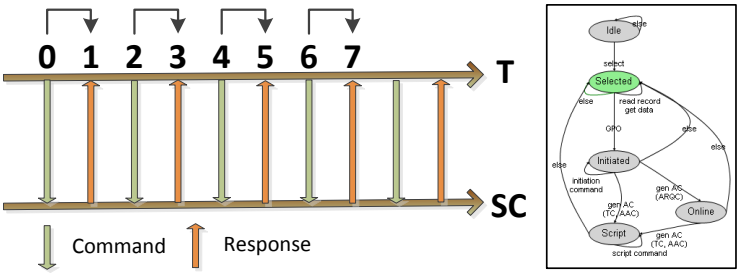

FIGURE 2.11 – Approche A

consiste en la génération de nombreuses propriétés à vérifier durant la transaction et une violation d'une propriété correspond à une anomalie de l'application. La figure 2.12 illustre l'approche.

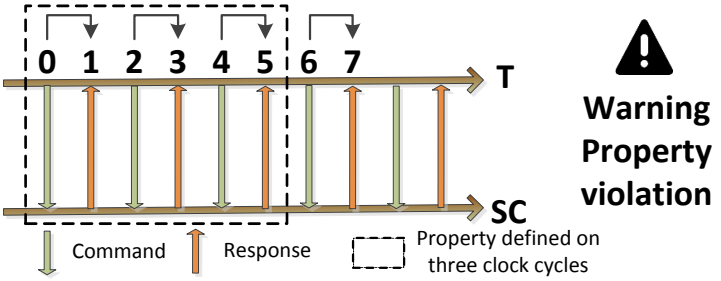

FIGURE 2.12 – Approche B

Approche C : Méthode hybride

Des limites sont apparentes dans les deux approches. La première manque de précision, nous ne détectons que l'erreur et pas la cause de cette erreur. La seconde approche donne plus d'information sur une erreur par la définition de la propriété violée (nous connaissons sa position dans la cinématique de la transaction et les données en cause). Cependant, la création d'un ensemble de propriétés, traitée plus tard dans ce document, n'est pas une tâche simple, surtout si nous ne prenons pas en compte l'état courant de l'application. La solution est de combiner les deux techniques afin de réduire la complexité par l'approche A tout en gardant la précision et l'exhaustivité pertinente de l'approche B. Le tableau suivant 2.2 récapitule la comparaison des approches.

Cette méthode adopte une approche de vérification boite-noire, automatique, générique et modulaire. L'exhaustivité et la complexité dépend bien sûr de l'application

TABLE 2.2: Comparaison des approches

Approche	Automatique	Complexité	Exhaustivité	Précision
A : basée sur la machine à états	⋆ ⋆ ⋆	⋆	⋆	⋆
B : basée sur les propriétés	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆
C : hybride	⋆ ⋆ ⋆	⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆

cible. L'utilisation de propriétés permet la modularité et la généricité et elle devient automatique après une configuration des propriétés par l'utilisateur (configuration des données, des clefs, etc). Dans la prochaine section, nous exposons l'analyse permise par l'observation de violation de propriétés du comportement de l'application.

2.5 Analyse par la détection de violation de propriété

2.5.1 Introduction

L'objectif principal est de s'assurer de la validation d'une application en détectant les possibles erreurs d'implémentation. Une propriété est un comportement local de l'application carte et doit être vraie à tout moment. Si une propriété est violée, l'implémentation n'est pas conforme aux spécifications. Schématiquement, une propriété consiste en une séquence de commandes et réponses APDU admissible par la machine à états décrivant l'application testée.

2.5.2 Définition de propriété

La figure 2.13 présente une horloge plus précise. En fixant le point d'origine (0), nous pouvons étudier l'évolution de la carte à puce, plus précisemment de l'application, par l'étude des séries de n paires commandes/réponses. Une séquence va donc être une série de paires commandes/réponses. Une propriété va être composée d'une séquence et d'une vérité. En effet, une propriété est toujours vraie, sauf si l'application possède une erreur et la propriété est donc violée. Nous pouvons définir deux types de propriétés : les propriétés simples (sur un cycle d'horloge, une paire commande/réponse) et les propriétés temporelles (définies sur plusieurs paires). La commande correspond alors au front montant de l'horloge et la réponse au front descendant. Ces propriétés doivent être conformes aux spécifications de l'application. Finalement, nous pouvons détecter, à la volée et sans connaître le code source de l'application, quand une propriété est violée et donc la présence d'une erreur d'implémentation dans une application.

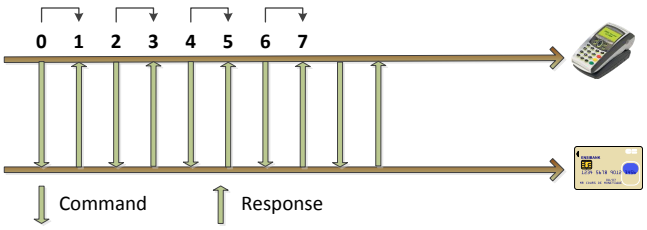

FIGURE 2.13 – L'horloge définie sur la communication APDU

Techniquement, une propriété est un comportement local, c'est à dire à un moment donné, nous sommes capables de prédire comment la carte à puce doit évoluer. Les commandes sont alors associées aux nombres pairs et les réponses aux nombres impairs. La valeur des champs composant la réponse (SW1, SW2 et UDR) nous permettent de savoir comment l'application réagit à une commande composée de champs (CLA, INS, P1, P2, LC, UDC et LE). P_0 est la structure de toute propriété. Une seconde écriture, par équivalence logique, est possible. En effet, nous avons *A implies B* equivalent à *B or Not(A)*.

$$P_0 : (champ(1)\ et\ ...\ et\ champ(j))\ or\ \overline{(champ(0)\ et\ ...\ et\ champ(i))}$$
$$avec\ \ 0 < i < j$$

Type simple

Une première propriété basée sur un seul cycle d'horloge est l'intégrité d'un couple commande/réponse. La propriété P_1 vérifie si une réponse est correcte. Les données renvoyées par la carte sont-elles en accord avec la commande émise ? LE est la longueur du champ UDR, qui n'est donc pas nul si LE n'est pas égale à 0.

Longueur de la réponse correcte :

$$P_1 : ((UDR(1) \neq 0)\ et\ (UDR(1).longueur = LE(0))\ ou\ \overline{(LE(0) \neq 0))}$$

Type temporel

Sur deux cycles d'horloge, nous pouvons détecter un rejeu grâce à la propriété P_2. La propriété "pas de rejeu" est vraie tout le temps sauf si tous les champs sont égaux.

Rejeu interdit :

$$P_2 : (CLA(2) \neq CLA(0)) \; ou \; (INS(2) \neq INS(0)) \; ou \; (P1(2) \neq P1(0)) \; ou \; (P2(2) \neq P2(0)) \; ou \; (LC(2) \neq LC(0)) \; ou \; (UDC(2) \neq UDC(0)) \; ou \; (LE(2) \neq LE(0))$$

2.5.3 Détection de violation

Nous présentons plus précisemment la détection d'une anomalie de manière concrète et l'apport de la méthode par rapport au test classique.

Objectif par rapport au test

Le test est une technique permettant la vérification de couples entrée/sortie d'un système. L'exactitude de ces couples permet de certifier que la carte se comporte convenablement (à condition que les suites de tests soient bien pensées et complètes). L'utilisation de propriétés permet de détecter à la volée une anomalie durant ce plan de test. Nous avons alors une documentation accrue sur la présence d'une erreur et sur les éléments déclencheurs de cette anomalie 2.14.

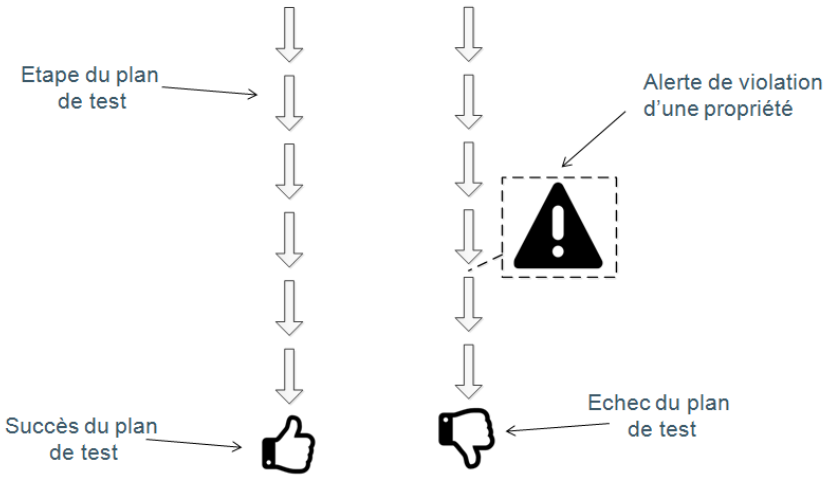

FIGURE 2.14 – Apport de l'analyse basée sur les propriétés au test

Exemple d'anomalie

Le dernier exemple, P_3, nous permet de détecter un problème d'implémentation d'une application de paiement. Il s'agit de vérifier localement le comportement de la carte à puce entre les états EMV "selected" et "initiated" (voir chapitre 1). Pour cet exemple, nous vérifions uniquement la commande GPO mais il faut généraliser afin

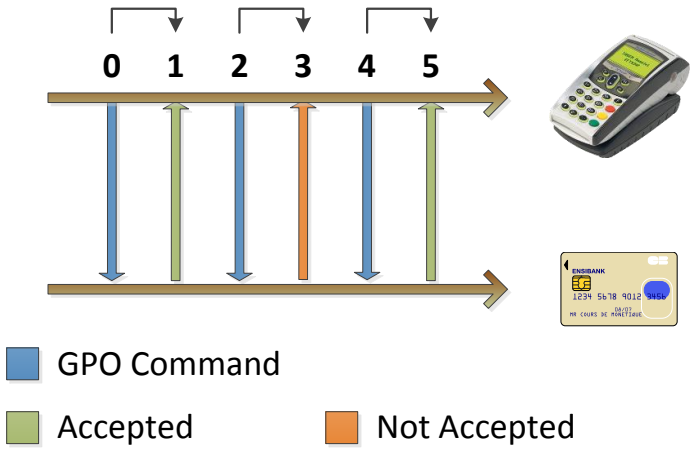

FIGURE 2.15 – Propriété P3

de contrôler le comportement global entre ces deux états. Nous pouvons reprendre la figure 2.15 qui nous montre une propriété impliquant trois couples commande/-réponse. Les commandes sont les éléments transitant aux temps d'horloge 0, 2 et 4 alors que les réponses transitent aux temps 1, 3 et 5. La propriété est donc une vérité sur une séquence de commandes et de réponses APDU, donc de leurs champs.

Comportement local entre selected et initiated en utilisant trois GPO d'affilée :

$$P_3 : ((SW1(1) = 90)\ et\ (SW2(1) = 00)\ et\ (SW1(3) \neq 90)\ et\ (SW2(3) \neq 00)\ et\ (SW1(5) = 90)\ et\ (SW2(5) = 00))\ ou\ \overline{(INS(0) = A8)\ et\ (INS(2) = A8)\ et\ (INS(4) = A8)}$$

Détection de violation

Pour comprendre la portée de la méthode de détection basée sur les propriétés, la figure 2.16 donne une explication des trois comportements possibles de l'application :

- **a)** Le premier schéma est le résultat d'un cas de test réussi à partir de la sélection de l'application jusqu'à l'utilisation de la commande Generate AC. Le testeur connait le comportement attendu de l'application comparé au comportement obtenu au test. L'application répond de manière appropriée à ce cas de test. Rien d'anormal n'est détecté à la fois par le test ou par les proprietés.
- **b)** Le second schéma présente un refus anormal de la commande Generate AC par rapport au comportement théorique mais aussi une détection de violation

de la propriété P3 (en bleu), ce qui explique la raison de l'anomalie détectée par le test (en orange). La violation de la propriété donne une documentation améliorée sur le résultat du test.

– **c)** Le dernier schéma représente un cas de test réussi. Cependant, une propriété a été violée. En fait, dans ce cas, l'application contient une erreur mais ce n'est pas critique comparé au comportement théorique car la commande Generate AC est acceptée, ce qui est le but du test. Dans ce cas, nous avons détecté une erreur non gérée par le test mais intéressante car montre la non conformité de l'application vis à vis des spécifications sur un point non critique.

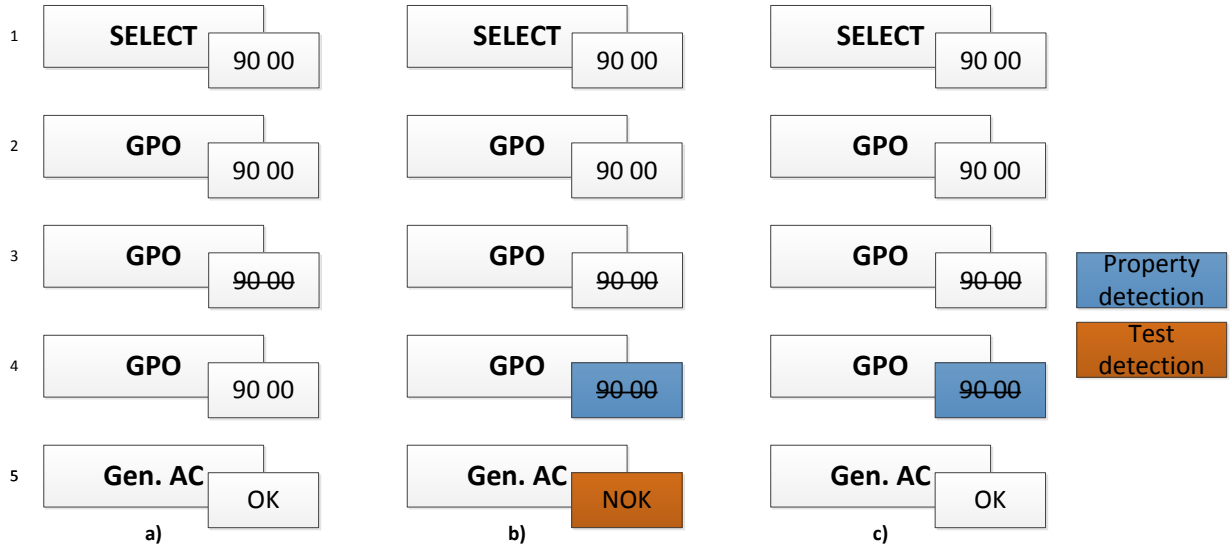

FIGURE 2.16 – Communication APDU durant une phase de test

Explication du comportement de l'application

La figure 2.17 présente une machine d'états partielle de l'application correspondant aux trois comportements vus précédemment.

– **a)** Le premier cas est le comportement d'une carte à puce certifiée et donc correcte.

– **b)** Le second cas montre un comportement anormal si trois commandes GPO sont appliquées à la carte. Dans cette situation, nous ne savons pas vraiment dans quel état l'application est après les deux refus car ce n'est pas défini par l'application. Nous pourrions continuer l'analyse et appliquer, après cet état, plusieurs commandes pour le savoir.

– **c)** Troisièmement, la détection de violation de la propriété P3 nous montre une erreur d'implémentation. Ce comportement est accepté par le test mais la

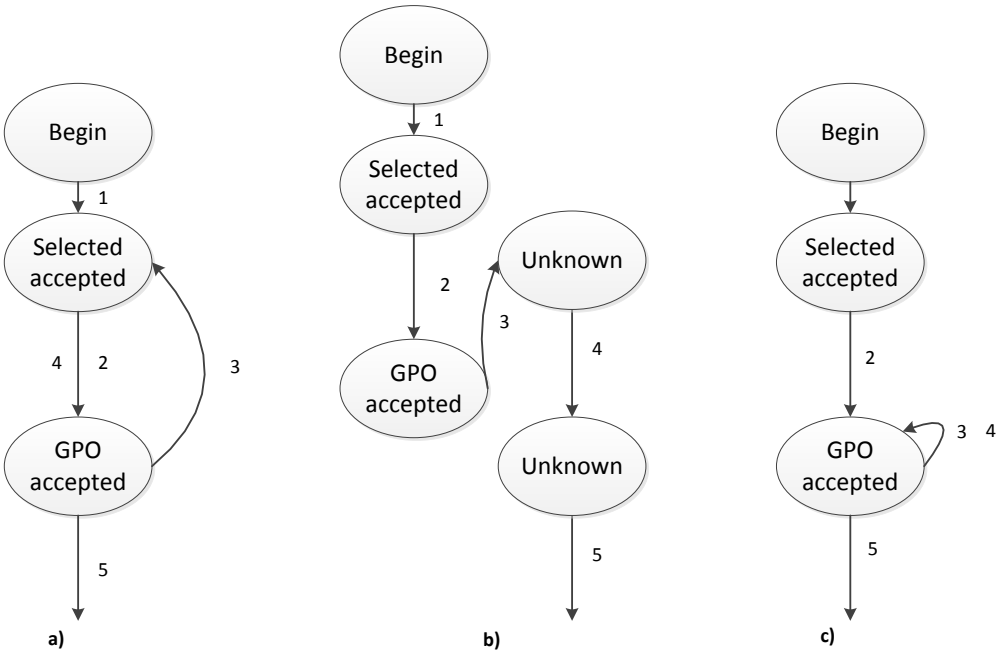

FIGURE 2.17 – Le comportement correspondant de la machine à états

propriété nous montre plus d'informations sur l'application, cet aspect peut être intéressant pour améliorer l'implémentation de l'application carte.

2.6 Illustrations de la méthodologie

Cette section expose la méthodologie générale d'évaluation d'application sur carte à puce. Nous exposons la mise en place de cette méthodologie, c'est à dire le framework utilisé pour notre preuve de concept et la représentation de l'application utilisée par notre outils d'analyse. Quant à l'utilisation dans un contexte concret, deux illustrations sont données de cette méthodologie sur des applications de paiement.

La figure 2.18 représente schématiquement l'utilisation de la méthodologie. Nous exposons une méthodologie de génération d'oracle automatique et pertinente dans le chapitre suivant.

2.6.1 Mise en place de la méthodologie

WSCT est utilisé pour l'exploration et la recherche de faille sur cartes à puce [130]. Des plugins permettent de travailler sur des types de cartes EMV, TB100, etc. Ce logiciel a été utilisé afin de rejouer des attaques logicielles connues, telle que

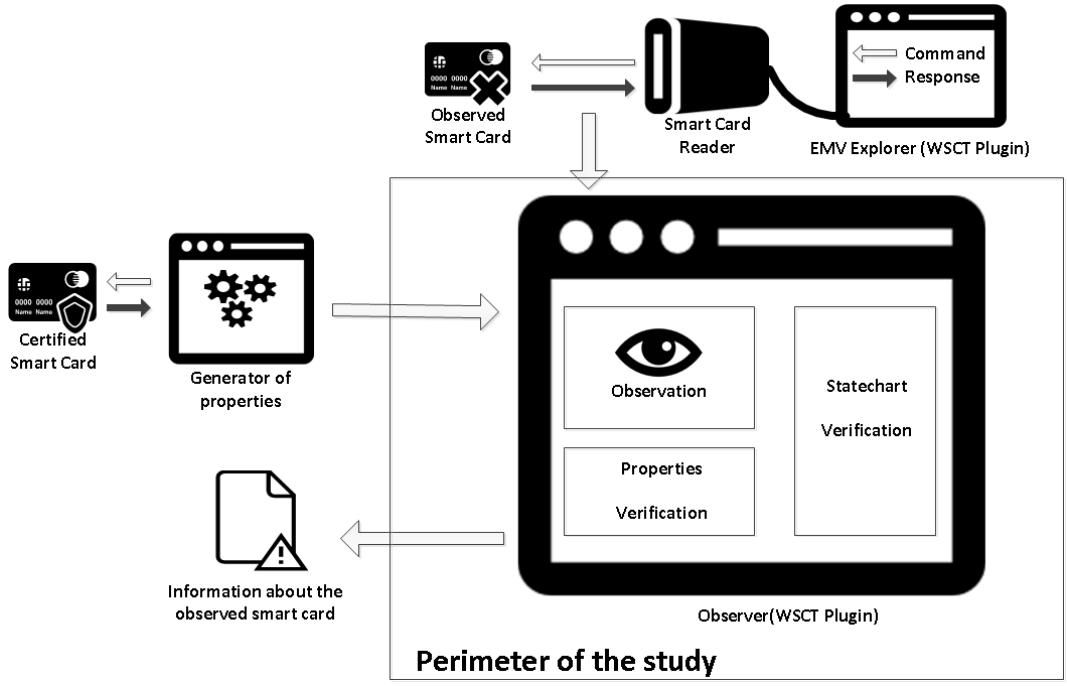

FIGURE 2.18 – Illustration de la méthodologie

l'attaque de Cambridge de manière logicielle [146]. On peut observer deux notions importantes :

- une interface de programmation orientée objet permettant l'accès à un lecteur de carte à puce ;
- une interface graphique évolutive avec la création de plugins permettant la manipulation de cartes à puce.

Le framework WSCT semble le plus adapté pour notre étude. L'ajout d'un module se fait comme donné dans le listing 2.3.

Listing 2.3: Système d'évolution du framework WSCT par l'ajout de plugins (bibliothèque dynamique)

```
<Plugins>

    <plugin name="Observer">
        <dll>Plugin.Observer.dll</dll>
        <className>WSCT.GUI.Plugins.Observer.PluginEntry</className>
        <pathToDll></pathToDll>
    </plugin>

</Plugins>
```

Ensuite, il faut ajouter les machines à états et les collections de propriétés associées de chaque application à l'étude, comme exposé dans le listing 2.4. Dans notre cas, le listing suivant nous donne la forme à suivre pour inclure des cas d'utilisation et leur configuration. On peut ajouter plusieurs configurations de propriétés par application. Nous voyons de manière concrète les deux cas d'utilisation suivants : Observation de l'application et analyse de l'application EMV.

Listing 2.4: Chargement des fichiers de configuration pour les applications cibles

```
<?xml version="1.0" encoding="utf-8"?>
<machines >
  <machine>
    <name>M/Chip</name>
    <path>MachineState.MChip.xml</path>
    <properties>
      <name>Default</name>
      <path>Observer.MChip.Properties.xml</path>
    </properties>
    <properties>
      <name>Test</name>
      <path>TestProperties.xml</path>
    </properties>
  </machine>
  <machine>
    <name>PME</name>
    <path>MachineState.Test.xml</path>
  <properties>
    <name>PME</name>
    <path>Observer.PME.Properties.xml</path>
  </properties>
  </machine>
</machines>
```

2.6.2 Cas 1 : Observation de l'application de paiement EMV

Le protocole que nous avons utilisé est basé sur trois points principaux :

– l'application carte et sa documentation. En utilisant la documentation, nous pouvons modéliser certaines parties de l'application. L'expérience permet d'avoir des propriétés écrites dans le langage défini précédemment de manière plus

précise dans ce cas. Nous avons utilisé une carte de paiement EMV certifiée par un laboratoire de test.

– l'application pour faire la transaction, à l'origine de la communication APDU, la carte étant passive. Nous avons utilisé le framework WSCT avec le plugin EMV explorer permettant de faire la transaction EMV avec la carte à puce.
– l'observateur est une application capable de récupérer la communication APDU et de détecter un mauvais comportement de l'application de la carte à puce. Il compare la communication APDU transmise avec la configuration de l'outil (les comportements globaux et locaux définis dans le fichier xml).

Voici un exemple de machine à états partielle, décrite plus haut en exemple. La figure 2.19 présente une illustration de la machine d'états complète à observer pour suivre le comportement global de l'application. L'observation permet de connaître l'état actuel de l'application. Cette application possède 5 états possibles et de nombreuses transitions possibles.

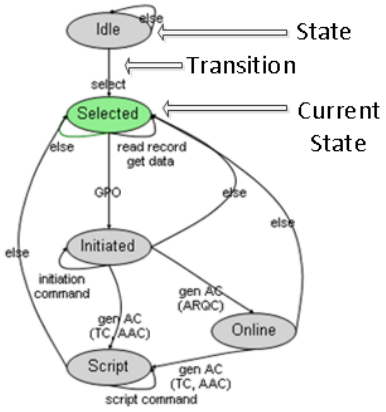

FIGURE 2.19 – La machine à états de l'application de paiement EMV

Pour définir la machine à états, nous avons besoin de deux éléments : "cardstate" et "transition". Le listing 2.5 montre la machine à états partielle. L'état "selected" est l'état de l'application une fois l'application sélectionnée (la commande de sélection a été acceptée) et "initiated" lorsque la commande GPO (initiation de la transaction effectuée) acceptée.

Listing 2.5: Machine à états partielle de l'application de paiement

```
<?xml version="1.0" encoding="utf-8"?>
<machine>
 <cardstate name="Selected" x="100" y="120" />
```

```
<cardstate name="Initiated" x="100" y="230" />

<transition text="GPO">
 <from name="selected" dir="Down" />
 <to name="initiated" dir="Up" />
 <and>
  <instruction instruction="0x80A8"/>
  <status status="0x9000"/>
 </and>
</transition>

<transition text="init. command">
 <from name="initiated" dir="Down" />
 <to name="initiated" dir="Left" />
 <or>
  <and>
   <instruction instruction="0x0084"/>
   <status status="0x9000"/>
  </and>
  <and>
   <instruction instruction="0x80CA"/>
   <status status="0x9000"/>
  </and>
  <and>
   <instruction instruction="0x0020"/>
   <or>
     <status status="0x9000"/>
     <status status="0x6983"/>
     <status status="0x63C0" mask="0xFFF0"/>
   </or>
  </and>
  <and>
   <instruction instruction="0x00B2"/>
   <status status="0x9000"/>
  </and>
 </or>
</transition>

<transition text="else" offsetX="70">
 <from name="initiated" dir="Right" />
 <to name="selected" dir="Right" />
</transition>
</machine>
```

Voici un exemple de propriété, la propriété P3 décrite plus haut. Cette propriété représente le comportement entre les états "selected" et "initiated" en utilisant une seule commande, Get Processing Options (GPO). Le P3 de la propriété est définie sur trois cycles d'horloge, à savoir trois couples APDU. Le fichier XML exposé dans le listing 2.6 est la propriété P3 utilisable par l'outil d'observation.

Good behavior with three GPO commands :

$$P_3 : ((SW1(1) = 90)\ and\ (SW2(1) = 00)\ and\ (SW1(3) \neq 90)\ and\ (SW2(3) \neq 00)\ and\ (SW1(5) = 90)\ and\ (SW2(5) = 00))\ or\ \overline{(INS(0) = A8)\ and\ (INS(2) = A8)\ and\ (INS(4) = A8)}$$

Listing 2.6: Propriété P3

```
<property name="Third GPO">
 <step>
  <instruction instruction="0x80A8"/>
  <status status="0x9000"/>
 </step>

 <step>
  <instruction instruction="0x80A8"/>
  <nor>
   <status status="9000"/>
  </nor>
</step>

<step>
 <instruction instruction="0x80A8"/>
 <require>
  <status status="0x9000"/>
 </require>
</step>

</property>
```

Un outil est créé grâce au framework WSCT et se lance parallèlement de la transaction (nominale ou de test) afin d'observer la communication transitant entre les deux éléments et détecter une anomalie présente dans l'application. La figure 2.20 expose la preuve de concept créée pour l'application de paiement EMV.

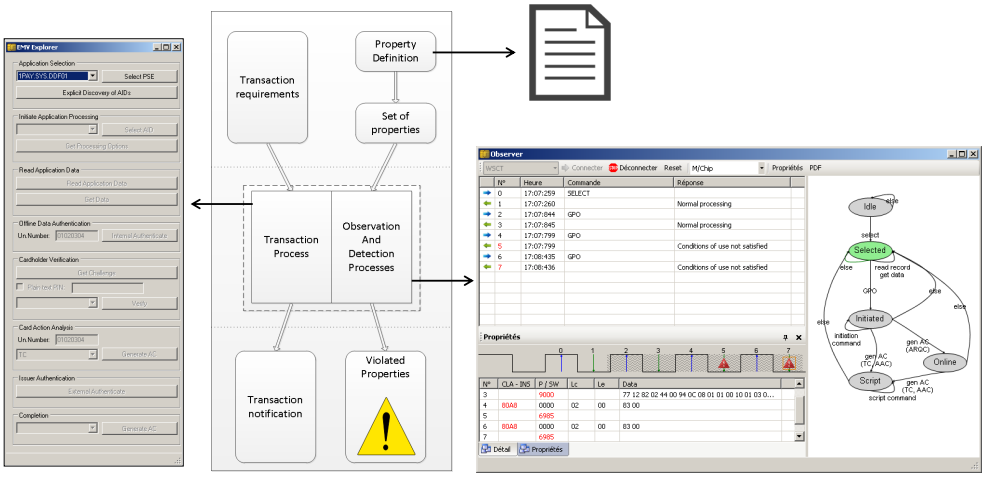

FIGURE 2.20 – Illustration de l'outil développé

2.6.3 Cas 2 : Analyse d'une application PME

Dans un second temps, nous avons utilisé la méthode sur une application de type Porte Monnaie Electronique. L'analyse est décrite dans cette section.

Les éléments minimaux pour débuter l'étude sont : une implémentation de référence, une implémentation cible et un ensemble de fichiers de configuration (correspondant à la représentation de la machine à états, l'ensemble des comportements locaux) sont les entrées du système d'analyse. Un terminal de test a été créé afin d'appliquer des cas de test à l'application, ce qui constitue la base de l'analyse. Pour analyser les raisons de cette erreur, le système utilise les propriétés violées observées par l'outil.

La première étape consiste en la représentation de l'application, donnée par la machine à états de la figure 2.21. Le fichier XML 2.7 permet de représenter la première partie de l'application

Listing 2.7: Machine à états partielle de l'application Porte Monnaie Electronique

```
<machine>
  <cardstate name="Idle" />
  <cardstate name="Selected" />
  <cardstate name="Verified" />
  <cardstate name="Blocked"/>


  <transition text="else">
    <from name="idle" dir="Right" />
```

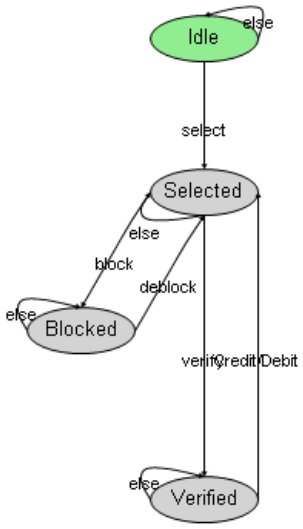

FIGURE 2.21 – La machine à états de l'application PME

```
    <to name="idle" dir="Up" />
  </transition>

  <transition text="select">
    <from name="idle" dir="Down" />
    <to name="selected" dir="Up" />
    <and>
      <or>
        <status status="0x6283" />
        <status status="0x9000" />
      </or>
      <instruction instruction="0x00A4" />
    </and>
  </transition>
</machine>
```

Afin de stresser l'application, un terminal de test est créé. Il applique plusieurs cas de test écrit grâce aux fonctionnalités de WSCT pour chaque étape de développement du système. Un exemple est fourni dans le listing 2.7.

Listing 2.8: Définition d'un cas de test

```
testPlan test1 = createTestPlan(1, "PinCode approval");
test1.addCommandResponse(new CommandAPDU(0x00, 0xA4, 0x04, 0, 0x06, new
    byte[] { 0xf2, 0x00, 0x00, 0x00, 0x00, 0x51, 0x00 }), new
   ResponseAPDU(new byte[] { 0x90, 0x00}));
```

```
test1.addCommandResponse(new CommandAPDU(0x90, 0x20, 0x0, 0, 0x04, new
    byte[] { 0x01, 0x02, 0x03, 0x04 }), new ResponseAPDU(new byte[] { 0
    x90, 0x00}));
addTestplanToStep(test1, step2);
```

Pour étudier l'ajout de l'approche sur une méthode d'évaluation tel que fuzzing, nous avons utilisé l'application de porte-monnaie électronique appelée PME. Afin de définir nos tests, nous avons utilisé :

- 4 implémentations : l'une est l'implémentation de référence et les autres contiennent des erreurs (vérification défectueuse du porteur par une mauvaise gestion du montant durant un paiement, une mauvaise gestion du code PIN et une mauvaise gestion du blocage de la carte en cas de code PIN erroné)
- 2 scripts de transaction : ces scripts de type plan de tests permettent de détecter la présence d'une anomalie (en vérifiant les réponses de l'application de référence et les réponses de l'implémentation cible)
- 5 propriétés : nous utilisons un nombre limité de propriétés afin de montrer la faisabilité de la méthodologie sur l'analyse de l'application Porte Monnaie Electronique. Chaque propriété représente un comportement local de l'application.

En utilisant la documentation de l'application et l'expérience sur ce sujet au sein du laboratoire, nous avons défini des propriétés en ayant pour objectif la détection d'une anomalie et son analyse : le blocage de la carte en cas de vérification erronée du code PIN. Plusieurs propriétés sont définies dans le listing 2.9.

Listing 2.9: Exemples de propriétés de l'application PME

```
<properties>
 <property name="blocageOK">
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170"/>
  </step>
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170" />
  </step>
 </property>

 <property name="blockOneLeft">
  <step>
    <instruction instruction="0x9020"/>
```

```
    <status status="0x9111" />
  </step>
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170" />
    <nor>
      <cdata cdata="01020304"/>
    </nor>
   </step>
 </property>

 <property name="block">
   <step>
    <instruction="0x9020" />
    <nor>
     <status="0x9000" />
    </nor>
   </step>
   <step>
    <instruction="0x9020" />
   <nor>
     <status="0x9000" />
    <status="0x9170" />
   </nor>
    <call method="control"/>
   </step>
 </property>
</properties>
```

Un compteur permettant de bloquer la carte (et ainsi éviter une attaque de type brute force) est présent sur l'application. Ces trois propriétés permettent de documenter la présence d'une anomalie en donnant plus d'informations sur les causes de cette anomalie.

- La première indique si le blocage de la carte est réellement pris en compte par l'application ou ce mécanisme est juste affiché par les réponses. Une réponse indiquant un blocage de la carte implique que les commandes suivantes sont suivies par une réponse indiquant le blocage de la carte (en particulier, pour deux commandes de suite dans ce cas).
- La deuxième permet de s'assurer que le blocage est activé lorsque le compteur arrive à zéro.
- La dernière permet de savoir si le compteur décrémente correctement. Une fonction externe, appellée grâce à l'élément de type "call" dans notre propriété permet le traitement des valeurs.

Les résultats sont fournis sur les quatre implémentations suivantes :

- A : sans erreur, il s'agit de l'implémentation de référence
- B : avec une erreur sur le compteur d'essais
- C : avec une erreur sur le blocage de l'application
- D : implémentation possédant une backdoor, un second code PIN acceptable

Nous comparaisons les résultats sur les quatre implémentations en comparant les réponses aux plans de test et la présence de violation de propriétés sur le tableau 2.3. OK signifie que tous les plans de test sont passés avec succès ou qu'aucune propriété n'est violée. Echec signifie qu'au moins un cas de test a échoué ou qu'une propriété est violée.

TABLE 2.3: Comparaison des résultats

	A	B	C	D
Test	Ok	Echec	Echec	OK
Property	Ok	Alerte	Alerte	Alerte

Les propriétés sont génériques et simples (elles sont définies à partir de couples commande/réponse). Un plan de test consiste à rejouer la transaction depuis le début. En outre, dans les cas A, B et C, nous concluons que la détection par le test est équivalente à la détection des violations de propriétés. Les plans de tests révèlent si l'application est correcte ou non mais l'analyse de la communication et la violation des propriétés permet de connaître l'emplacement exact de l'anomalie (une description est ajoutée à chaque propriété). Toutefois, en ce qui concerne le cas D, on obtient une détection supplémentaire. Ceci permet d'effectuer le test et de mentionner que la vérification a révélé des erreurs d'implémentations non critiques.

2.6.4 Discussion

Nous obtenons alors une méthodologie configurable par l'utilisateur (définition des propriétés et de la machine à états à partir de documentation). La question de la génération de la collection de propriétés utilisée en entrée de notre système est une problématique importante pour obtenir une méthodologie efficace. Nous ne pouvons nous permettre de générer ces propriétés de manière manuelle. Il faudra aussi veiller à avoir la possibilité d'utiliser notre outil avec tout type de terminal et pas uniquement des terminaux internes à WSCT.

2.7 Conclusion

Nous avons dressé un état de l'art des méthodes académiques et de certains outils industriels de test. Il existe de nombreuses méthodes demandant l'accès au code source, ce que nous ne nous permettons pas. Une documentation sera bien sûr nécessaire pour arriver à notre fin. Cependant, à terme, l'outil générique proposé est destiné à être utilisé rapidement et simplement par un professionnel du développement ou du test d'une application carte qui par conséquent pourra configurer notre outil. L'environnement de développement sera basé sur le framework WSCT qui permet de travailler de manière efficace et modulaire sur les éléments ISO7816 mais aussi au niveau applicatif. La méthode étudiée et les différentes étapes traitées pour aboutir à une preuve de concept seront exposées dans le prochain chapitre.

Nous avons présenté dans ce chapitre le coeur de la méthodologie de détection et d'analyse d'anomalie sur application carte. Cette méthodologie permet une analyse d'une application carte en boîte-noire, automatique, générique et modulaire afin de vérifier sa conformité vis à vis de spécifications. Un langage compréhensible et utilisable facilement a été créé. Cependant, il manque l'appréciation de l'oracle, l'élément qui détient la véritié du comportement de l'application, qui prend la forme d'une collection de propriétés dans notre étude. Comment générer l'oracle pour notre méthodologie ? Ce sujet constituera l'objet du prochain chapitre.

Chapitre 3

De la génération d'oracle aux usages

Ce chapitre présente la génération de l'oracle, élément possédant la vérité sur le comportement d'une application sur carte à puce, utilisé dans le contexte de la méthodologie d'évaluation proposée dans cette thèse. La génération des propriétés est permise par l'utilisation d'un algorithme génétique. Cette méthode configurable est automatisable via une technique de Fuzzing. Pour finir, nous traitons les cas d'usages de cette méthodologie : trois cas sont exposés.

Sommaire

3.1 Introduction

Comme nous travaillons dans un contexte boîte noire, l'oracle ne peut pas être une implémentation ou un modèle formel, qui nécessitent plus d'informations qu'accessibles dans notre cas.

Nous construisons l'oracle comme une collection de comportements locaux et attendus de l'application. Ceci consiste en la génération de séquences à partir de la communication APDU. Après avoir listé les études les plus pertinentes, nous présentons deux méthodologies. Afin d'automatiser de manière intelligente la génération de

ces propriétés, nous proposons l'usage des algorithmes évolutifs dans une troisième approche. Enfin, nous traitons des cas d'usage. Notre approche pragmatique nous permet de penser la méthodologie pour une utilisation à différentes phases du cycle de vie de la carte à puce.

Les méthodes V&V consistent à vérifier si un système suit les spéfications définissant son comportement théorique. Dans notre méthode, tout comme dans les méthodes de test, il est question de déterminer les différences entre les comportements réél et théorique de l'application carte. L'élément possédant la vérité grâce à laquelle on peut dire si oui ou non le comportement est correct s'appelle l'oracle. L'idée est d'utiliser un modèle du système, modèle qui peut être créé de différentes manières. Hormis la génération des entrées à appliquer au système, notamment pour les techniques de test, la génération automatique de l'oracle est un élément primordial pour l'automatisation de méthodes de vérification d'un système [134]. La figure 3.1 illustre le périmètre de cette partie.

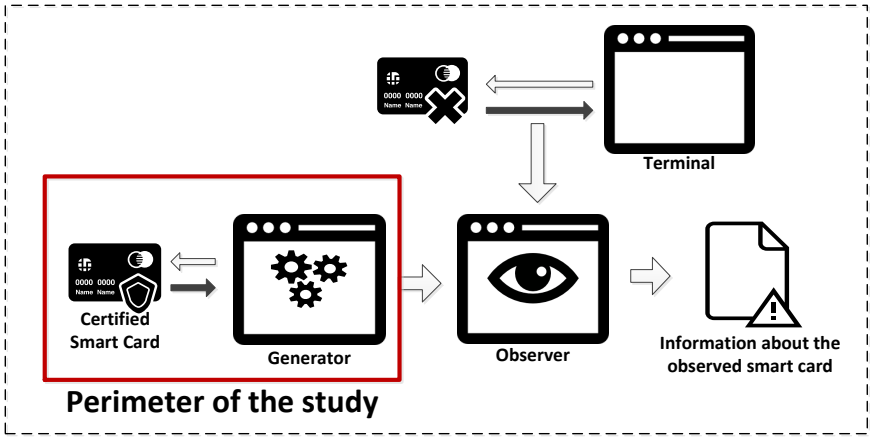

FIGURE 3.1 – Périmètre de l'étude de la génération d'oracle

3.2 Etat de l'art

Particulièrement, dans notre domaine, un modèle peut être généré de différentes façons. Premièrement, une étude expose la génération d'un modèle à partir de la documentation d'un programme mais cette méthode est limitée par les valeurs des données inaccessibles [147]. Plus qu'une simple documentation, un modèle peut aussi être une seconde version d'un programme (i.e., une version validée du programme cible) [97]. Si nous n'avons pas accès au code source, une solution est de créer un modèle à partir d'une application certifiée en mode boite-noire, on parle alors de

rétro-ingénierie. La méthode consiste à envoyer des commandes afin de recréer le modèle [84]. Une autre méthode consiste à générer des affirmations [89]. Ces études traitent surtout, en plus de la génération de la structure de l'oracle, du problème de la génération des données utilisées dans l'oracle ([148], [149]). Les tableaux 3.1 et 3.2 permettent de comparer ces méthodes.

TABLE 3.1: Méthodes de génération d'oracle (boîte noire)

Référence	Méthode	Utilisation	Date
Peters and Pranas	A partir de la documentation	Facile mais limitée	1998
Aarts *et al.*	Rétro-ingénierie	Facile	2013
Pacheco and Ernst	Ensemble de tests	Nécessite un modèle	2013

TABLE 3.2: Méthodes de génération d'oracle (boîte blanche)

Référence	Méthode	Utilisation	Date
Alimi *et al.*	Implémentation de référence	Nécessite le développement	2014
Loyoal *et al.*	basée sur les tests	Utilise le framework DODONA	2014

Pour cette étude, nous voulions étudier deux cas : l'utilisation de la documentation dans un premier temps de la génération et l'utilisation de la rétro-ingénierie sur un système certifié. Finalement, la solution retenue est basée sur la rétro-ingénierie, l'optimisation via des algorithmes génétiques et un outil semi-automatique configurable par l'utilisateur ayant connaissance de la documentation. La figure 3.2 expose les méthodes de génération étudiées.

Les méthodes sont :
- la méthodologie théorique : A partir de la documentation, il s'agit d'automatiser de façon triviale la génération des propriétés.
- la méthodologie de laboratoire : En stressant l'application, il s'agit de modéliser le système en automatisant l'envoi de scripts de transaction à la carte à puce.
- la méthodologie directe : A partir de logs de transaction, il s'agit de reconstruire la collection de propriétés associée à l'application observée.

Nous illustrons nos contributions sur une application de paiement de type PME (Porte Monnaie Electronique) dont la figure 3.3 représente le comportement global.

Les transitions sont :
- 0 : Pas d'évolution
- 1 : Sélection de l'application

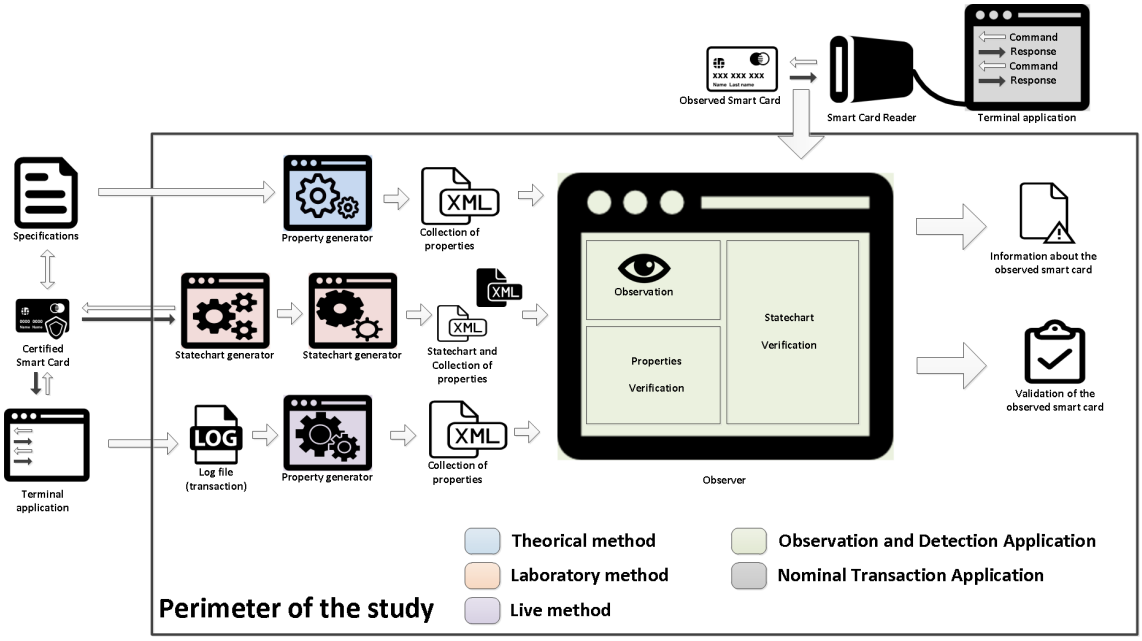

FIGURE 3.2 – Comparaison des approches (entrée/sortie des générateurs)

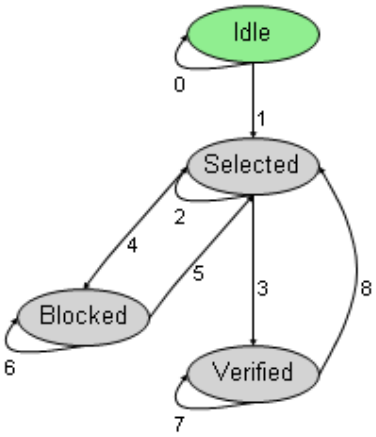

FIGURE 3.3 – Machine à états de l'application PME

- 2 : Pas d'évolution
- 3 : Vérification du Code PIN
- 4 : Blocage de la carte
- 5 : Déblocage de la carte
- 6 : Pas d'évolution
- 7 : Pas d'évolution
- 8 : Effectuer une opération remet l'application dans son état selected

3.3 Méthodes de génération d'oracle

Nous exposons les deux premières approches pour obtenir l'oracle de l'application carte.

3.3.1 Méthode de génération théorique ou triviale

Description de l'approche

Dans un premier temps, nous avons travaillé sur une méthode de génération de propriété à partir de la documentation. Il s'agit de connaître l'ensemble du comportement de l'application carte afin de savoir quelles commandes (associées aux données) sont acceptables alors que l'application est dans un certain état. La méthode dite triviale permet de lister les comportements possibles sur 1 à n couples commandes/réponses à partir d'une liste de transitions possibles visibles sur le listing 3.4. Chaque ligne représente une transition (un numéro, un état de départ, un état d'arrivée et la configuration du couple commande/réponse). L'idée est de partir de la machine à états connue dans les spécifications et sa configuration (états possibles, transitions). Il s'agit typiquement d'une problèmatique de parcours de graphe.

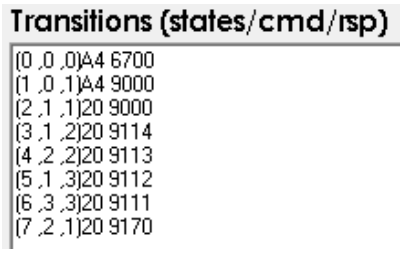

FIGURE 3.4 – Exemple de transitions de base possibles

Filtrage

Grâce à cette génération, nous obtenons tous les comportements possibles, ce qui fait un nombre conséquent. Avec un système de filtrage, nous pouvons réduire le nombre de propriétés. Il s'agit d'un système de tri permettant de sélectionner les meilleures propriétés, et ainsi écarter les propriétés non pertinentes qui n'apportent aucune information sur l'évolution du système. L'intérêt d'une propriété est évaluée par le nombre d'états visités et le nombre de transitions comme exposé dans la figure 3.5. Des compteurs (des états visités par une transition sur plusieurs couples commande/réponse et du nombre de transitions unitaires impliquées) sont ajoutés afin de filtrer les résultats.

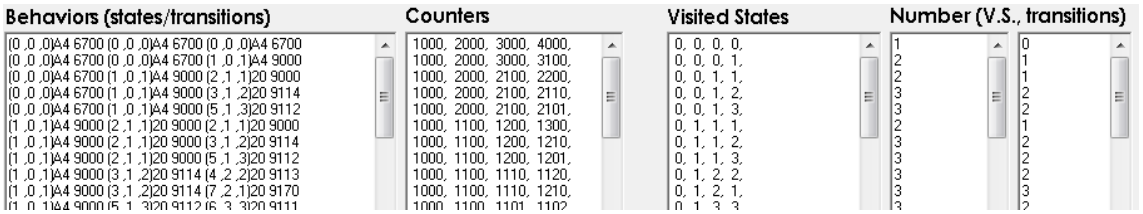

FIGURE 3.5 – Filtrage

Cette méthode consiste en l'automatisation de la génération des chemins possibles d'une machine à états. Elle nécessite une connaissance accrue des spécifications (machine à états et transitions) et permet de rendre automatique la génération de propriétés sur n couples commandes réponses de manière triviale. Le filtrage est assez simple et permet de réduire drastiquement le nombre de propriétés. Cependant, de nombreuses propriétés ainsi générées sont toujours superflues et il est difficile de savoir quelles sont les propriétés dites critiques c'est-à-dire d'évaluer de manière détaillée la pertinence des propriétés. Cette méthode nécessite trop d'implication de la part de l'utilisateur pour un rendu trop trivial, elle a donc été écartée.

3.3.2 Méthode de génération de laboratoire basée sur la rétro-ingénierie et la constitution d'un modèle

Description de l'approche

Des études traitant de la rétro-ingénierie sur les applications carte ont montré la possibilité de générer un modèle ([84] ou [109]). La méthode dite "Angluin" permet de générer la machine à états. Nous avons travaillé sur la possibilité de générer des propriétés avec cette méthode basée sur deux actions du module élève qui est complété par un module nommé professeur.

- L'élève demande au professeur si un mot appartient au langage correspondant à l'automate réél
- L'élève émet une hypothèse en construisant un automate hypothétique de l'application

Le professeur retourne un contre-exemple dans le cas où l'automate hypothétique ne correspond pas au comportement de l'application et donc à son automate réel. Un nouvel état est alors créé.

Limites

Cette stratégie comporte plusieurs limites. Les possibilités pour les mots sont configurées préalablement par l'utilisateur ce qui évite de bloquer la carte, sans cette précaution, la carte peut vite se bloquer (limitation du nombre d'APDU des cartes). Pour savoir si l'automate hypothétique correspond à l'automate réél, une série de plan de tests est générée (1000 tests contenant chacun 10 à 50 couples APDU). On revient donc vite au problème de génération du vecteur d'entrée d'une méthode de test et non de l'oracle à proprement parlé. Il est bon de noter que nous ne pouvons obtenir qu'une approximation de l'automate réél.

Cette méthode permet d'appréhender le sujet par la rétro-ingénierie, c'est-à-dire que l'on va partir d'un système existant certifié afin de modéliser le comportement attendu de l'application carte. Cependant, la méthode permet d'avoir des résultats pertinents mais son exhaustivité rend complexe la génération de l'oracle car il s'agit de générer dans un premier temps une série de tests conséquente et pertinente.

3.3.3 Méthode de génération directe et intelligente basée sur la rétro-ingénierie en mode boîte noire

Nous nous sommes inspirés des deux méthodes afin de penser une méthode de génération automatisée, intelligente et de complexité raisonnable par une configuration possible par l'utilisateur. La configuration de la méthode inspirée de la génération à partir de la documentation permet de guider la génération automatisée par des algorithmes génétiques permettant une génération intelligente et pertinente des propriétés. Nous partons d'un ensemble de transactions. Pour résumer, notre méthode est basée sur trois points principaux :

1. mode boîte noire (assertions basées sur les entrées et sorties du système)
2. configurable pour guider la génération automatique d'une application précise
3. basée sur la mutation par l'utilisation d'algorithme génétique

La figure 3.6 rappelle le positionnement de cette étude vis à vis de la méthodologie générale.

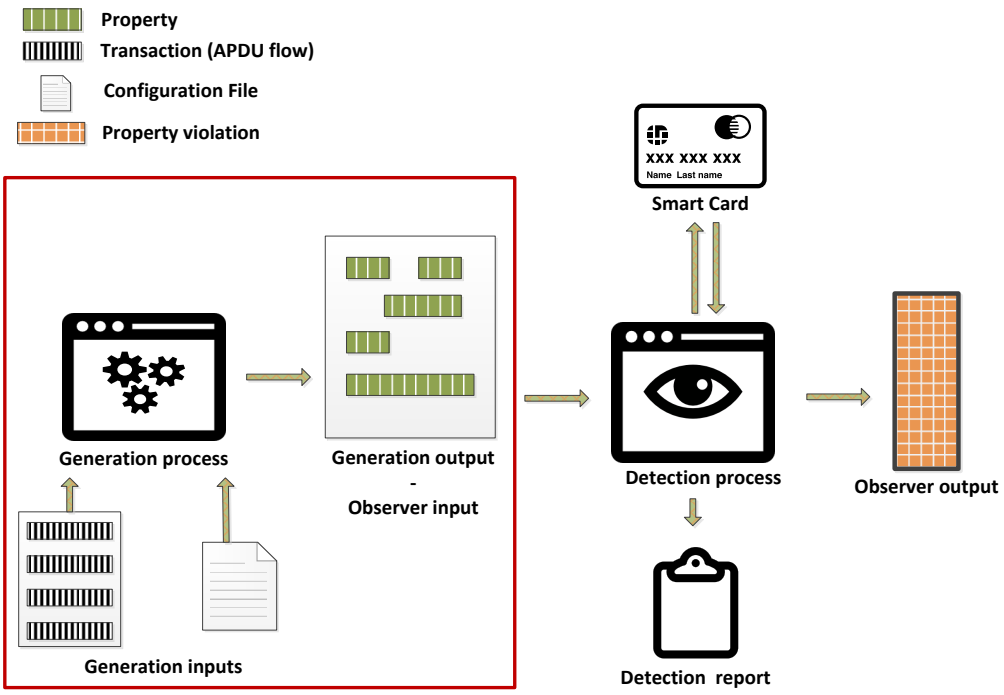

FIGURE 3.6 – Récapitulatif du positionnement de la méthode de génération proposée

Description de l'approche

L'approche que nous proposons permet d'automatiser la génération de propriétés pertinentes. Afin d'y arriver, nous générons aléatoirement des propriétés, puis nous selectionnons les plus pertinentes grâce à un ensemble de logs de transactions entre le terminal et l'application certifiée. Ces logs constituent une base de la connaissance du comportement de l'application qui peut être complétée par du Fuzzing ([97] ou [96]).

La génération et la sélection de propriétés sont permises par un algorithme génétique [150]. Un algorithme génétique est utilisé pour l'optiminisation en s'appuyant sur la théorie de l'évolution naturelle. Concernant ses avantages, nous pouvons dire qu'un nombre conséquent de propriétés possibles peut alors être généré à chaque lancement de l'algorithme. Nous créons une population aléatoire de propriétés, puis nous mesurons les performances grâce à une méthode de scoring. Plus le score est proche de zéro, plus l'individu est représentatif du comportement de l'application. On soustrait des points à une valeure initiale lorsque les propriétés constituant un individu sont en accord avec les logs de transactions ou lorsque les données utilisées par ces propriétés sont présentes dans le fichier de configuration mis en place par

l'utilisateur. Une fois que la population est évaluée, nous gardons les meilleures propriétés pour la prochaine génération. Nous remplaçons alors les propriétés exclues par des nouvelles créées aléatoirement (par mutation, fusion, etc). Après plusieurs générations, nous arrêtons l'évolution des propriétés et les meilleures d'entre elles sont ajoutées à la base de propriétés sélectionnées qui forment ainsi l'oracle (la vérité sur le comportement attendu de l'application).

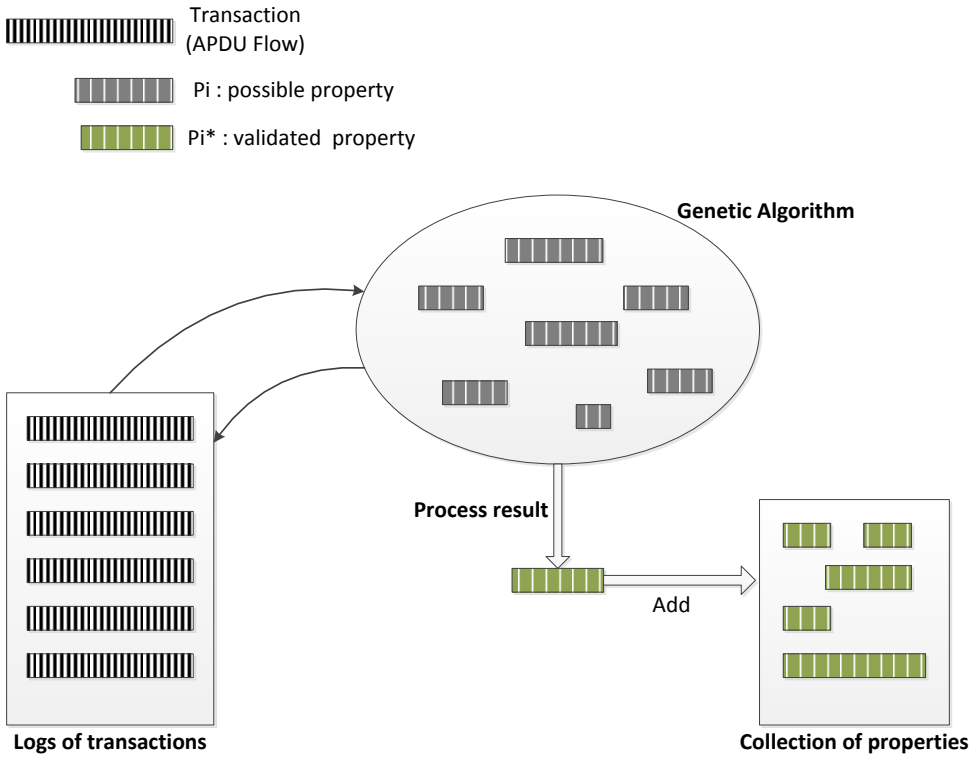

FIGURE 3.7 – Génération d'une collection de propriétés

Sur la figure 3.7, nous pouvons voir un schéma représentant l'approche de génération de propriétés à base d'algorithme génétique qui constitue le coeur de la méthode. L'entrée pour l'algorithme génétique est un fichier de transactions (un fichier de log), créé à partir d'une communication entre le terminal (nominal ou test) et une carte à puce certifiée. Il s'agit donc d'une séquence de couples commande/réponse APDU, définie comme suit :

```
-> APDU command
<- APDU response

-> :00|A4|04|00|A0 00 00 00 04 10 10|00
<- :90|00|6F 3F 84 07 A0 00 00 00 04 10 10
```

```
 A5 34 50 0A 4D 41 53 54 45 52 43 41 52 44
 87 01 02 5F 2D 04 66 72 65 6E 9F 11 01 01
 9F 12 0A 4D 41 53 54 45 52 43 41 52 44 BF
 0C 0A DF 60 02 0B 14 9F 4D 02 0B 14
-> :80|CA|9F|36||00
<- :6A|88|
-> :80|CA|9F|13||00
<- :6A|88|
-> :80|CA|9F|17||04
<- :90|00|9F 17 01 03
-> :80|CA|9F|4F||13
<- :90|00|9F 4F 10 9F 02 06 9F 27 01 9F 1A
 02 5F 2A 02 9A 03 9C 01
-> :80|A8|00|00|83 00|00
<- :90|00|77 0E 82 02 78 00 94 08 08 01 03
 00 10 01 04 01
-> :00|88|00|00||00
<- :67|00|
```

A partir de ce document, l'algorithme de génération extrait les valeurs possibles des différents champs (CLA, INS, etc) et génère des propriétés possibles. La fonction d'aptitude se charge d'évaluer la pertinence de ces propriétés par rapport aux données récupérées dans le fichier log et le fichier de configuration défini par l'utilisateur.

Une fois qu'un ensemble de propriétés est sélectionné, un fichier de type xml, dont le template est donné ci-dessous dans le listing 3.1, est produit afin d'être compatible avec notre outil d'observation.

Listing 3.1: Template d'une propriété

```
<properties>
  <property>
    <step>
      <instruction instruction="XXXX"/>
      <parameters parameters="XXXX" />
      <status status="XXXX"/>
    </step>
    <step>
      <instruction instruction="XXXX"/>
      <status status="XXXX"/>
    </step>
  </property>
```

Gestion de paramètres

La génération d'une collection de propriétés pertinentes est permise par l'utilisation de l'algorithme génétique, basée sur quatre fonctions principales : la génération, la mutation et le croisement permettent de manipuler la population de propriétés et la fonction d'évaluation qui permet de sélectionner les meilleures propriétés. Dans notre étude, nous avons ainsi défini les fonctions suivantes :

- génération de la population initiale : choisie aléatoirement à partir du fichier de log.
- sélection des individus : aléatoire basée sur le score des individus. Nous supprimons les plus faibles individus.
- mutation : une petite chance (0.1 à 1%) de modifier la composition d'un individu afin d'éviter une convergence vers un extremum local.
- croisement : mélange d'individus choisis aléatoirement afin de créer un individu meilleur.
- évaluation : évaluation des propriétés via leur composition (adéquation de la réponse de l'application de référence).

A chaque génération, toutes les propriétés sont évaluées avec la fonction d'évaluation (ou d'aptitude), et seules les propriétés avec le meilleur score sont gardées pour la prochaine génération. Le score d'une propriété est calculé à partir des logs de transaction et des valeurs ciblées contenues dans une base de donnée configurable par l'utilisateur. La population compte 100 individus, et 10% sont sélectionnées pour faire partie de la prochaine génération. Après 10 générations, nous gardons les meilleures propriétés de la population.

Résultats

Un ensemble de propriétés est alors créé au format XML. Nous pouvons réécrire ces propriétés dans le formalisme vu dans les précédentes sections. La première propriété signifie que si la sélection de l'application a échoué ("A4"), alors les commandes GPO ("A8") et VERIFY ("20") ne peuvent être acceptées.

$$\begin{aligned}((SW1(1) \neq 90)\ and\ (SW2(1) \neq 00)\ and\ \\ (SW1(3) \neq 90)\ and\ (SW2(3) \neq 00)\ and\ \\ (SW1(5) \neq 90)\ and\ (SW2(5) \neq 00))\ or\ \\ \overline{(INS(0) = A4)\ and\ (INS(2) = A8)\ and\ (INS(4) = 20)}\end{aligned} \quad (3.1)$$

La seconde propriété signifie que nous pouvons sélectionner l'application deux fois de suite. Une amélioration possible est de spécifier les champs possibles (paramètres) pour la sélection.

$$((SW1(1) = 90)\ and\ (SW2(1) = 00)\ and\ (SW1(3) = 90)\ and\ (SW2(3) = 00)\ or\ \overline{(INS(0) = A4)\ and\ (INS(2) = A4)} \quad (3.2)$$

Le temps de calcul est raisonnable, on considère qu'une collection de propriétés de longueur quatre maximum est générée par seconde. Le fichier généré est bien entendu compatible avec l'outil d'observation présenté dans la section précédente. Si nous lançons plusieurs fois le générateur, on peut agréger les résultats. Voici sur la figure 3.8 l'évolution du score de quatre individus au fur et à mesure des incréments.

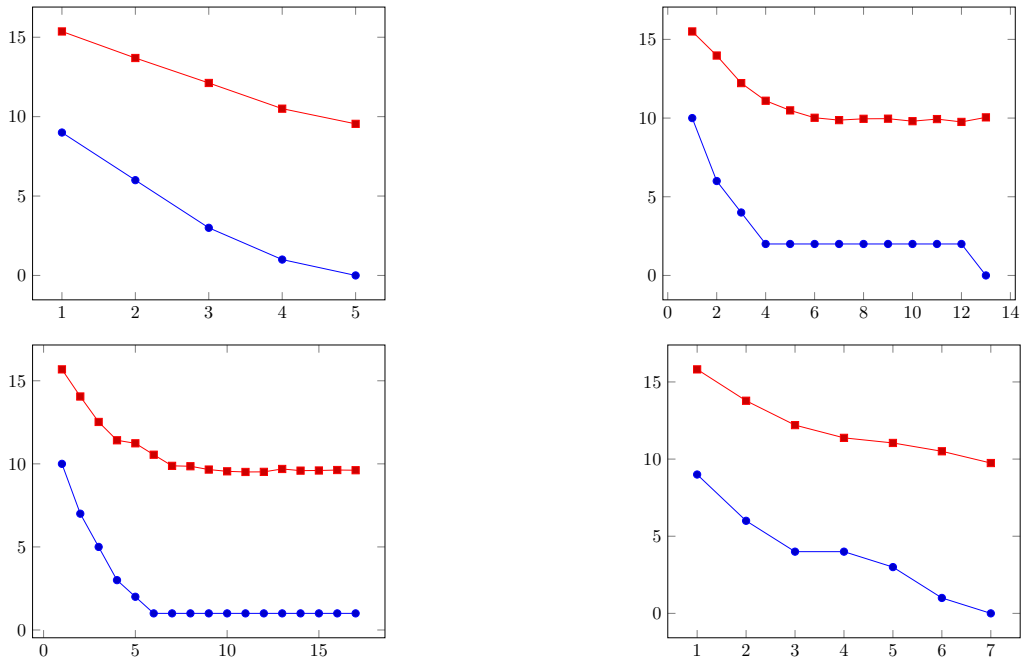

FIGURE 3.8 – Génération de 30 propriétés : Evolution du score de l'individu (la moyenne des scores des propriétés en rouge et le meilleur score en bleu des propriétés selon les itérations de l'algorithme). La fonction d'évaluation, à minimiser, est illustrée par quatre exemples de collections.

Discussion

La génération des propriétés via l'utilisation d'un algorithme génétique est fonctionnelle. Nous pouvons générer autant de propriétés que nécessaire dans un temps

acceptable. Néanmois, certaines améliorations sont possibles. L'efficacité de l'algorithme génétique, c'est-à-dire la pertinence des résultats générés, est donnée par la fonction d'évaluation. Guider cette génération permet de réduire la complexité mais diminue l'exhaustivité des résultats. Il faudrait coupler la génération à partir des logs à l'envoi de commandes à l'application certifiée pour découvrir le comportement de l'application avec des commandes non présentes dans les logs, et permettre ainsi plus d'ouverture quant aux résultats.

3.4 Applications d'usage

Cette méthodologie d'évaluation d'application carte complétée par la méthode de génération automatisée et intelligente de propriétés permet une utilisation dans différents contextes :

- lors de la phase de test, la méthodologie peut être utilisée en parallèle de la méthode utilisée par l'entreprise de certification (par exemple, du test) afin d'obtenir plus d'informations concernant une anomalie et avoir un rapport d'évaluation de meilleure qualité (plus complet et plus sûr) ;
- lors du développement de l'application, l'utilisation de cette méthodologie permet un guidage du développement. On peut ainsi utiliser la méthode dans le cadre d'un enseignement du développement JavaCard par un enseignant. Cet outil permet alors de traiter le développement JavaCard mais aussi la sécurité des applications carte et leur conformité vis à vis des spécifications.

3.4.1 Cas 1 : Outil d'analyse d'une application carte utilisant un terminal externe à WSCT

Prototypage interne de la méthodologie

Dans un premier temps, l'étude est effectuée grâce aux fonctionnalités de WSCT et en particulier par la récupération de la communication APDU. Vous pouvez voir sur la figure 3.9 l'interaction entre les différents modules et la bibliothèque dynamique Winscard.dll. Afin de créer l'outil présenté dans le chapitre 2, nous avons créé deux plugins WSCT. Le premier, le plugin transaction, permet d'effectuer une transaction (nominale, fuzzing, autre). Le second, plugin observateur, permet d'observer indépendamment du terminal logiciel effectuant la transition, la comunication entre la carte à puce et le terminal. L'observateur va, à la volée, vérifier les propriétés préalablement définies et incorporées au plugin afin de s'assurer du comportement de la carte.

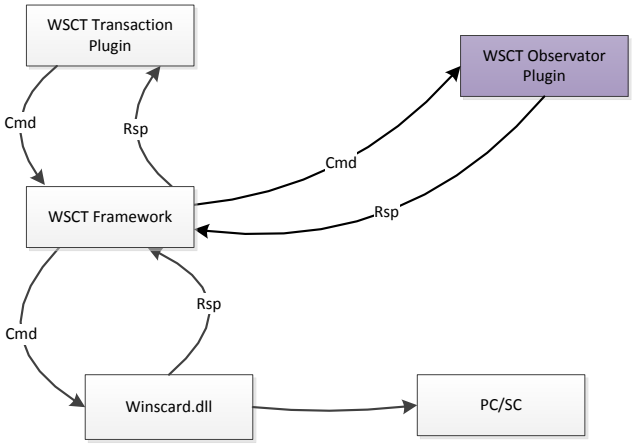

FIGURE 3.9 – Man in the middle version interne à WSCT

La première version de l'outil consistait en la définition manuelle de propriétés et sa vérification automatique. Il s'agissait de mettre en place un outil observant en parallèle la communication. Comme visible sur la figure 3.10, l'observation est faite sur un terminal interne à WSCT (l'explorateur EMV à gauche).

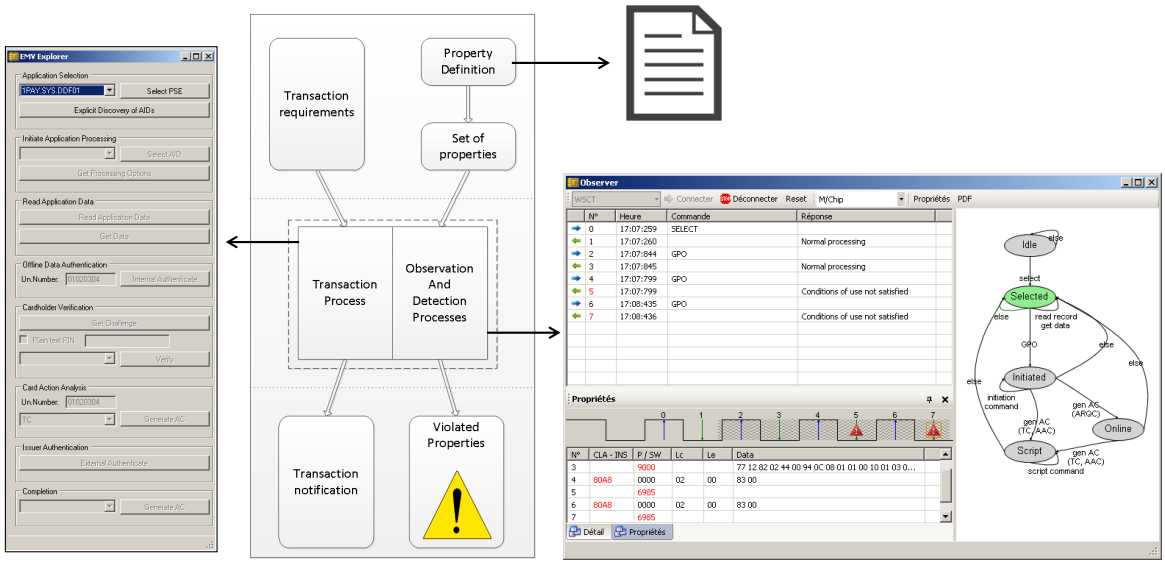

FIGURE 3.10 – Schéma de l'outil d'observation et des interactions avec le terminal et la carte à puce

Généralisation de la méthode d'observation

Afin d'aller plus loin dans l'usage de la méthodologie en tant qu'outil complémentaire au test, l'observation de la communication APDU avec un terminal externe à WSCT est nécessaire. Des travaux ont été effectués sur une bibilothèque dynamique modifiée et remplaçant Winscard.dll. Cette bibliothèque appelle les fonctions de la bibliothèque native Winscard.dll et récupère l'ensemble de la communication APDU qui transite entre un terminal externe à WSCT afin de le transférer au framework WSCT. Cette technique est exposée sur la figure 3.11.

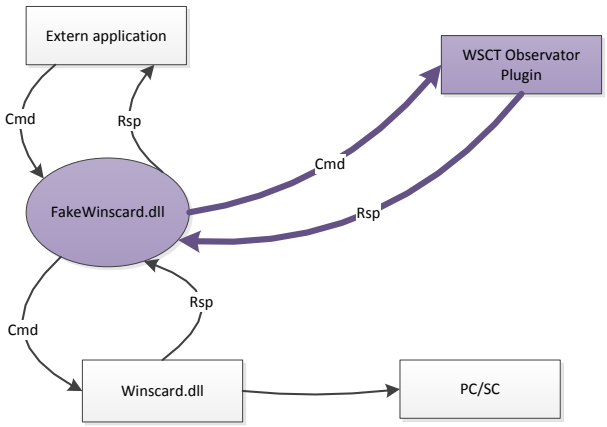

FIGURE 3.11 – Man in the middle version externe à WSCT

La méthode est utilisée dans des études telle que [151] et des outils indépendants sont disponibles ([152] ou [153]). Nous nous sommes inspirés de ces travaux afin de créer notre module d'interception de type bibliothèque dynamique (fichier de type DLL). L'intercepteur fonctionne en trois étapes :

- Le programme de type terminal charge la bibliothèque modifiée (fakewinscard.dll)
- Cette bibliothèque utilise les fonctions permises par la bibliothèque native (winscard.dll)
- Nous pouvons soit enregistrer les communications soit les envoyer sur un canal TCP récupérables par la suite par les fonctionnalités de WSCT.

Un exemple d'utilisation et d'interception est donné par le listing 3.2 et les fonctions permettant le lien entre l'intercepteur de type DLL et le framework WSCT par le listing 3.3. Il s'agit d'une connexion TCP qui permet l'envoi des informations à la volée au framework.

Listing 3.2: Exemple d'appel et d'interception d'une fonction native

```
WINSCARDAPI LONG WINAPI SCardTransmit(IN SCARDHANDLE hCard, IN
    LPSCARD_IO_REQUEST pioSendPci, IN LPBYTE pbSendBuffer, IN DWORD
    cbSendLength, IN OUT LPSCARD_IO_REQUEST pioRecvPci, OUT LPBYTE
    pbRecvBuffer, IN OUT LPDWORD pcbRecvLength)
{
    logMethodBegin(__FUNCTION__, hCard, pioSendPci, pbSendBuffer,
        cbSendLength, pioRecvPci, pbRecvBuffer, pcbRecvLength);
    LONG result = (*Original_SCardTransmit)(hCard, pioSendPci,
        pbSendBuffer, cbSendLength, pioRecvPci, pbRecvBuffer,
        pcbRecvLength);
    logMethodEnd(__FUNCTION__, result, hCard, pioSendPci, pbSendBuffer,
        cbSendLength, pioRecvPci, pbRecvBuffer, pcbRecvLength);
    return result;
}
```

Listing 3.3: Lien avec WSCT

```
#ifndef RESEAU_H
#define RESEAU_H

#include <Windows.h>
void TCPConnect(char *address, int port);
void TcpSendMess(char *message);
void TCPDisconnect(SOCKET id_de_la_socket);


#endif RESEAU_H
```

3.4.2 Cas 2 : Utilisation pour l'enseignement du développement et de l'évaluation d'applet JavaCard

Dans cette partie, l'utilisation de la méthode est illustrée dans le cadre d'un enseignement du développement JavaCard. Pour ce faire, nous utilisons trois environnements : Eclipse permet le développement de l'application, un terminal de test est simulé avec le framework WSCT et permet d'envoyer des plans de test par étape de développement à la carte et enfin un observateur, aussi basé sur le framework WSCT, utilisant le language vu dans le chapitre 2 qui permet l'analyse plus poussée de l'anomalie détectée.

Représentation de l'application cible

La représentation de l'application est l'oracle à utiliser pendant l'évaluation de l'application. Dans cette étude, l'enseignant fournit aux élèves des spécifications mais aussi des plans de test (voir Listing 3.4), une machine à états (voir Listing 3.5) et une collection de propriétés (voir Listing 3.6). Le premier document permet aux élèves de débuter le développement de l'application. Grâce aux éléments supplémentaires, l'étudiant peut valider son développement et se laisser guider par les plans de test, ou plutôt les résultats de ces plans de test. La découverte des erreurs d'implémentations par l'observation des communications et la détection d'anomalie dans celles-ci permet un guidage plus précis par la détection de violation de propriétés. Ceci permet à l'enseignant de faire appréhender la notion de sécurité et d'évaluation d'un système basée sur une méthode de test complété par un outil créé dans un contexte de recherche académique.

Listing 3.4: Création d'un plan de test pour le terminal de test

```
testPlan test1 = createTestPlan(1, "PinCode approval");
test1.addCommandResponse(new CommandAPDU(0x00, 0xA4, 0x04, 0, 0x06, new
    byte[] { 0xf2, 0x00, 0x00, 0x00, 0x00, 0x51, 0x00 }), new
    ResponseAPDU(new byte[] { 0x90, 0x00}));
test1.addCommandResponse(new CommandAPDU(0x90, 0x20, 0x0, 0, 0x04, new
    byte[] { 0x01, 0x02, 0x03, 0x04 }), new ResponseAPDU(new byte[] { 0
    x90, 0x00}));
addTestplanToStep(test1, step2);
```

Listing 3.5: Machine à états partielle de l'application de paiement EMV

```
<machine>
 <cardstate name="Selected"/>
 <cardstate name="Initiated"/>
 <transition text="GPO">
  <from name="selected" />
  <to name="initiated" />
  <and>
   <instruction="0x80A8"/>
   <status="0x9000"/>
  </and>
 </transition>
 <transition text="else">
  <from name="initiated" />
  <to name="selected" />
```

```
 </transition>
</machine>
```

Voici un exemple de propriétés :

Listing 3.6: Quelques exemples de propriétés de l'application EMV

```
<properties>
 <property name="blocageOK">
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170"/>
  </step>
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170" />
  </step>
 </property>

 <property name="blockOneLeft">
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9111" />
  </step>
  <step>
    <instruction instruction="0x9020"/>
    <status status="0x9170" />
    <nor>
      <cdata cdata="01020304"/>
    </nor>
   </step>
 </property>

 <property name="block">
   <step>
    <instruction="0x9020" />
    <nor>
     <status="0x9000" />
    </nor>
   </step>
   <step>
    <instruction="0x9020" />
   <nor>
     <status="0x9000" />
```

```
    <status="0x9170" />
   </nor>
    <call method="control"/>
   </step>
 </property>
</properties>
```

Un compteur permettant de bloquer la carte (et ainsi éviter l'attaque de type brute force) doit être présent sur l'application. Ces trois propriétés permettent de détecter une anomalie et même de spécifier la cause de l'erreur.

- La première permet de savoir si le blocage est effectif et non pas seulement affiché. Une réponse indiquant un blocage de l'application à l'instant t est suivie d'une réponse indiquant un blocage à l'instant t+2.
- La seconde vérifie que l'application retourne bien le message correspondant au blocage (le blocage semble alors être pris en compte) une fois le compteur arrivé à zéro.
- la dernière permet de savoir si le compteur se décrémente correctement. Celle-ci utilise une fonction interne à l'observateur permettant la vérification des valeurs (élément call).

Développement JavaCard

L'élève peut alors développer son application sur eclipse comme illustré sur la figure 3.12. Au fur et à mesure qu'il avance, il peut lancer son application via JCOP (un plugin pour Eclipse nous permettant d'installer l'application sur une carte physique ou de lancer son émulation récupérable par les fonctionnalités de WSCT) .

Connexion entre l'application et le framework WSCT

Ensuite, le framework WSCT peut être lancé afin de se connecter à la carte émulée, ce qui ne change rien pour nous car nous nous focalisons sur la partie logicielle. La figure 3.13 nous donne l'interface de connexion de WSCT.

Terminal de test

Un plugin nommé PME Terminal permet d'appliquer à l'application des plans de test pour chaque étape de développement. Ceux-ci sont bien sûr créés par l'enseignant et doivent correspondre aux spécifications fournies à l'étudiant. Cette méthode est donc une méthode de développement guidée en partie par le test. Sur la figure 3.14, nous pouvons voir que la seconde étape de développement n'est pas validée. En effet, un test n'est pas passé.

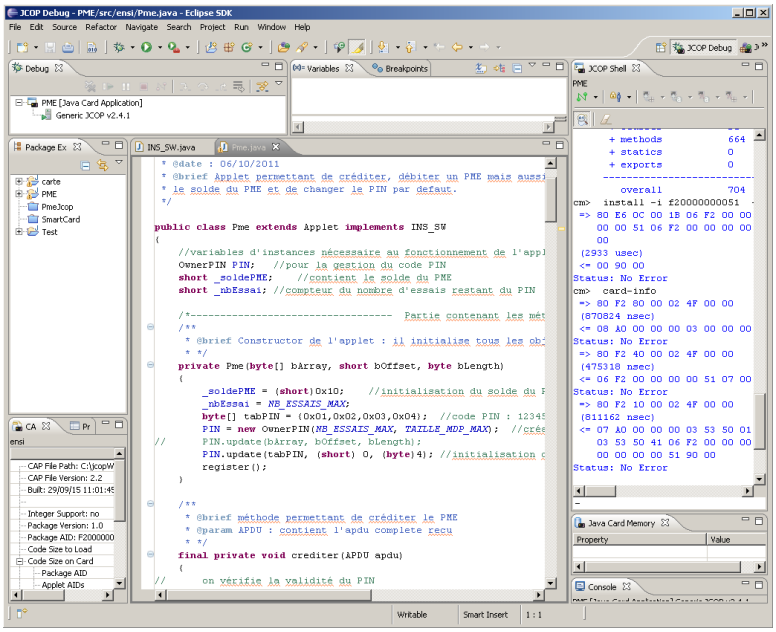

FIGURE 3.12 – Environnement de développement de l'application

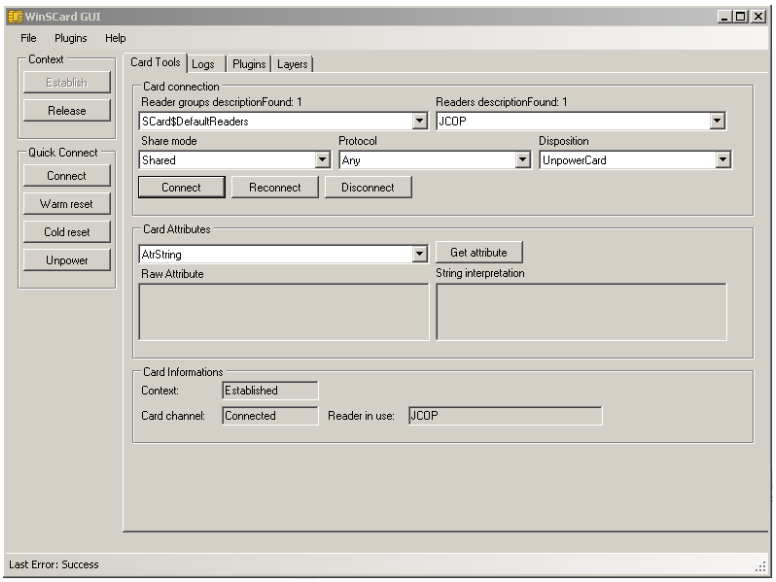

FIGURE 3.13 – Environnement de connexion de WSCT

Analyse des erreurs d'implémentation

Un second plugin, créé dans le cadre de travaux de recherche peut alors être lancé en parallèle afin de détecter les anomalies le plus précisemment possible. Il s'agit de la partie complémentaire que l'enseignant peut générer manuellement ou de manière plus automatique (voir section précédente).

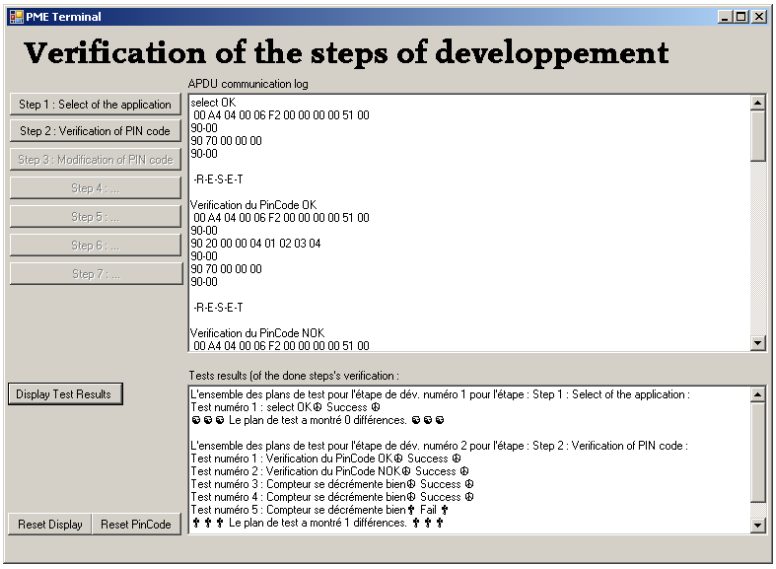

FIGURE 3.14 – Outil pour effectuer le test

Sur la figure 3.15, nous avons une capture du plugin correspondant à la fin de transaction, c'est-à-dire dans notre cas la fin du plan de test. On observe une violation de propriété qui permet de savoir quelle fonctionnalité n'est pas valide. Le blocage de la carte n'est donc pas effectif. Il s'agit bien sûr d'un exemple. Plus l'enseignement est préparé par l'enseignant, plus les propriétés seront précises. Tout est configurable selon la volonté de l'enseignant.

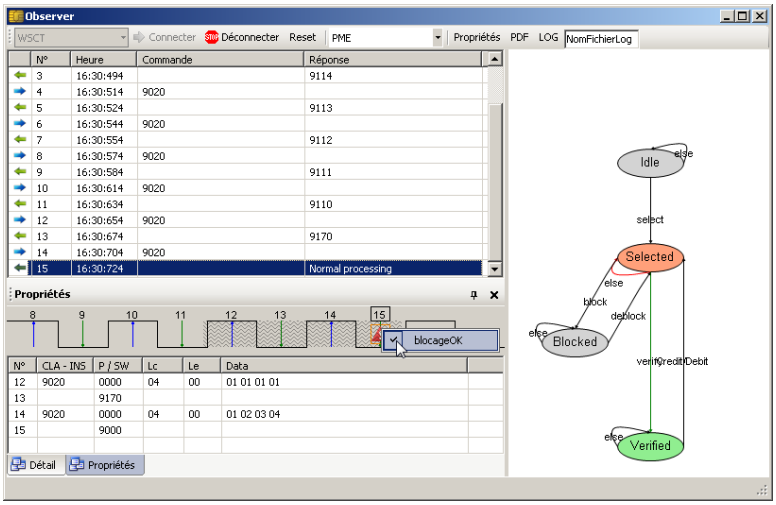

FIGURE 3.15 – Outil de détection d'anomalie

Résultats

Nous obtenons une maquette pédagogique à destination d'une école d'ingénieur pour l'enseignement du développement JavaCard et de l'évaluation d'application. Les figures 3.16 et 3.17 présentent le lien entre l'enseignant et les outils utilisés durant l'enseignement. Au début de l'enseignement pratique, l'enseignant configure les outils et rédige les documents nécessaires. Enfin, l'enseignant utilise les rapports de test et de détection afin de proposer aux élèves un système de notation objectif permettant un retour efficace et personnalisé par étudiant.

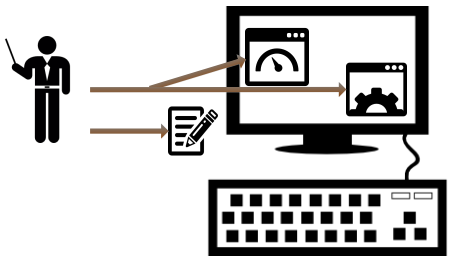

FIGURE 3.16 – Configuration

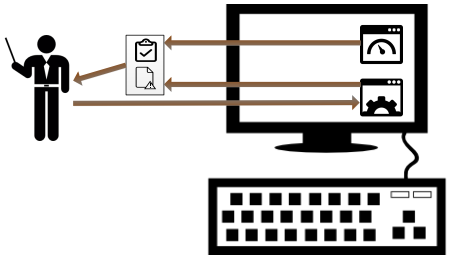

FIGURE 3.17 – Notation des rendus

Les figures 3.18 et 3.19 présentent le lien entre l'étudiant et les outils. Dans un premier temps, l'étudiant développe de manière classique l'application JavaCard. Dans un second temps, l'élèvel lance le terminal de test par étape de développement sur son application pour obtenir une correction et un guidage de son travail. Ce qui lui permet une approche de type développement incrémental tout en appliquant les notions de vérification, validation et évaluation de manière pratique.

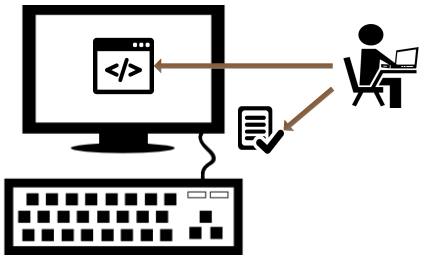

FIGURE 3.18 – Développement de l'application

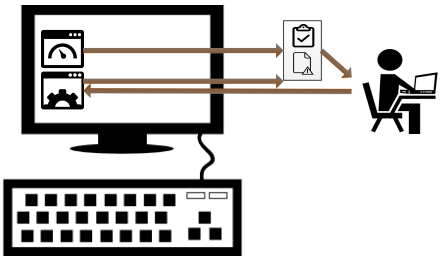

FIGURE 3.19 – Auto-évaluation incrémentale

Pour l'institution, notre maquette permet une formation professionnalisante impliquant les étudiants et améliorant le retour (corrections et notes) de l'enseignant.

Discussion

Cette méthodologie d'enseignement renforce une rigueur dans le développement JavaCard. De plus, nous pouvons la caractériser de méthode modulaire, simple et accessible. Son ouverture sur le monde de la recherche et son lien avec la pédagogie de la sécurité logicielle ne font que renforcer l'intérêt de cette méthodologie. Cependant, cette technique peut s'avérer longue à mettre en place, surtout la première année. Cependant, si l'enseignant a les compétences pour représenter le programme (ce qui est normalement le cas), il n'est pas difficile de prendre en main l'outil et de le configurer et la collection de propriétés peut être générée par l'algorithme génétique. Le lien entre développement et évaluation est vu de manière concrète et permet de faciliter et d'illustrer la méthode de notation des TPs auprès des élèves (la relation serait comparable à la relation entre le constructeur et le laboratoire qui doit noter l'application).

3.5 Conclusion

La génération de l'oracle est une problématique importante dans l'élaboration d'une méthodologie d'évaluation. Nous avons exposé les méthodes de génération connues. Ensuite, nous avons travaillé sur deux méthodes : l'une partant de la documentation de l'application carte et la seconde utilisant une méthode de type rétro-ingénierie. Ces méthodes paraissant limitantes, nous avons mis en place une méthode permettant de garder les avantages de la rétro-ingénierie et de la connaissance des spécifications. Elle permet donc la génération de propriétés pertinentes via l'utilisation d'algorithmes génétiques. L'utilisateur peut configurer la génération de propriétés afin de guider la découverte de propriétés pertinentes et doit veiller à avoir un nombre conséquent de logs est nécessaire. Coupler cette génération avec une méthode de Fuzzing est une piste intéressante pour améliorer cette génération et avoir ainsi plus de donneés en entrée de notre outil.

Concernant notre approche pragmatique, nous avons exposé deux cas d'usages de la méthodologie : durant la phase de développement et durant la phase de test. Aussi, nous avons pu illustrer l'apport de la méthodologie dans des domaines tels que le développement et son enseignement.

Conclusions et perspectives

Bilan

Après avoir rencontré la nécessité d'améliorer l'évaluation des cartes à puces et après avoir traité des définitions nous permettant d'appréhender le sujet, nous avons défini une méthodologie de type boîte noire pour l'évaluation des applications embarquées sur cartes à puces. Cette méthode, bien que générique et utilisable sur tout type d'applications sur carte à puce, est exposé sur des applications de paiement (EMV et PME). Une preuve de concept a été développée grâce au framework WSCT. Finalement, des usages concrets de la méthodologie sont explicités.

Cette approche pragmatique à destination des milieux industriels et académiques est utilisable et configurable simplement via la définition d'un langage simple. La pertinence et l'automatisation de la génération de l'oracle sont permises par l'utilisation d'algorithmes génétiques et de techniques de Fuzzing. Au final, cette thèse permet en plus de fournir les clefs sur l'évaluation des applications de cartes à puces, de donner un outil intéressant utilisable dans différents contextes.

Contributions

Avec cette thèse, nous contribuons à proposer une méthodologie d'évaluation logicielle d'une carte à puce. Nous contribuons de manière large à l'évaluation des applications de ce type en proposant une méthodologie complète allant de la génération de l'oracle à la détection d'anomalie. Cette méthodologie peut être utilisée durant la phase de test ou la phase d'utilisation de la carte. De plus, nous avons contribué à l'enseignement du développement JavaCard et de l'évaluation par un usage de notre outil pour la pédagogie. Enfin, notre méthodologie est conçue pour être utilisable simplement dans des situations concrètes.

Perspectives

Les perspectives de cette étude sont nombreuses. Il s'agit dans un premier temps d'améliorer la génération de l'oracle afin d'obtenir un oracle plus complet et pertinent. Nous avons une efficacité d'une collection de propriétés de longueur quatre maximum générée par seconde. Cependant, il faut améliorer la configuration et le guidage de la génération de manière automatique. Nous pourrions envisager de créer des collections de propriétés pré-configurées et génériques à appliquer selon le type de système cible. En effet, les applications étudiées dans notre étude sont basées sur ISO/IEC 7816. Une collection permettrait de vérifier la base ISO/EIC 7816, une seconde pour le déroulement de l'application EMV ainsi qu'une dernière pour les propriétés liées aux spécifications propriétaires. Cette vision modulaire de notre observateur permettrait d'utiliser plus simplement et plus largement notre méthodologie. Une piste est d'utiliser notre preuve de concept sur des résultats de test du monde industriel pour en évaluer l'apport.

Une maquette d'enseignement viable et complète du développement JavaCard permettant une formation professionnalisante est présentée dans la section précédente. Une perspective est de la mettre en place dans un contexte réel afin d'en évaluer l'efficacité sur le ressenti des étudiants et de l'enseignant. Le gain opérationnel est possible sur plusieurs années. En effet, la première année, un travail est nécessaire pour la mettre en place. Cependant, l'apport pour la formation ne fait pas de doute et cette maquette devrait être mise en place à l'ENSICAEN pour améliorer les enseignements actuels et donner plus de poids à la formation proposée.

Un moteur d'analyse comportementale portable et modulaire est une piste très intéressante. En effet, il est nécessaire, d'une part, de matérialiser notre approche afin de la rendre accessible (démonstration, utilisation en contexte réel dans le monde industriel, ...) et d'autre part, nous pourrions avoir des résultats intéressants en ayant un outil matériel permettant l'analyse à la volée d'une application en poussant l'étude dans la direction de la rétro-ingénierie. Cette étude pourrait nous permettre d'analyser le comportement d'une application inconnue mais aussi de vérifier le comportement d'une application durant son utilisation de la vie de tous les jours. Actuellement, nous débutons la faisabilité et l'efficacité de ce type d'outil via l'utilisation d'un raspberry permettant l'observation de la communication entre un terminal et une carte et d'un mobile Android permettant d'avoir un terminal d'analyse et de détection portable.

Publications de l'auteur

Conférences internationales avec comité de lecture et avec actes

1. **G**. Jolly, Vernois, S., & Rosenberger, C. An Observe-and-Detect methodology for the security and functional testing of smart card applications, In The International Conference on Information Systems Security and Privacy (ICISSP), Rome, Italy, (2016, February).

2. **G**. Jolly, Hemery, B., & Rosenberger, C. Generation of local and expected behaviors of a smart card application to detect software anomaly. In Availability, Reliability and Security (ARES), Toulouse, France, (2015, August).

3. **G**. Jolly, S. Vernois and J.-L. Lambert, "Improving Test Conformance of Smart Cards versus EMV-Specification by using on the Fly Temporal Property Verification", Second International Conference on Security in Computer Networks and Distributed Systems (SNDS), Thiruvananthapuram , India , (2014, March).

Conférences nationales avec comité de lecture et sans actes

1. G. Jolly, S. Vernois and C. Rosenberger, "Méthodologie d'enseignement du développement et de l'évaluation d'application carte par un outil venant de la recherche académique",Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI), Cité de l'Espace (Toulouse) , France , (Mai 2016).

2. **G**. Jolly, S. Vernois and J.-L. Lambert, "Améliorer le test de conformité des cartes à puces aux spécifications EMV en utilisant la vérification de propriétés

temporelles à la volée", Sécurité des Architectures Réseaux et des Systèmes d'Information (SARSSI), Saint-Germain-Au-Mont-d'Or (Lyon) , France , (Mai 2014).

Autres communications

1. **G**. Jolly, Evaluation logicielle des puces avec et sans contact, Journée de l'Ecole Doctorale (JED) SIMEM, Caen, France, Octobre 2015.

2. G. Jolly, S. Vernois and J.-L. Lambert, "Software Evaluation of smart cards : Detection of abnormal behavior of a smart card application",COINS Ph.D. student seminar in Tromsø, Norvège, Octobre 2014.

3. **G**. Jolly, Evaluation logicielle des puces avec et sans contact, Journée de l'Ecole Doctorale (JED) SIMEM, Caen, France, Juin 2014.

4. Participation à l'événement "ma thèse en 180 secondes" organisé par Normandie Université, Mai 2014.

5. Interventions de médiation scientifique sur la sécurité de la carte à puce et la recherche avec relais De Sciences, au collège de Giberville en 2013 et au lycée Chartier à Bayeux en 2014.

6. Interventions de médiation scientifique lors de la fête de la science à Caen durant la période 2012 - 2016 sur la sécurité informatique, les transactions sécurisés, la programmation informatique et la recherche.

Bibliographie

[1] Roland Moreno. Procédé et dispositif de commande électronique. *French patent FR2266222*, 1974. [cité p. 1]

[2] EMVCo. http://www.emvco.com/, 2014. [cité p. 2]

[3] SVP. http ://www.svp.com/article/le-marche-mondial-des-cartes-a-puce-de-belles-perspectives-de-croissance-100007857, 2016. [cité p. 3]

[4] TES. *TES les transaction électroniques un enjeu clé*. 2006. [cité p. 6, 107]

[5] Bresson. Mémorandum sur les transactions electroniques sécurisées et la monétique. 2004. [cité p. 6]

[6] Mostafa Hashem Sherif and Ahmed Serhrouchni. *La monnaie électronique : systèmes de paiement sécurisé*. Eyrolles, 1999. [cité p. 6]

[7] Hallépée Didier. *L'univers de la monetique*. 2010. [cité p. 7]

[8] Comprendre les paiements. http ://www.comprendrelespaiements.com/modele-a-4-coins-en-monetique/, 2014. [cité p. 7, 107]

[9] http://www.emvco.com/approvals.aspx, 2015. [cité p. 8]

[10] EMVco. https ://www.emvco.com/approvals.aspx ?id=104, 2016. [cité p. 8]

[11] ANSSI. http://www.ssi.gouv.fr/administration/produits-certifies/cc/, 2016. [cité p. 8]

[12] ISO, 2010. [cité p. 9]

[13] Common Criteria. http ://www.commoncriteriaportal.org/, 2016. [cité p. 9]

[14] SOGIS. http://www.sogis.org/, 2010. [cité p. 9]

[15] Cartes America. Are embedded secure elements more secure than traditional smart cards ? http://www.cartes-america.com/files/are_embedded_secure_elements_more_secure_than_traditional_smart_cards___tilburg_witteman.pdf, 2014. [cité p. 10, 107]

[16] Oracle. http://www.oracle.com/technetwork/java/javacard/-overview/index.html, 2010. [cité p. 10]

[17] Global Platform. https ://www.globalplatform.org/, 2016. [cité p. 10, 17]

[18] Iso/iec 7816. [cité p. 11]

[19] Iso/iec 14443. [cité p. 11]

[20] C. Enrique Ortiz. An introduction to java card technology, 2003. [cité p. 13, 107]

[21] PC/SC WorkGroup. http ://www.pcscworkgroup.com/, 1996. [cité p. 13]

[22] EMVCo. *A guide to EMV*. EMVCo, 2011. [cité p. 14, 107]

[23] *EMV Book 1*. 2008. [cité p. 14]

[24] *EMV Book 2*. 2008. [cité p. 14]

[25] *EMV Book 3*. 2008. [cité p. 14]

[26] *EMV Book 4*. 2008. [cité p. 14]

[27] MasterCard International. *M/Chip 4 Card Application Specifications for Credit and Debit*. MasterCard International, 2002. [cité p. 16]

[28] Java Card Forum. https ://javacardforum.com/. [cité p. 17]

[29] Damien Sauveron. La technologie java card, 2002. [cité p. 17]

[30] Samia Bouzefrane. La technologie java card, 2008. [cité p. 17]

[31] C. Enrique Ortiz. An introduction to java card technology. http://www.oracle.com/technetwork/java/javacard/javacard1-139251.html, 2003. [cité p. 18, 107]

[32] Steven J Murdoch and Ross Anderson. Security protocols and evidence : Where many payment systems fail. *Proceedings of Financial Cryptography and Data Security*, pages 3–7, 2014. [cité p. 19, 31]

[33] Joint Interpretation Library. Application of attack potential to smartcards, 2013. [cité p. 19]

[34] Marc Joye and Michael Tunstall. *Fault Analysis in Cryptography*, volume 7. Springer, 2012. [cité p. 19]

[35] Andrey Sidorenko, Joachim van den Berg, Remko Foekema, Michiel Grashuis, Jaap de Vos, and BV Brightsight. Bellcore attack in practice. *IACR Cryptology ePrint Archive*, 2012 :553, 2012. [cité p. 19]

[36] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12. Springer, 2002. [cité p. 19]

[37] Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. In *Smart Card Research and Advanced Applications VI*, pages 159–176. Springer, 2004. [cité p. 19]

[38] Agnès Cristèle Noubissi, A Séré, Julien Iguchi-Cartigny, Jean-Louis Lanet, Guillaume Bouffard, and Julien Boutet. Cartesa puce : Attaques et contremesures. *MajecSTIC*, 16 :1112, 2009. [cité p. 19]

[39] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis : Concrete results. In *Cryptographic Hardware and Embedded SystemsCHES 2001*, pages 251–261. Springer, 2001. [cité p. 20]

[40] Oliver Kömmerling and Markus G Kuhn. Design principles for tamper-resistant smartcard processors. *Smartcard*, 99 :9–20, 1999. [cité p. 20]

[41] Abdelaziz Elaabid. *Attaques par canaux cachés : expérimentations avancées sur les attaques template*. PhD thesis, Université Paris VIII Vincennes-Saint Denis ; Ecole nationale supérieure des telecommunications-ENST, 2011. [cité p. 20]

[42] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in CryptologyCRYPTO99*, pages 388–397. Springer, 1999. [cité p. 20]

[43] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema) : Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001. [cité p. 20]

[44] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in CryptologyCRYPTO96*, pages 104–113. Springer, 1996. [cité p. 21]

[45] Mike Bond, Omar Choudary, Steven J Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and skim : cloning emv cards with the pre-play attack. *arXiv preprint arXiv :1209.2531*, 2012. [cité p. 21]

[46] Anti Skimming Eye. What is atm card skimming ? [cité p. 21, 107]

[47] Lishoy Francis, Gerhard Hancke, and Keith Mayes. A practical generic relay attack on contactless transactions by using nfc mobile phones. *International Journal of RFID Security and Cryptography*, 2(1) :92–106, 2013. [cité p. 21]

[48] S.J. Murdoch, S. Drimer, R. Anderson, and M. Bond. Chip and pin is broken. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 433–446. IEEE, 2010. [cité p. 21]

[49] Ziv Kfir and Avishai Wool. Picking virtual pockets using relay attacks on contactless smartcard. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 47–58. IEEE, 2005. [cité p. 22]

[50] Wolfgang Rankl. *Smart Card Applications : Design models for using and programming smart cards*. 2007. [cité p. 24, 25, 30, 40]

[51] International Organization for Standardization. *ISO 8402 : 1994 : Quality Management and Quality Assurance-Vocabulary*. International Organization for Standardization, 1994. [cité p. 24, 25]

[52] Jane Radatz, Anne Geraci, and Freny Katki. Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990 :121990, 1990. [cité p. 25]

[53] James A Wentworth, Rodger Knaus, and Hamid Aougab. Verification, validation and evaluation of expert systems. *World Wide Web electronic publication : http ://www. tfhrc. gov/advanc/vve/cover. htm*, 1995. [cité p. 25]

[54] Natalia Juristo. Chapter 3 verification and validation : Current and best practice. [cité p. 25]

[55] James A Wentworth, Rodger Knaus, and Hamid Aougab. Verification, validation, and evaluation of expert systems. volume 1 : An fhwa handbook. Technical report, 1997. [cité p. 26]

[56] Dolores R Wallace, Laura M Ippolito, and Barbara B Cuthill. *Reference information for the software verification and validation process*, volume 500. DIANE Publishing, 1996. [cité p. 27]

[57] Food, Drug Administration, et al. General principles of software validation ; final guidance for industry and fda staff. *Rockville (MD) : FDA*, 2002. [cité p. 27]

[58] Wolfgang Rankl and Wolfgang Effing. *Smart card handbook*. John Wiley & Sons, 2010. [cité p. 28, 34]

[59] Joint Interpretation Libray. Smartcard evaluation guidance, 2010. [cité p. 28]

[60] Explorathèque. Les acteurs du secteur de la carte à puce, 2012. [cité p. 29, 107]

[61] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. An overview of formal methods tools and techniques. In *Rigorous Software Development*, pages 15–44. Springer, 2011. [cité p. 32]

[62] Samia Bouzefrane. La carte à puce. 2013. [cité p. 32]

[63] Ludovic Casset. *Construction correcte de logiciels pour carte à puce*. PhD thesis, Citeseer, 2002. [cité p. 33]

[64] Mark Utting and Bruno Legeard. *Practical model-based testing : a tools approach*. Morgan Kaufmann, 2010. [cité p. 33]

[65] Jean-Louis Lanet. Are smart cards the ideal domain for applying formal methods. In *ZB 2000 : Formal Specification and Development in Z and B*, pages 363–373. Springer, 2000. [cité p. 33, 41]

[66] Denis Sabatier and Pierre Lartigue. The use of the b formal method for the design and the validation of the transaction mechanism for smart card applications. In *FM99 Formal Methods*, pages 348–368. Springer, 1999. [cité p. 33, 41]

[67] Lilian Burdy, Antoine Requet, and Jean-Louis Lanet. Java applet correctness : A developer-oriented approach. In *FME 2003 : Formal Methods*, pages 422–439. Springer, 2003. [cité p. 33]

[68] Bart Jacobs, Claude Marché, and Nicole Rauch. Formal verification of a commercial smart card applet with multiple tools. In *Algebraic Methodology And Software Technology*, pages 241–257. Springer, 2004. [cité p. 33, 41]

[69] C-B Breunesse, Nestor Catano, Marieke Huisman, and Bart Jacobs. Formal methods for smart cards : an experience report. *Science of Computer Programming*, 55(1) :53–80, 2005. [cité p. 33]

[70] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager, et al. The key tool. *Software & Systems Modeling*, 4(1) :32–54, 2005. [cité p. 33]

[71] Renaud De Landtsheer, Christophe Ponsard, and Nicolas Devos. A constraint-solving approach for achieving minimal-reset transition coverage of smartcard behaviour. *CTIT*, 2 :221, 2014. [cité p. 33]

[72] Julien Iguchi-Cartigny. *Contributions à la sécurité des Java Card*. PhD thesis, Université de Limoges, 2014. [cité p. 33]

[73] Sam Owre, Sreeranga Rajan, John M Rushby, Natarajan Shankar, and Mandayam Srivas. Pvs : Combining specification, proof checking, and model checking. In *Computer Aided Verification*, pages 411–414. Springer, 1996. [cité p. 33]

[74] Dino Distefano and Matthew J Parkinson J. jstar : Towards practical verification for java. *ACM Sigplan Notices*, 43(10) :213–226, 2008. [cité p. 33]

[75] Pieter Philippaerts, Jan Tobias Mühlberg, Willem Penninckx, Jan Smans, Bart Jacobs, and Frank Piessens. Software verification with verifast : Industrial case studies. *Science of Computer Programming*, 82 :77–97, 2014. [cité p. 33]

[76] Software verification with verifast : Industrial case studies. *Science of Computer Programming*, 2013. [cité p. 33]

[77] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8) :1090–1123, 1996. [cité p. 33]

[78] Dominik Haneberg, Holger Grandy, Wolfgang Reif, and Gerhard Schellhorn. Verifying smart card applications : an asm approach. In *Integrated Formal Methods*, pages 313–332. Springer, 2007. [cité p. 33, 41]

[79] Joachim Posegga and Harald Vogt. Byte code verification for java smart cards based on model checking. In *Computer SecurityESORICS 98*, pages 175–190. Springer, 1998. [cité p. 33, 41]

[80] Elizabeta Fourneret, Martin Ochoa, Fabrice Bouquet, Julien Botella, Jan Jurjens, and Parvaneh Yousefi. Model-based security verification and testing for smart-cards. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 272–279. IEEE, 2011. [cité p. 33]

[81] Kristin Y Rozier. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2) :163–203, 2011. [cité p. 33]

[82] J. de Ruiter and E. Poll. Formal analysis of the emv protocol suite. *Theory of Security and Applications*, pages 113–129, 2012. [cité p. 33]

[83] Joeri De Ruiter and Erik Poll. Formal analysis of the emv protocol suite. In *Theory of Security and Applications*, pages 113–129. Springer, 2012. [cité p. 33]

[84] F. Aarts, E. Poll, and J. de Ruiter. Formal models of banking cards for free. *Unpublished*, 2012. [cité p. 33, 35, 69, 72]

[85] Noura Ouerdi, Mostafa Azizi, M'Hammed Ziane, Abdelmalek Azizi, and Aymerick Savary. Security vulnerabilities tests generation from sysml and event-b models for emv cards. *International Journal of Security and Its Applications*, 8(1) :373–388, 2013. [cité p. 33]

[86] Testing vs. Model Checking. http ://babelfish.arc.nasa.gov/trac/jpf/wiki/. [cité p. 34, 107]

[87] Thierry Jéron and Pierre Morel. Test generation derived from model-checking. In *Computer Aided Verification*, pages 108–122. Springer, 1999. [cité p. 34]

[88] Jean-Claude Fernandez, Claude Jard, Thierry Jéron, and César Viho. Using on-the-fly verification techniques for the generation of test suites. In *Computer Aided Verification*, pages 348–359. Springer, 1996. [cité p. 34]

[89] Carlos Pacheco and Michael D Ernst. *Eclat : Automatic generation and classification of test inputs*. Springer, 2005. [cité p. 34, 69]

[90] Antonio Cicchetti. Supporting the automatic test case and oracle generation using system models. Master's thesis, 2009. [cité p. 34]

[91] Hugues Martin and Lydie du Bousquet. Automatic test generation for java card applets. In *Java on Smart Cards : Programming and Security*, pages 121–136. Springer, 2001. [cité p. 34]

[92] Jan Philipps, Alexander Pretschner, Oscar Slotosch, Ernst Aiglstorfer, Stefan Kriebel, and Kai Scholl. Model-based test case generation for smart cards. *Electronic Notes in Theoretical Computer Science*, 80 :170–184, 2003. [cité p. 34]

[93] Guido Wimmel and Jan Jürjens. Specification-based test generation for security-critical systems using mutations. In *Formal Methods and Software Engineering*, pages 471–482. Springer, 2002. [cité p. 34]

[94] Michael Eddington. Peach fuzzing platform. *Peach Fuzzer*, 2011. [cité p. 34]

[95] Éric Lacombe. Fuddly : un framework de fuzzing et de manipulation de données. [cité p. 34]

[96] J. Lancia. Un framework de fuzzing pour cartesa puce : application aux protocoles emv. In *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, 2011. [cité p. 34, 74]

[97] Vincent Alimi, Sylvain Vernois, and Christophe Rosenberger. Analysis of embedded applications by evolutionary fuzzing. In *High Performance Computing & Simulation (HPCS), 2014 International Conference on*, pages 551–557. IEEE, 2014. [cité p. 34, 68, 74]

[98] Yoonsik Cheon, Myoung Yee Kim, and Ashaveena Perumandla. A complete automation of unit testing for java programs. 2005. [cité p. 34]

[99] http ://users.polytech.unice.fr/ rueher/cours/test/. [cité p. 35, 107]

[100] Arjen van Weelden, Martijn Oostdijk, Lars Frantzen, Pieter Koopman, and Jan Tretmans. On-the-fly formal testing of a smart card applet. In *Security and Privacy in the Age of Ubiquitous Computing*, pages 565–576. Springer, 2005. [cité p. 35]

[101] Johannes Ryser and Martin Glinz. A practical approach to validating and testing software systems using scenarios. In *QWE99, 3 rd International Software Quality Week Europe*. Citeseer, 1999. [cité p. 35]

[102] Wei-Tek Tsai, Akihiro Saimi, Lian Yu, and Raymond Paul. Scenario-based object-oriented testing framework. In *Quality Software, 2003. Proceedings. Third International Conference on*, pages 410–417. IEEE, 2003. [cité p. 35]

[103] Pierre Bontron and Marie-Laure Potet. Stratégie de couverture de test à un haut niveau d'abstraction. *TSI-Technique et Science Informatiques*, 23(7) :905–928, 2004. [cité p. 35, 46]

[104] Frédéric Dadeau and Régis Tissot. jsynopsys–a scenario-based testing tool based on the symbolic animation of b machines. *Electronic Notes in Theoretical Computer Science*, 253(2) :117–132, 2009. [cité p. 35]

[105] Elliot J Chikofsky, James H Cross, et al. Reverse engineering and design recovery : A taxonomy. *Software, IEEE*, 7(1) :13–17, 1990. [cité p. 35]

[106] Dennis Vermoen, Marc Witteman, and Georgi N Gaydadjiev. Reverse engineering java card applets using power analysis. In *Information Security Theory and Practices.*

Smart Cards, Mobile and Ubiquitous Computing Systems, pages 138–149. Springer, 2007. [cité p. 35]

[107] Omar Choudary. The smart card detective : a hand-held emv interceptor. *University of Cambridge, Computer Laboratory, Darwin College, Cambridge, MPhil Thesis*, 2010. [cité p. 35, 37]

[108] Gerhard de Koning Gans and Joeri de Ruiter. The smartlogic tool : Analysing and testing smart card protocols. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 864–871. IEEE, 2012. [cité p. 35, 37]

[109] Andriana Gkaniatsou, Fiona McNeill, Alan Bundy, Graham Steel, Riccardo Focardi, and Claudio Bozzato. Getting to know your card : Reverse-engineering the smart-card application protocol data unit. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 441–450. ACM, 2015. [cité p. 35, 72]

[110] http ://www.program-transformation.org/transform/decompilationandreverseengineering. [cité p. 36, 107]

[111] http ://www.fisglobal.com/. [cité p. 36]

[112] Clear2Pay. Open test platform, https ://www.clear2pay.com/testing/about-your-testing/open-test-system. [cité p. 36]

[113] Visucard, galitt, http ://www.galitt.fr/visucard_790.html. [cité p. 36]

[114] Scriptis picc, keolabs, http ://www.keolabs.com/emv-solutions.html. [cité p. 36]

[115] Emvericard, fime, http ://www.fime.com/emvericard.html. [cité p. 36]

[116] http ://www.elitt.com/fr/news/verification-de-la-personnalisation-des-cartes-a49.html. [cité p. 36]

[117] https ://www.ul-ts.com/standards/jcb/test-tools/ul-emv-personalization-validation-tool/c-38/c-1525/p-118. [cité p. 36]

[118] Daysmart, daybreaksoftware, http ://www.daybreaksoftware.com/specs/daysmart.pdf. [cité p. 36]

[119] Spectro 2, comprion, http ://www.comprion.com/en/products/card_testing/spectro_2/overview. [cité p. 36]

[120] Gemalto, gem pcsc, http ://support.gemalto.com/index.php ?id=download_tools. [cité p. 37]

[121] https ://www.riscure.com/security-tools/jcworkbench/. [cité p. 37]

[122] http ://www.barnestest.com/. [cité p. 37]

[123] http ://www.microprossmicropross.com/laboratory-testing-smart-card-contact-iso-7816-swp-smartcard-tester-mp300-tc3-20-p. [cité p. 37]

[124] Scard soft, cardtoolsetpro , http ://www.scardsoft.com/index.php. [cité p. 37]

[125] http ://www.openscdp.org/. [cité p. 37, 38]

[126] https ://code.google.com/p/javaemvreader/. [cité p. 37]

[127] Cardpeek, https ://code.google.com/p/cardpeek/. [cité p. 37]

[128] Maurizio Talamo, Maulahikmah Galinium, Christian H Schunck, Franco Arcieri, et al. Secure messaging implementation in opensc. *Journal of Information Security*, 3(04) :251, 2012. [cité p. 37]

[129] https ://github.com/nicbedford/cardbrowser. [cité p. 37]

[130] S. Vernois and V. Alimi. Winscard tools : a software for the development and security analysis of transactions with smartcards. *Norsk informasjonssikkerhetskonferanse (NISK)*, 2010. [cité p. 37, 38, 41, 55]

[131] Smarcard framework, http ://www.codeproject.com/articles/17013/smart-card-framework-for-net. [cité p. 37]

[132] Pcsc sharp, https ://code.google.com/p/pcsc-sharp/. [cité p. 38]

[133] Anis Bkakria, Guillaume Bouffard, Julien Iguchi-Cartigny, and Jean-Louis Lanet. Opal : an open-source global platform java library which includes the remote application management over http. In *e-Smart 2011*, 2011. [cité p. 38]

[134] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing : A survey. 2015. [cité p. 38, 68]

[135] http ://www.xml.com/axml/testaxml.htm. [cité p. 42]

[136] H.D. Foster, A.C. Krolnik, and D.J. Lacey. *Assertion-based design*. Springer, 2004. [cité p. 46, 47, 48, 108]

[137] C Michael Chang. Property specification : the key to an assertion-based verification platform. 2003. [cité p. 46]

[138] Shmuel Katz. Aspect categories and classes of temporal properties. In *Transactions on aspect-oriented software development I*, pages 106–134. Springer, 2006. [cité p. 46]

[139] J. Groslambert. *Vérification de propriétés temporelles par génération dannotations*. PhD thesis, 2007. [cité p. 46]

[140] L. Ferro. *Vérification de propriétés logico-temporelles de spécifications SystemC TLM*. PhD thesis, Université de Grenoble, 2011. [cité p. 46]

[141] R. Grigore, R.L. Petersen, and D. Distefano. Topl : A language for specifying safety temporal properties of object-oriented programs. FOOL, 2011. [cité p. 46]

[142] Erik Poll, Joeri de Ruiter, and Aleksy Schubert. Protocol state machines and session languages : specification, implementation, and security flaws. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 125–133. IEEE, 2015. [cité p. 46]

[143] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 408–411. IEEE, 2001. [cité p. 46]

[144] Jan Jürjens and Guido Wimmel. Security modelling for electronic commerce : The common electronic purse specifications. In *Towards the E-Society*, pages 489–505. Springer, 2002. [cité p. 46]

[145] Germain Jolly, Sylvain Vernois, and Jean-Luc Lambert. Improving test conformance of smart cards versus emv-specification by using on the fly temporal property verification. In *Recent Trends in Computer Networks and Distributed Systems Security*, pages 192–201. Springer, 2014. [cité p. 48]

[146] B. Vibert, V. Alimi, S. Vernois, et al. Analyse de la sécurité de transactions à puce avec le framework winscard tools. *SAR-SSI 2012*, 2012. [cité p. 56]

[147] Dennis K Peters and David Lorge Parnas. Using test oracles generated from program documentation. *Software Engineering, IEEE Transactions on*, 24(3) :161–173, 1998. [cité p. 68]

[148] Pablo Loyola, Matt Staats, In-Young Ko, and Gregg Rothermel. Dodona : automated oracle data set selection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 193–203. ACM, 2014. [cité p. 69]

[149] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. Does automated white-box test generation really help software testers ? In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 291–301. ACM, 2013. [cité p. 69]

[150] Lawrence Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991. [cité p. 74]

[151] Hanno Langweg. Malware attacks on electronic signatures revisited. In *Sicherheit*, pages 244–255. Citeseer, 2006. [cité p. 81]

[152] http ://www.fernandes.org/apduview/. [cité p. 81]

[153] http ://www.fi.muni.cz/ xsvenda/apduinspect.html. [cité p. 81]

Liste des abréviations

AFL	Application File Locator
AID	Application IDentifier
$APDU$	Application Protocol Data Unit
API	Application Programming Interface
$ANSSI$	Agence Nationale de la Sécurité des Systèmes d'Information
CDA	Combined DDA/Generate Application Cryptogram
$CESTI$	Centre d'Evaluation de la Sécurité des Technologies de l'Information
$COFRAC$	Comité Français d'ACréditation
CVM	Cardholder Verification Method
DDA	Dynamic Data Authentication
DDL	Dynamic Link Library
EAL	Evaluation Assurance Level
EMV	Europay Mastercard Visa
$PDOL$	Processing Options Data Object list
PIN	Personal Identifier Number
PME	Porte Monnaie Electronique
PSE	Payment System Environnement
SDA	Static Data Authentication
SE	Secure Element
SFI	Short File Identifier
SW	Status Word
TES	Transaction Electronique Sécurisé
TVR	Terminal Veirification Results
$UICC$	Universal Integrated CIrcuit Card

Table des figures

Liste des tableaux