MajecSTIC 2009
Avignon, France, du 16 au 18 novembre 2009

Cartes à puce : Attaques et contremesures.

Agnès Cristèle Noubissi, Ahmadou Al-Khary Séré, Julien Iguchi-Cartigny, Jean-Louis Lanet, Guillaume Bouffard, Julien Boutet
Université de Limoges, Laboratoire XLIM, Equipe SSD, 83 rue d'Isle, 87000 Limoges - France.
{agnes.noubissi, ahmadou.sere, julien.cartigny, jean-louis.lanet}@xlim.fr
{guillaume.bouffard02, julien.boutet}@etu.unilim.fr

Résumé
Nous présentons dans cet article nos travaux de recherche traitant des attaques en fautes et des attaques logiques sur les cartes à puce, en particulier sur la Java Card. Nous introduisons en présentant la Java Card et ses mécanismes sécuritaires. Ensuite nous présentons les types d'attaques réalisées sur les cartes à puce et, nous présentons quelques contremesures de ces attaques et en particulier celles sur les attaques en fautes. Et nous terminons par la présentation de nos travaux sur l'outil de manipulation d'un format de fichier de la Java Card et des propositions d'autres contremesures pour les attaques en faute.

Abstract
We present in this article our research dealing with faults attacks and logical attacks on smart cards, especially on Java Card. We introduce by presenting Java Card and its security mechanisms, then we present the types of attacks carried out on smart cards and we present some countermeasures for these attacks and in particular those faults attacks. And we finish with the presentation of our work on the tool of manipulation of a file format of Java Card and other proposals for countermeasures on faults attacks.

Mots-clés : Carte à puce, Java Card, attaques en faute.

Keywords : Smart card, Java Card, fault attacks.

1.Introduction

Une carte à puce est une carte plastique de dimension réduite possédant un microprocesseur permettant de réaliser un certain nombre de tâches. Elle possède généralement une ROM (*Read Only Memory*), une RAM et une EEPROM de quelques kilo-octets de capacité. Certaines cartes à puce sont dotées de coprocesseur cryptographique fournissant un accélérateur de calcul pour les fonctions cryptographiques. Elle communique avec l'extérieur grâce au lecteur de carte ou via une antenne selon qu'elle soit sans contact, avec contact ou hybride.
La carte à puce s'est rapidement répandue dans le domaine des télécommunications, bancaire, santé, transport, etc... Aujourd'hui, elle est vue comme un support sécurisé capable de stocker et de manipuler des données sensibles de façon fiable, raison pour laquelle elle est de plus en plus utilisée pour sécuriser des systèmes d'information de taille plus importante. Cependant, une question générale reste toujours en suspend: La carte à puce est-elle vraiment sécurisée au point de servir de base sécuritaire pour d'autres systèmes? En effet, Nous verrons dans cet article que la sécurité de la carte est souvent mise à rude épreuve par les attaquants qui utilisent tous les procédés matériels ou logiciels possibles.
Nous présenterons dans cet article certaines attaques réalisées sur les cartes à puce en général. Après une brève présentation de certaines contremesures existantes, nous présenterons nos travaux de recherche soit: (1) Un outil conçu pour réaliser des attaques logicielles sur la Java Card en manipulant les fichiers CAP, (2) Les contremesures élaborées contre certaines attaques en faute sur la Java Card.

2. Java Card 2.2.2

2.1. Présentation

La Java Card est une carte à puce possédant une plateforme basée sur la technologie Java. Elle est dite « ouverte » car elle permet de charger, d'exécuter et de supprimer des applications après émission de la carte. Ces applications, appelées *applets* sont écrites dans un sous-ensemble du langage Java. Elles sont ensuite transformées en un format compact CAP (*Converted Applet*), puis sont chargées dans la carte pour être exécutées par la machine virtuelle Java Card (JCVM) [1] dont le rôle est d'interpréter le *byte code* associé. Pour des raisons dues aux ressources limitées de la carte, la JCVM est subdivisée en deux parties:

- partie *off card :* le vérifieur de *bytecode* (BCV ou *ByteCode Verifier*) et le convertisseur (applet vers fichier CAP).
- partie *on card :* l'interpréteur, l'API et l'environnement d'exécution des applications de la Java Card (JCRE : *Java Card Runtime Environment*). Dès que l'applet est chargée sur la carte et que l'édition des liens effectuée, elle peut être exécutée par l'interpréteur grâce aux fonctions contenues dans les API.

2.2. Mécanismes sécuritaires dans la Java Card

La plate-forme Java Card est un environnement multi-applicatif dans lequel les données sensibles d'une applet doivent être protégées contre l'accès malveillant pouvant être réalisé par d'autres applets [12]. Pour se prémunir de ces attaques, plusieurs mesures de sécurité sont utilisées dans le processus de chargement et d'exécution des *applets*. Elles peuvent être classées comme suit :
(1) Avant chargement : A l'extérieur de la carte, un vérifieur de byte code s'assure de la sûreté de types et des données manipulées par l'applet. C'est un composant crucial de la sécurité pour la plateforme Java Card.
(2) Lors du chargement : La couche logicielle Global Platform permet de s'authentifier auprès de la carte afin de pouvoir charger les applets. De plus, contrairement à l'approche Java classique, il n'existe qu'un seul chargeur de classe et il est impossible de redéfinir son comportement.
(3) Lors de l'exécution : Afin d'assurer l'innocuité lors de l'exécution des applications, l'isolement des applets est imposé grâce au *firewall* de Java Card assurant l'isolement des contextes d'application. La communication entre applets étant effectuée grâce au protocole de partage SIO *(Sharing Interface Object)*.

A côté de ces mécanismes logicielles de sécurité, s'ajoutent les mécanismes physiques de protection. Néanmoins, nous verrons que la carte à puce en général est très prisée par les attaquants.

3.Cartes à puce et sécurité : les attaques

3.1.Cartes à puce

Le but d'un attaquant est de pouvoir accéder aux informations et aux secrets contenus dans la carte (code PIN, clé(s) secrète(s) cryptographique(s), etc...) ou tout simplement de nuire aux applications embarquées afin de tester le niveau sécuritaire de la carte. Les attaques sont de deux types: physiques et logicielles. Dans la suite de l'article, nous allons principalement aborder les attaques physiques, car nos travaux de recherches sur les contremesures gravitent autour de cet axe.

3.2.Attaques matérielles ou physiques

Une attaque physique est une attaque menée sur la partie électronique de la carte. En effet, les algorithmes cryptographiques sont souvent implémentés sur des modules physiques que l'attaquant peut observer, voir perturber. Ces attaques peuvent être de deux types : invasives et non invasives.

3.2.1.Attaques invasives

Elles permettent de récupérer un ensemble d'information de la carte en se basant sur une cartographie des circuits [10]. Ces attaques sont dites de *reverse engineering* car l'attaquant tente de déduire les algorithmes utilisés, leurs implémentations, les systèmes de sécurité mis en place et les informations contenus dans la puce à partir de l'analyse ou la modification des circuits intégrés dans la carte. Pour pouvoir arriver à ce résultat, l'attaquant cherche à isoler les circuits de manière physique ou chimique. Le plus souvent, la puce n'est plus réutilisable après ces attaques, d'où l'aspect invasif. On peut citer comme exemple:

- La modification de circuit à l'aide du FIB (*Focused Ion-Beam*) permettant d'ajouter ou de retirer des nouvelles pistes conductrices sur la puce.
- La rétro-conception des blocs fonctionnels du microprocesseur dans le but de déterminer toutes les informations secrètes de la carte.
- le *probing* physique, c'est-à-dire la pose de micro-sondes sur les bus de la puce dans le but de déterminer ou falsifier l'information qui y circule.

3.2.2.Attaques non invasives

Attaques par conditions anormales

L'attaquant fait fonctionner la carte avec des valeurs en dehors des normes acceptables de fonctionnement de la carte. Ces caractéristiques peuvent par exemple être la fréquence d'alimentation ou la tension aux bornes de la carte. Aujourd'hui, une majorité des cartes possèdent des détecteurs d'activités anormales permettant de désactiver la carte pour lutter contre ce type d'attaque.

Attaques par canaux cachés

Ce sont des attaques par observation du signal puis d'analyse statistique. Les paramètres d'observation pouvant être le temps d'exécution, la consommation du courant ou même l'émission électromagnétique. L'attaque par analyse du temps d'exécution [2, 3] est basée sur le temps d'exécution des instructions. L'attaquant essaie de déduire des informations sur le type d'opération, les opérandes pour avoir des informations sur un programme ou un algorithme donné.

L'attaque par analyse de courant [8] est basée sur le fait que la consommation électrique d'un module est fonction de l'instruction exécutée, des opérandes (adresses et valeurs) et l'état précédent du module (valeurs des cellules mémoires et bus). Un attaquant peut donc, en observant cette consommation déterminer les opérations effectuées ainsi que les données manipulées par ces opérations. Ces attaques sont divisées en deux catégories : **SPA** *(simple power attack)* et **DPA** (*Differential power attack*). Le but de la SPA étant d'obtenir l'information sur la clé sécrète à partir

d'une seule mesure de consommation, ce qui n'est pas le cas pour la DPA qui utilise des méthodes statistiques (la distance des moyennes, le coefficient de *Pearson*, le maximum de vraisemblance, etc...). Il existe aussi HODPA (*High Order DPA*), plus puissante que la DPA car utilisant des méthodes statistiques sur la corrélation de plusieurs paramètres d'entrée et les résultats des mesures.

L'attaque par analyse de l'émission électromagnétique [9, 10] est similaire à celle par analyse de courant avec pour but de déterminer l'information grâce à l'intensité du rayonnement électromagnétique observée à la surface de la puce lors de l'exécution d'une instruction d'un module.

Attaques par injection de fautes

Elles sont encore appelées attaques par perturbation [1, 4, 5, 6, 7]. En effet, l'attaquant tente d'injecter des modifications physiques dans l'environnement de la carte (lumineuses, impulsions électriques, magnétiques, etc...) pour introduire des modifications dans le contenu des mémoires de la carte. Le but étant d'introduire des fautes lors de l'exécution d'un programme afin de provoquer des sorties erronées exploitables, d'éviter un test, d'appeler une autre sous fonction, de sauter l'appel de vérification d'un code PIN, etc...
Cependant l'attaquant doit pouvoir localiser les cellules mémoires à attaquer et synchroniser l'attaque pour qu'elle coïncide avec la fenêtre d'opportunité offerte par le programme. Pour cela, l'attaquant peut observer l'activité de la carte à l'aide d'attaques par canaux cachés.

3.3.Attaques logicielles

Avec l'apparition des cartes « ouvertes » permettant de charger des applications dans la carte après émission de celle-ci, les attaques logicielles sont de plus en plus répandues. Elles utilisent des failles pour contourner les protections mises en place. Généralement, il s'agit d'une mise à défaut des mécanismes d'isolation et d'intégrité du code et des données des applications embarquées.

3.4.Les contremesures

Ces contremesures peuvent être logicielles et/ou matérielles selon le type d'attaques. Pour les attaques physiques, elles consistent en général à brouiller non seulement les informations circulant sur la puce mais aussi les paramètres physiques pouvant être utilisés. Ceci passe par la génération d'activités aléatoires, la modification de la consommation de la puce, les mécanismes de chiffrement des bus et de protection de l'intégrité des données. Cependant, les attaques en fautes sont les plus résistantes.

Contremesures des attaques en fautes

À partir de l'injection de fautes, un attaquant peut réduire la sureté d'un algorithme cryptographique ou déduire des informations permettant de découvrir certains secrets enfouis. Les contre-mesures associées peuvent détecter des comportements anormaux de l'algorithme en vérifiant certains paramètres durant l'exécution des algorithmes à protéger. Elles peuvent aussi consister à modifier les algorithmes afin que les attaques en fautes ne dévoilent aucune information sensible. Mais en général, ces contre-mesures sont spécialisées, car elles ont une connaissance de la sémantique des données associées.
Akkar, Goubin et Ly [22] proposent une approche basée sur le graphe de flux de contrôle. En effet, un programmeur écrit son application en précisant dans son code les fonctions à protéger grâce à des annotations exploitables par un outil de *preprocessing*. Le mécanisme de protection consiste donc à vérifier durant l'exécution si l'historique de passage par certains drapeaux est valide vis-à-vis de *patterns* indiqués par le programmeur comme étant des historiques cohérents de la zone correspondante.
En [24], il est question d'un mécanisme permettant de vérifier l'intégrité des flux de contrôle que les auteurs de l'article ont appelé CFI (Control Flow Integrity). CFI se base donc sur le graphe de flux de contrôle pour s'assurer de l'exécution correcte du programme. Le principe est de vérifier lorsqu'il y a un transfert du flux de contrôle (un saut conditionnel par exemple) que ce transfert a bien lieu vers une destination valide.

Le mécanisme cité en [23] propose une méthode basée sur des blocs élémentaire et dont le but est de
protéger les applications embarquées sur la carte, des manipulations non autorisées. Pour cela, un partitionnement du code de l'application en plusieurs blocs élémentaires est effectué et à chaque bloc élémentaire est associée une valeur de contrôle préalablement calculée. Pendant ou avant l'exécution du bloc élémentaire, la valeur de contrôle est à nouveau calculée et comparée avec celle sauvegardée. Si les résultats ne sont pas identiques alors il y a une erreur. Dans ce brevet, la technique proposée afin de calculer les valeurs de contrôle est un MD5 ou un SHA-1. Cependant, la complexité de calcul d'un MD5 ou d'un SHA-1 est très importante en terme de ressource et de temps processeur pour la carte car en plus il faut effectuer le calcul autant de fois qu'il existe de blocs élémentaires dans le code.

4.Nos travaux de recherche

4.1.Outil d'attaque logique : la librairie de manipulation

Le vérificateur de byte code permet de s'assurer de la sûreté des types et des données manipulées par le fichier CAP. Cependant, il est possible de contourner le vérificateur et de faire ainsi passer des applets cohérentes structurellement parlant mais avec des opérations illicites et des erreurs cachées. Ce type d'attaque, appelée attaque logique, permet de mettre en échec une partie de la sécurité logicielle en utilisant les faiblesses du vérificateur embarqué dans la carte.
Notre outil développé en langage Java permet d'altérer les fichiers CAP à charger sur la carte de façon à faire exécuter des opérations non autorisées par la JCVM (renvoyer une référence sur des objets normalement non accessibles) ou encore à détourner le déroulement normal d'un programme (contourner par exemple une levée d'exception).
Cependant, modifier un fichier CAP est fastidieux. Si l'on modifie des éléments dans un composant du fichier CAP, il faut aussi mettre à jour la taille du composant et les informations du *directory component* (composant qui répertorie l'ensemble des composants, ainsi que leur taille). Si d'autres composants référencent des éléments du composant modifié, alors il faudra également mettre à jour ces références. On comprend donc que ces modifications peuvent très rapidement devenir délicates à réaliser. De plus, le fichier CAP modifié doit resté cohérent, afin d'être accepté par la carte. D'où la nécessité de cette librairie destinée à la manipulation des fichier CAP.

Fonctionnalités
Cette librairie appelée *CAPFileManipulator* est écrite en Java et permet de:
- Charger un fichier CAP et le parser;
- Afficher les données du fichier composant par composant et sous un format intelligible;
- Modifier les éléments d'un fichier CAP et vérifier sa validité en maintenant sa cohérence;
- Et de sauvegarder les fichiers CAP éventuellement altérés.

Architecture de la librairie

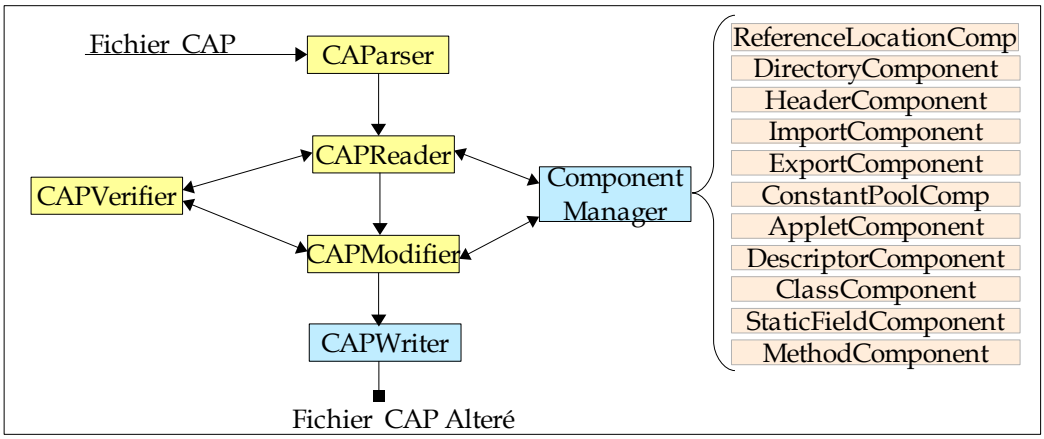

FIG. 1 –Composants de la librairie *CAPFileManipulator*

La figure 1 ci-dessus présente l'architecture de la librairie : ces composants et les interactions entre les composants. Chaque composant ayant un rôle bien précis:

1. CAPParser prend en entrée le fichier binaire et extrait les données nécessaires pour la lecture du fichier.
2. CAPReader effectue la lecture du fichier en décomposant celle ci en composant de fichier CAP grâce au *ComponentManager* permettant ainsi de le rendre compréhensible par un non expert et dans un langage naturel.
3. CAPModifier permet de prendre en compte les modifications effectuées sur le fichier CAP en vérifiant que le fichier reste cohérent et valide grâce au CAPVerifier.
4. CAPWriter s'occupe de l'écriture du fichier modifié.

Cette librairie permet donc de modifier un fichier CAP tout en garantissant que celui-ci reste exécutable sur la plateforme Java Card. L'avantage principal de cet outil est le gain de temps lors des tests d'attaques logiques basés sur la manipulation du fichier CAP en automatisant les vérifications destinées à maintenir la cohérence et la validité du fichier pouvant contenir des instructions illicites.

4.2. Méthodes de contremesure d'attaques en faute

Le modèle d'attaque retenu est celui de la modification d'un octet précis à une valeur aléatoire, ou au *reset* c'est-à-dire le passage de cette valeur à 0x00 ou 0xFF.

4.2.1. La méthode basée sur les blocs élémentaires

Elle est basée sur l'idée du brevet [23]. En effet, elle consiste en une subdivision du code en plusieurs blocs élémentaires et au calcul d'une valeur de contrôle associée à chaque bloc. Cependant, l'approche se base sur une fonction de calcul qui s'appuie sur l'opération logique XOR. Cette opération logique moins coûteuse que la fonction de calcul du MD5 ou d'un SHA-1 permet d'obtenir une complexité beaucoup plus réduite satisfaisant ainsi les contraintes processeur de la carte. L'opération XOR possède une meilleure distribution par rapport aux opérations AND et OR par conséquent elle a été la meilleure candidate pour les opérations de vérification et de calcul de signature dans cette approche. En *off card*, un outil permet de déterminer, pour une méthode donnée, les blocs élémentaires [15] qui la compose, puis pour chaque bloc élémentaire, une valeur de contrôle est calculée grâce à l'opération logique XOR. L'outil sauvegarde dans le fichier CAP les données relatives à chaque méthode. Le fichier est ensuite chargé dans la carte. En ***on card,*** lors de l'exécution, l'interpréteur de byte code recalcule les valeurs de contrôle et vérifie pour chaque bloc élémentaire du code de la méthode considérée leur cohérence avec la valeur sauvegardée. Cette méthode permet de détecter des modifications de code tout en réduisant la complexité du mécanisme [23]. L'augmentation du fichier CAP reste donc dans des limites acceptables pour la carte: de l'ordre d'environ 3% pour la protection de tout le code.

4.2.2. La méthode du renommage de constante

Elle consiste à coder 0x00, qui est l'opération neutre (NOP) par une autre valeur. Ainsi, si lors de l'interprétation du code l'opération NOP apparaît avec son codage de base, c'est-à-dire le 0x00, c'est qu'il y a eu modification d'une instruction vers la valeur 0x00. Afin de montrer l'intérêt d'un tel codage, supposons qu'une instruction *ISOException.throwIt(...)* (levée d'exception) correspondant à aux octets 8D 00 0D. Si l'attaquant remplace cette instruction du code par des NOP NOP NOP (00 00 00), il annule ainsi la levée d'exception. Supposons qu'un code pin erroné entraine la levée d'une exception dans le code, il est possible avec cette méthode et dans ce cas précis d'empêcher l'arrêt du programme. Cette modification peut aussi être appliquée à la valeur 0xFF. Grâce à ce changement, on va être en mesure de détecter les attaques de type *reset* vers 0x00 ou 0xFF (qui est une instruction réservée de Java).

4.2.3. La méthode du champ de bit

Elle permet de contrer les attaques sur le remplacement d'opcode. L'idée de cette contre-mesure vient du constat que le remplacement d'un opcode par un autre entraîne soit aucun changement, une augmentation ou une diminution du nombre d'opérande. Nous proposons une méthode visant à détecter les décalages provoqués par ce type de changement. Pour cela, notre approche est effectuée en deux parties *off card* et o*n card*. Soit l'exemple suivant où un opcode est substitué par le bit 1 et une opérande par le bit 0 :

Code	**Valeur de contrôle**
0: load_1	1
1: invokevirtual #34	1 0 0
4: astore_2	1
5: aload_1	1
6: invokevirtual #52	1
9: pop	1

FIG. 2 –Exemple de calcul du tableau de bit

Partie *off card*

Une application permet de déterminer le tableau de bit correspondant au byte code à vérifier, puis le sauvegarde dans le fichier *class* correspondant. Dans notre exemple, La valeur du tableau de bit obtenu est de : 1100111001. Dans un soucis d'optimisation de l'espace mémoire, le tableau de bit est compressé en groupe d'octets.

Partie *on card*

Une vérification du byte code est effectuée lors de l'exécution grâce au tableau de bit pour déterminer s'il n'y a pas eu de modification de byte code.

Cette méthode a cependant des limites, puisqu'elle ne permet pas la détection d'attaque de modification de byte code par un autre de même type voir de même nombre de paramètres. Des travaux sont en cours pour pouvoir détecter avec cette méthode ce type d'attaque.

5.Conclusion

Nous avons présenté dans cet article différents types d'attaques sur les cartes à puce en général, nos travaux de recherche sur l'outil de manipulation de fichier CAP et quelques contremesures réalisées sur les attaques en fautes dans le cas de la Java Card. Cet article reprend les grandes idées sur les types d'attaques, sur les contremesures des attaques en fautes et offre ainsi une vision à la communauté STIC sur une relation entre informatique et électronique. Nous avons constaté que l'attaquant dispose de beaucoup plus de temps et de puissance de calcul pour mener à bien son attaque. Et que malgré l'existence de nombreuses contremesures, celles-ci n'assurent pas toujours une sécurité absolue mais permet toutefois d'assurer que les moyens à mettre en oeuvre pour réaliser l'attaque seront suffisamment importants (temps, argent, et expertise).

Nos travaux futurs portent sur l'élaboration d'un simulateur d'attaques avec l'intégration de nos contremesures permettant de visualiser en temps réel les comportements du système (les valeurs des registres, de la pile d'exécution, etc …) et de déterminer clairement les performances de l'outil et de ces contremesures.

6. Bibliographie

1. Andrew Appel and Sudhakar Govindavajhala. *Using Memory Errors to Attack a Virtual Machine*. In Proceedings of IEEE Symposium on Security and Privacy, 2003.

2. Paul Kocher. *Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems*. International Cryptology Conference on Advances in Cryptology, Springer-Verlag, 1996.
3. James Alexande Muir. *Techniques of Side Channel Cryptanalysis*. Master's thesis, University of Waterloo, Ontario, Canada, 2001.
4. Claire Whelan, David Naccache, Hagai Bar-El, Hamid Choukri and Michael Tunstall. *The Sorcerer's Apprentice Guide to Fault Attacks*. Workshop on Fault Detection and Tolerance in Cryptography, Italy, 2004.

5. Ross Anderson and Sergei Skorobogatov. *Optical Fault Induction Attacks.* Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2002), USA, 2002.
6. Markus G. Kuhn and Oliver Kömmerling. *Design Principles for Tamper-Resistant Smartcard Processors*. Workshop on Smartcard Technology (Smartcard '99), Chicago, USA, May 1999.
7. C. Giraud and H. Thiebeauld. *A survey on fault attacks*. CARDIS 2004.
8. Benjamin Jun, Joshua Jaffe and Paul Kocher. *Differential Power Analysis*. Cryptology Conference on Advances in Cryptology, 1999.
9. David Samyde and Jean-Jacques Quisquater. *ElectroMagnetic Analysis (EMA) : Measures and Countermeasures for Smart Cards*. E-smart, 2001.
10. Christophe Mourtel, Francis Olivier and Karine Gandol. *ElectroMagnetic Analysis : Concrete Results*. CHES'2001.
11. Markus G. Kuhn and Oliver Kömmerling. *Design Principles for Tamper-Resistant Smartcard Processors*. SmartCard'99.
12. Jean-Louis Lanet and Girard Paul. *New security issues raised by open cards*. Information Security Technical Report 4(1), 4–5 (1999)
13. Jean-Louis Lanet. I*ntroduction à la carte à puce et problématique de la sécurité.* Crypto'puces, Porquerolles, Avril 2007.
14. Jean-Louis Lanet. *Sécurité des systèmes ouverts pour cartes à puces.* Workshop ISYPAR, février 2000.
15. Jean-Louis Lanet. *Support de cours : carte à puce.* Octobre 2006.
16. Z. Chen. *Java Card Technology for Smart Cards*. Addison, Wesley, 2000.
17. Sun Microsystem. *The Java Card 3.0 specification.* `http://java.sun.com/javacard/`, `March 2008`.
18. Marc Roper, Murray Wood, and Neil Walkinshaw. *The Java System Dependance Graph.* International Workshop on Source Code Analysis and Manipulation, 2003.
19. Milan Fort. *Smart card application development using Java Card Technology*. SeWeS 2006.
20. D. Boneh, R. DeMillo and R. Lipton. *New Threat Model Breaks Crypto Codes.* Bellcore Press Release, September 25th, 1996.
21. A. Shamir. *Method and apparatus for protecting public key schemes from timing and fault attacks.* US Patent No. 5,991,415, Nov. 23, 1999.
22. L. Goubin, M-L. Akkar and Olivier Ly. *Automatic integration of counter-measure against fault injection attacks*. 2003
23. K. Sachdeva and S. Prevost. *Application Integrity Check During Virtual Machine Runtime.* 2006, US Patent No. 20060047955A1, Mar. 2, 2006.
24. J. Ligatti, M. Abadi, M. Bidiu, U. Erlingsson. *Control Flow integrity.* Proceedings of the 12th ACM Conference on Computer and communications security, 2005.
25. N.Oh, P. P. Shirvani and E. J.McCluskey. *Control Flow Checking by software signature.* IEEE transaction on reliability, Vol. 51, N°2, March 2002.