Rapport technique :
Introduction aux cartes à puce

Nicolas Stouls

Septembre 2006

Dans le cadre de ma thèse *Outils formels pour la spécification et le développement de systèmes*, nous nous sommes intéressés particulièrement aux systèmes embarqués et à leur contraintes de sécurité. En effet, ce domaine est actuellement en très forte expansion et a des besoins de plus en plus importants en sécurité de part son intégration dans des systèmes critiques de la vie courante (carte de paiement, automobile, avionique, etc). De plus, les logiciels embarqués sont particulièrement intéressants à étudier du point de vue de la sécurité, car leurs spécifications sont de petite taille, bien que leur complexité soit en augmentation constante. Dans ce chapitre, nous nous intéressons à un type particulier de système embarqué : les cartes à puce.

Ce rapport technique a été partiellement inspiré par les thèses de L. Casset [Cas02], G. Dufay [Duf03] et G. Grimaud [Gri00] ainsi que par les articles [Tré03] de H. Trézéguet, [Cha05] S. Chaptal et [Ave05] de Y. Avenel. Enfin, certaines informations ont été tirées du cours sur les cartes à puce fait par J-L Lanet (GEMPLUS) et C. Barral (GEMPLUS) à L'ENSIMAG en décembre 2005 [Lan05] et des sites web Wikipedia[1] et commentcamarche [2].

1 Technologies des cartes à puce

Inventée en 1974 par le français Roland Moreno, la carte à puce est issue de travaux conjoints entre des laboratoires français, japonais et allemands. Son interface externe est définit avec exactitude par la norme ISO 7816 [Hus01], qui se décompose en six alinéas :

- ISO-7816-1 : Caractéristiques physiques de la carte (taille) ;
- ISO-7816-2 : Emplacement des contacts électriques ;
- ISO-7816-3 : Nature des signaux électriques et des protocoles de communication entre le terminal et la carte ;

[1] http ://fr.wikipedia.org/wiki/Carte_à_puce
[2] http ://www.commentcamarche.net/pc/rom.php3

- ISO-7816-4 : Organisation des données et sécurisation ;
- ISO-7816-5 : Procédure d'inscription des applications ;
- ISO-7816-6 : Données communes et règles de codage.

Il existe trois variantes de cartes à puce : les cartes à mémoire, les cartes à logiques internes câblée et les cartes à microprocesseur intégré.

Les premières étaient les *cartes à mémoire*. Utilisées notamment pour la téléphonie, ces cartes intègrent un système de fusibles qui peuvent être grillés permettant ainsi de décrémenter le nombre d'unités présentes sur la carte. Ces cartes sont encore utilisées aujourd'hui car elles permettent de ne pas avoir de monnaie dans les appareils téléphonique et donc de diminuer les actes de vandalisme.

Un second type de carte est apparu en dérivé de ce premier : les cartes à *logique interne câblée*. Celles-ci sont assimilables aux cartes à *pistes magnétiques* dans le sens où les données stockées ne peuvent pas être modifiées. Cependant, elles s'en distinguent par la masse d'informations stockables et le fait que le contact peut être maintenu indéfiniment. C'est pourquoi l'une des utilisations les plus connues de ces cartes est le stockage des clefs de déchiffrage pour les décodeurs de chaînes payantes. Si l'on retire la carte, alors le décodage s'arrête rendant impossible l'utilisation de deux décodeurs avec une même carte (un même abonnement).

Enfin, le troisième type de cartes à puce est celui des cartes avec microprocesseur intégré. L'utilisation de ces cartes est très répandue dans la vie de tous les jours, ce qui les rend progressivement indispensables et justifie un développement très rapide dans ce secteur de l'industrie. En effet, on trouve notamment dans cette catégorie les cartes de paiement[3] [DCS01], les cartes SIM[4] [DD04] et les cartes d'identification[5] [SG02, Fla05]. Ces cartes se différencient des autres car elles embarquent un processeur, de la mémoire vive, de la mémoire persistante et parfois des coprocesseurs (cryptographique la plupart du temps). Ainsi, ces cartes sont capables d'exécuter un programme et sont donc assimilables à un ordinateur. Le tableau 1 résume l'évolution des principales caractéristiques de ce type de cartes. Pour comparaison, le tableau 2 donne les caractéristiques de certains ordinateurs et consoles de jeux. Nous remarquerons que les capacités des cartes à puce actuelles sont très proches voir supérieures à celles des ordinateurs d'il y a 15 ans.

Afin de différencier les technologies de mémoire persistante utiliser dans les cartes, voici une description des caractéristiques de chacune d'entre elles.

Les ROM (Read Only Memory) sont des puces gravée directement dans des plaques de silicium et mémorisant des informations de manière perma-

[3] Carte Bleue, Moneo, etc.
[4] Carte d'identification servant pour les téléphones portables.
[5] Carte VITAL, carte d'identité électronique, passeport électronique, etc.

Année	Taille bus	Fréquence (MHz)	RAM	Mémoire persistante
1981	8 bits	4,77	36 octets	1 Ko d'EPROM
1985	8 bits	4,77	128 octets	2 Ko d'EEPROM
1990	8 à 16 bits	4,77	256 octets	8 Ko d'EEPROM
1996	8 à 32 bits	4,77 à 28,16	512 octets	32 Ko de FLASH
2000	8 à 32 bits	4,77 à 28,16	1536 octets	32 Ko de FLASH + 32 Ko d'EEPROM
2002	8 à 32 bits	4,77 à 28.16	4Ko	64 Ko de FLASH + 64 Ko d'EEPROM
2003	8 à 32 bits	4,77 à 66	16Ko	528 Ko de FLASH
2005	8 à 32 bits	4,77 à 100	16Ko	768 Ko de FLASH + 256 Ko Page FLASH
2006	8 à 32 bits	2 processeurs	30Ko	1 Go FLASH USB

TAB. 1 – Evolution des caractéristiques cartes à puce à microprocesseur

nante.

Les PROM (Programmable Read Only Memory) sont des puces pouvant mémoriser des informations de manière permanante. Elles sont constituées de fusibles qui peuvent être grillés pour mémorisé un 0 ou laissés intacts pour mémoriser un 1.

Les EPROM (Erasable Programmable Read Only Memory) sont des PROM pouvant être effacées. Celles-ci possèdent une vitre permettant de laisser passer des rayons ultra-violets. Lorsque la puce est en présence de rayons ultra-violets d'une certaine longueur d'onde les fusibles sont régénérés ce qui fait que tous les bits de la mémoire sont à nouveau à 1. C'est pour cette raison que l'on qualifie ce type de PROM d'effaçable.

De même que les EPROM, les EEPROM (Electrically Erasable Read Only Memory) sont des puces pouvant mémoriser des informations de manière permanante et pouvant être effacées. Cependant, elle s'en distingue par le fait que celles-ci peuvent être effacées par un simple courant électrique.

La mémoire FLASH est une variante de l'EEPROM également appelée ROM Flash ou Flash EPROM. Contrairement aux EEPROM classiques, utilisant 2 à 3 transistors par bit à mémoriser, la Flash EPROM n'utilise qu'un seul transistor. D'autre part l'EEPROM peut-être écrite et lue mot par mot, alors que la Flash ne peut être effacée que par pages (la taille des pages étant en constante diminution). Enfin la densité de la mémoire Flash est plus importante, ce qui permet la réalisation de puces ayant beaucoup plus de mémoire. Des EEPROM sont ainsi préférentiellement utilisées pour la mémorisation de données de configuration et la mémoire FLASH pour des programmes.

La mémoire FLASH-USB est en fait de la mémoire FLASH, connectée au reste du système par un bus USB. Cela permet d'adresser une quantité beaucoup plus importante de zone mémoire.

Appareil	**Taille bus**	**Fréquence**	**RAM**	**Mémoire persistante**
Apple II (1977)	8 bits	1MHz	4Ko à 64Ko	12Ko
Atari 2600 (1977)	8 bits	1.19MHz	128 octets	2Ko à 64Ko
Famicom (Aussi appelée NES - 1983)	8 bits	1,66MHz	16Ko	4Ko à 512Ko
Gameboy (1988)	8 bits	4.19 MHz	8Ko	16Ko à 512Ko
GameGear (1990)	8 bits	3,58MHz	24Ko	4Ko à 512Ko
Macintosh Classic couleur (1993)	16 bits	16MHz	4Mo à 10Mo	40Mo à 160Mo

TAB. 2 – Caractéristiques d'ordinateurs et de consoles de jeux pour comparaison avec les cartes à puce

Notons enfin, que dans le tableau 2 la colonne mémoire persistante fait référence, suivant les cas, à de la ROM ou à des disques durs.

Par la suite, nous allons nous intéresser plus particulièrement aux cartes avec processeur embarqué.

2 Systèmes d'exploitation pour carte à puce et JavaCard

2.1 Systèmes d'exploitation pour carte à puce

Comme tout ordinateur, les cartes à microprocesseur embarqué nécessitent la présence d'un système d'exploitation faisant office d'interface pour l'accès aux ressources de la carte. Le comportement de cet OS[6] est partiellement défini dans la norme ISO-7816 aux alinéas 3 à 6. Par exemple, l'alinéa 7816-4 décrit comment les fichiers peuvent être stockés sous forme d'arborescence de répertoire et de fichiers. Les premières cartes embarquaient un OS propriétaires et un logiciel spécifique à l'application voulue. Cependant, il n'était pas possible de désinstaller un logiciel et les OS étaient souvent adaptés pour être dédié à l'application embarquée. L'une des conséquence de cet état de faits était l'impossibilité de développer une carte à puce pour une application ne permettant pas de vendre suffisamment d'unités pour amortir le coût de conception. C'est notamment le cas des cartes d'accès à des batiments qui étaient développés en détournant des cartes conçues pour des applications bancaires.

C'est pourquoi, un certain nombre de systèmes d'exploitation dits *ouvert*, permettant d'installer et de désinstaller des applications, ont alors progressivement vu le jour. Le premier à se démarquer fut **JavaCard** [SM03, BDJ+01] de Sun, basé sur le langage **Java**, et dont la spécification évolue selon les be-

[6] OS = Operating System (Système d'exploitation)

soins des industriels présents dans le *JavaCard Forum*[7].

Depuis, d'autres systèmes lui ont emboîté le pas comme *Windows for Smart Card* [Tal99] de Microsoft, basé sur le langage Visual Basic, *MultOS* [dL99] repris par le consortium *MAOSCO* (MasterCard, Mondex, Europay et Discover) et basé sur le langage MEL (MULTOS Executable Language) ou *Camille* [Gri00, GHSR05] de l'Université de Lille et basé sur le langage intermédiaire *Façade*.

Depuis son lancement en 1997, **JavaCard** a été largement utilisé dans de nombreux domaines, y compris la carte à puce. Pour donner un exemple, dans le domaine des cartes SIM, **JavaCard** représentais 24% du marché en 2003, 43% en 2004 (450 millions d'unités) et en 2005 la barre des un milliard de cartes SIM embarquant cet OS depuis sa sortie a été franchie. Aujourd'hui c'est le système d'exploitation pour carte à puce le plus utilisé dans le monde [Ave05]. La popularité de **JavaCard** vient du fait que sa spécification n'est pas définie par le propriétaire des droits (Sun) mais par un consortium d'industriels, ce qui n'est pas le cas de Windows for Smart Cards, qui est un OS clef en main. De plus, les spécifications sont publiques, ce qui laisse chaque fondeur de carte à puce développer sa propre version de **JavaCard**, ce qui apporte une valeur ajoutée au produit.

2.2 Cas du JavaCard

Le système **JavaCard** considère les applications (applettes) comme des objets, ce qui permet de gérer plusieurs applications sur une même carte et de désinstaller des programmes. Ces applettes sont écrites en **JavaCard**, sous-ensemble du langage **Java**, et compilées en un Bytecode qui est interprété par une machine virtuelle embarquée dans l'OS. Le tableau 3 résume les principales différences entre le langage **Java** et la version actuelle de **JavaCard** (2.2.1).

Fonctionnalités **Java** supportées	Fonctionnalités **Java** non supportées
Types de base : booléens, byte, short	Types de base : long, double, float
Tableaux à une dimension	Tableau à plusieurs dimensions
Packages, classes, interfaces et exceptions	Caractères et chaînes de caractères
Création dynamique d'objets	
Héritage, méthodes virtuelles et surcharge	

TAB. 3 – Principales différences entre les langages **Java** et **JavaCard** (V2.2.1)

De plus, les limitations dûes aux cartes à puce font que certaines fonctionnalités de la machine virtuelle **Java** ne sont pas supportées par **JavaCard**.

[7] http ://www.javacardforum.org

Par exemple, une classe ne peut pas être chargée dynamiquement puisque l'installation se fait sur un terminal particulier. Le Security Manager est alors remplacé par un simple firewall qui garanti qu'une applette n'accède pas à une zone mémoire qui ne lui appartient pas. De même, le ramasse miettes est inutile puisqu'il suffit décrire en RAM les objets temporaires et en mémoire persistantes les objets permanants. Ensuite, la limitation de l'espace mémoire fait que la création d'objets n'est que très rare, ce qui rend le clonage inutile. Enfin, l'embarquement du vérificateur de Bytecode étant trop couteuse en place, celui-ci est vérifié avant son chargement. Afin de garantir que le code n'a pas été altéré depuis la vérification, celui-ci est embarqué avec une signature. Une technique de signature consiste à embarquer une preuve de la correction de l'applette [RB00, Nec97], car il est très peu coûteux de vérifier qu'une peuve est vraie, contrairement à une vérification sans informations. Le tableau 4 résume les principales différences de spécificités entre les machines virtuelles de **Java** et **JavaCard**.

Spécificités **JavaCard**	Fonctionnalités **Java** non supportées
FireWall	Security Manager
Controle d'accès dans VM	Chargement dynamique de classes
Maitrise du partage	Multi-tâches
Objets temporaires en RAM	Ramasse miettes et finalisation
Objets persistants	sérialisation d'objets
Atomicité des transactions	Clonage d'objets
Vérification du Bytecode au chargement	
(Signature, proof carrying code)	

TAB. 4 – Principales différences de spécificités entre **JavaCard** et **Java**

JavaCard fournit également un certain nombre d'API[8] permettant de simplifier le travail et les connaissance techniques nécessaires des développeurs. Par exemple, lors d'une communication entre une carte (le serveur) et un terminal (le client), seul un "fil" (un connecteur) est utilisé, nécessitant donc une communication asynchrone semi-duplex entre les deux partis. Cette communication a lieu par l'échange de messages, nommés APDU[9], dont la forme est décrite par la norme ISO-7816-3. **JavaCard** fournit une API permettant d'accéder aux APDU au travers d'objets, simplifiant la récupération et l'émission des paramètres. Dans sa version actuelle (2.2.1), **JavaCard** fournit également une seconde API, nommée RMI[10], permettant d'accèder aux méthodes de la carte par des appels d'opération classiques et masquant ainsi l'utilisation des APDU. Enfin, une autre API intéressante fournie avec **JavaCard** est celle nommée Open Platform [BWT02], déve-

[8] API = Application and Programming Interface
[9] APDU = Application Protocol Data Unit
[10] RMI = Remote Methode Invocation

loppée par VISA, et mettant à disposition toute une interface de gestion du chiffrage, du déchiffrage et de la signature d'un message ainsi que la gestion des clefs.

Parmi les contraintes liées aux cartes à puce, se trouve également le risque d'arrachage de la carte [SL00, SL99]. Or certaines transactions sensibles telles que l'authentification ou les opérations sur le solde d'un porte monnaie électronique nécessitent d'avoir la garantie que soit l'opération est effectuée complètement, soit que rien n'a changé. Cette garantie peut être acquise par l'appel successif de deux opérations. La première consiste à demander si une action est faisable et la seconde permet d'exécuter cette action le cas échéant. C'est pourquoi la classe `JCSystem` de **JavaCard** met à disposition trois opérations (`beginTransaction()`, `commitTransaction()` et `abortTransaction()` permettant de gérer automatiquement ce type de transactions en gérant la persistance des données temporaires et la cohérence du système.

Enfin, le langage **JavaCard** dispose d'un certain nombre de sécurités permettant d'avoir confiance en les applettes installées [Req00, Duf03], bien que le Security Manager ne soit pas embarqué sur la carte. Par exemple, l'exécution des applettes se fait dans un contexte personnel dont elles ne peuvent pas sortir et où personne d'autre ne peut lire ou écrire (la Sandbox) [Hen01]. Par contre, une partie de cette zone peut être partagée volontairement avec une autre applette. Cette protection est effectuée par un FireWall qui permet de vérifier dynamiquement la légalité des accès. Enfin, la levée d'une exception permet de renvoyer un APDU d'erreur de manière transparente. Parmi les techniques proposées pour garantir la sécurité du code chargé on trouve notamment le développement formel d'un vérificateur de Bytecode chargeable sur carte [BCR03].

2.3 Cycle de vie d'une applet JavaCard

Comme tout programme **Java**, la compilation d'une applette **JavaCard** donne un fichier *class*, décri dans un langage de Bytecode. Cependant, le Bytecode est un langage contenant beaucoup de chaînes de caractères (les noms des méthodes par exemple), alors que la place mémoire est très limitée sur une carte. Comme le chargement dynamique n'est pas autorisé, on peut résoudre à la compilation les dépendances de classes et il devient possible de remplacer ces chaines par des valeurs numériques, beaucoup plus légères à stocker, que l'on appelle des *tokens*[11]. A l'issu de ce remplacement, deux fichiers sont crées : un fichier *cap* contennt le Bytecode tokenisé et un fichier *export* contenant l'index des correspondances token / nom de méthode.

[11] Token = jeton. Dans le cas du remplacement des noms de méthode (*tokenisation*), les tokens sont des valeurs numériques.

Notons que seul le fichier cap sera chargé sur la carte. Lors de cette conversion un certain nombre de vérifications de sécurité supplémentaires vont être effectuées car, contrairement à **Java**, le vérificateur de Bytecode n'est pas embarqué sur la carte.

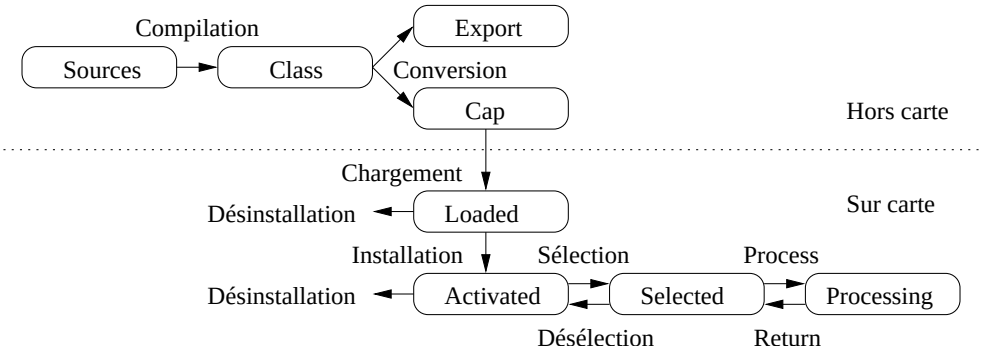

FIG. 1 – Cycle de vie d'une applet **JavaCard**

Une fois le fichier cap produit il peut être chargé sur la carte, puis installé (par appel de la méthode `install()` du système). Cette phase instancie les objets correspondants à l'application et enregistre l'instance auprès du système, rendant ainsi la nouvelle applette opérationnelle. Pour l'exécuter il faut alors la sélectionner puis appeler ses méthodes. Cette dernière phase se fait par l'échange d'APDUs. C'est la méthode `process(APDU)` de l'applette qui reçoit et traite les messages entrants. Enfin, il est possible de désinstaller une applette lorsqu'elle n'est pas sélectionnée. Le cycle de vie d'une applet **JavaCard** est décrit en figure 1.

3 Sécurité des cartes à puce

Les cartes à puce sont principalement utilisées des domaines où les besoins en sécurité et en miniaturisation sont très importants. Les quatre domaines les plus importants sont [Hen01] : le paiement, la téléphonie, le contrôle d'accès et la gestion des informations. La sécurité des cartes passe alors par la protection contre les attaque [BECN+04, BG01] matérielles et logicielles. Voici quelques exemples d'attaques ayant existé ou existantes :

- Fausse devanture de terminal
- Attaques invasives (avec démontage de la carte) [SS99]
- Attaques statistiques pour retrouver des secrets [CKN01]
- Analyse du flux électromagnétique [GMO01, QS01]
- Attaques basées sur le retrait de la carte [HP04]
- Attaques basées sur la recherche de canaux caché temporels [Koc96, KK99, MAC+02, DKL+99]

- Déduction des secrets basée sur l'analyse de la consommation de la carte [KJJ99]
- Attaques par induction de faute, dans le but de modifier le comportement du système [GT04, SA02]

La sécurité dans les cartes à puce se base principalement sur les attaques connues et des principes de bonne conduite. Par exemple, la première règle de base est que toute donnée secrète doit rester dans la carte et être traitée par la carte. Cependant, il existe, comme pour toute règle des exceptions. Ainsi, la carte SIM des téléphones portables déroge à celle-ci en transmettant une clef secrète au téléphone, car elle n'est pas assez puissante pour effectuer seule l'opération de chiffrement/déchiffrement à la volée de la communication, qui est donc réalisée par le téléphone.

Enfin, notons que les principales composantes de la sécurité d'un logiciel pour carte à puce sont les trois suivantes :

- Confidentialité : préservation des secrêts
- Non répudiation : garantie de l'identité de l'utilisateur
- Intégrité : protection contre l'altération des données

L'étude de cas DEMONEY que nous avons réalisé (Chapitre **??**) illustre un exemple de modélisation d'un porte monnaie électronique sous **JavaCard** et prend en compte les différentes considérations de sécurité exposées ici.

4 Glossaire

APDU :

Application Protocol Data Unit

API :

Application and Programming Interface

Applette :

Nom donné aux programmes développés en **JavaCard**.

Bytecode :

EPROM :

Erasable Programmable Read Only Memory - Ce sont des PROM pouvant être effacées. Celles-ci possèdent une vitre permettant de laisser passer des rayons ultra-violets. Lorsque la puce est en présence de rayons ultra-violets d'une certaine longueur d'onde les fusibles sont régénérés ce qui fait que tous les bits de la mémoire sont à nouveau à 1. C'est pour cette raison que l'on qualifie ce type de PROM d'effaçable. Page 3.

EEPROM :
De même que les EPROM, les EEPROM (Electrically Erasable Read Only Memory) sont des puces pouvant mémoriser des informations de manière permanante et pouvant être effacées. Cependant, elle s'en distingue par le fait que celles-ci peuvent être effacées par un simple courant électrique. Page 3.

FireWall :
Composant logiciel ou matériel d'un réseau permettant de filtrer et de rediriger les messages.

FLASH :
La mémoire FLASH est une variante de l'EEPROM également appelée ROM Flash ou Flash EPROM. Contrairement aux EEPROM classiques, utilisant 2 à 3 transistors par bit à mémoriser, la Flash EPROM n'utilise qu'un seul transistor. D'autre part l'EEPROM peut-être écrite et lue mot par mot, alors que la Flash ne peut être effacée que par pages (la taille des pages étant en constante diminution). Enfin la densité de la mémoire Flash est plus importante, ce qui permet la réalisation de puces ayant beaucoup plus de mémoire. Des EEPROM sont ainsi préférentiellement utilisées pour la mémorisation de données de configuration et la mémoire FLASH pour des programmes. Page 3.

Flash EPROM :
Cf. mémoire FLASH

FLASH-USB :
La mémoire FLASH-USB est en fait de la mémoire FLASH, connectée au reste du système par un bus USB. Cela permet d'adresser une quantité beaucoup plus importante de zone mémoire. Page 3.

Ordinateur :
Moyen conçu pour accélérer et automatiser les erreurs.

OS :
Operating Système (Système d'exploitation).

PROM :
Programmable Read Only Memory. Ce sont des puces pouvant mémoriser des informations de manière permanante. Elles sont constituées de fusibles qui peuvent être grillés pour mémorisé un 0 ou laissés intacts pour mémoriser un 1. Page 3.

RMI :
Remote Methode Invocation. Nouvelle API de **JavaCard** depuis sa version 2.2.1.

ROM :
Read Only Memory. Ce sont des puces gravée directement dans des plaques de silicium et mémorisant des informations de manière permanante. Page 2.

ROM Flash :

Cf. mémoire FLASH

Sandbox :

Littéralement *bac à sable*. Zone mémoire réservée d'une application carte à puce. Aucune autre application ne peut lire ou écrire dans cette plage d'adresses.

Références

[Ave05] Y. Avenel. Les os pour cartes à puce entre ouverture et diversification. *Electronique*, 163 :34–38, 11 2005.

[BCR03] L. Burdy, L. Casset, and A. Requet. Développement formel d'un vérificateur embarqué de byte-code java. In D. Bert, V. Donzeau-Gouge, and H. Habrias, editors, *Développement rigoureux de logiciel avec la méthode B*, volume 22. Technique et Science Informatiques, 2003.

[BDJ+01] G. Barthe, G. Dufay, L. Jakubiec, B. Serpette, and S. Melo de Sousa. A Formal Executable Semantics of the JavaCard Platform. *Lecture Notes in Computer Science*, 2028 :302, 2001.

[BECN+04] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. In *Workshop on Fault Detection and Tolerance in Cryptography, Italy*, 2004.

[BG01] C. Bidan and P. Girard. La securite des cartes a microprocesseur. *Revue de l'Electricité et de l'Electronique (REE)*, 5 :60–65, 5 2001.

[BWT02] D. Brewer, C. Wang, and P. Tsai. Proving protection profile compliance for the ccl/itri visa open platform smart card. In *3rd InternationalN Common Criteria Conference*, 2002.

[Cas02] L. Casset. *Construction Correcte de Logiciels pour Carte à Puce. Développement formel d'un vérifieur embarqué de byte code Java Card à l'aide de la méthode B.* PhD thesis, Université d'Aix-Marseille II, 2002. http ://www.atelierb.societe.com/liens/theseLudovic.pdf.

[Cha05] S. Chaptal. Les cartes à puce géantes débarquent dans les mobiles. *01net*, 11 2005.

[CKN01] J.J. Coron, P. Kocher, and D. Naccache. Statistics and secret leakage. In *Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 157–173. Springer-Verlag, 2001.

[DCS01] Direction centrale de la sécurité des systèmes d'information DCSSI. *Carte mixe MONEO/CB, rapport de certification 2001/10*, Octobre 2001.

[DD04] C. Demoulin and M. Van Droogenbroeck. Principes de base du fonctionnement du réseau GSM. *Revue de l'AIM*, 04 :page 318, 2004.

[DKL+99] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, and J.-L. Willerns. A practical implementation of the

timing attack. In *Third Smart Card Research And Advanced Application Conference CARDIS'98*. Springer-Verlag, 1999.

[dL99] J. de Langavant. The multos system and architecture. In *Gemplus Developer Conference, Paris*, 1999.

[Duf03] G. Dufay. *Vérification formelle de la plate-forme JavaCard.* thèse, Université de Nice - Sophia Antipolis UFR Sciences, décembre 2003.

[Fla05] M. Flamenbaum. La carte nationale d'identité électronique. Rapport de master 2 droit du cyberspace, Université de Lille 2 Faculté des sciences juridiques politiques et sociales, 2005.

[GHSR05] G. Grimaud, Y. Hodique, and I. Simplot-Ryl. Secure extensible type system for efficient embedded operating system by using metatypes. In *Proc. 1st International Workshop on System and Networking for Smart Objects (SaNSO 2005)*, Fukuoka, Japan, 2005. IEEE Press.

[GMO01] K. Gandol, C. Mourtel, and F. Olivier. Electromagnetic analysis : Concrete results. In *Workshop on Cryptographic Hardware and Embedded Systmes (CHES'2001)*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.

[Gri00] G. Grimaud. *CAMILLE, un système d'exploitation ouvert pour cartes à microprocesseur.* thèse, Laboratoire d'Informatique Fondamentale de Lille, décembre 2000.

[GT04] C. Giraud and H. Thiebeauld. A survey on fault attacks. In *CARDIS'04, Smart Card Research and Advanced Applications VI, Toulouse, France*, pages 159–176. Kluwer academic, 2004.

[Hen01] Mike Hendry. *Smart Card Security and Applications.* Artech House telecomunication library, 2001.

[HP04] E. Hubbers and E. Poll. Reasoning about card tears and transactions in java card. In *Fundamental Approaches to Software Engineering (FASE'2004), Barcelona, Spain*, volume 2984 of *Lecture Notes in Computer Science*, pages 114–128. Springer-Verlag, 2004.

[Hus01] D. Husemann. Standards in the smart card world. *Computer Networks*, 36(4) :476–487, 2001.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397. Springer-Verlag, 1999.

[KK99] O. Kömmerling and M.G. Kuhn. Design principles for tamper-resistant smartcard processors. In *the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA*, pages 9–20, 1999.

[Koc96] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113. Springer-Verlag, 1996.

[Lan05] J-L. Lanet. *(Gemplus) Support de cours : Carte à puce*, 2005. Institut National Polytechnique de Grenoble.

[MAC+02] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving smart card security using self-timed circuits. In *ASYNC'02*, pages 211–218, 2002.

[Nec97] G.C. Necula. Proof-carrying code. In *In 24th Principles of Programming Languages (POPL)*, Paris, France, 1997.

[QS01] J.J. Quisquater and D. Samyde. Electromagnetic analysis (ema) : Measures and countermeasures for smart cards. In *E-smart'01*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2001.

[RB00] A. Requet and G. Bossu. Embedding formally proved code in a smart card : Converting b to c. In *Third IEEE International Conference on Formal Engineering Methods (ICFEM'00)*, pages 15–22, 2000.

[Req00] A Requet. A B Model for Ensuring Soundness of the Java Card Virtual Machine. In *FMICS'2000*, Berlin, March 2000.

[SA02] S. Skorobogatov and R. Anderson. Optical fault induction attacks. In *Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2002), San Francisco Bay (Redwood City), USA*, 2002.

[SG02] E. Soulier and C. Grenier. A political model for the co-operative production of knowledge in the design process : the shared medical file (smf). In *European Conference on Artifical Intelligence : Knowledge Management and Organizational Memories*, 2002.

[SL99] D. Sabatier and P. Lartigue. The Use of the B Formal Method for the Design and the Validation of the Transaction Mechanism for Smart Card Applications. In *FM'99 - Formal Methods - volume 1*, volume 1708 of *Lecture Notes in Computer Science*, pages 348–387. Springer-Verlag, september 1999.

[SL00] D. Sabatier and P. Lartigue. The use of the b formal method for the design and the validation of the transaction mechanism for smart card applications. *Formal Methods in System Design*, 17(3) :245–272, 12 2000.

[SM03] Inc. Sun Microsystems. *Java Card 2.2.1 Virtual Machine Specification*, 10 2003. Santa Clara, California.

[SS99] B. Schneier and Adam Shostack. Breaking up is hard to do : Modeling security threats for smart cards. In *USENIX Workshop on Smart Card Technology*, pages 175–185. USENIX Press, 5 1999.

[Tal99] L. Talvard. Smart card for windows overview. In *Gemplus Developer Conference, Paris*, 1999.

[Tré03] H. Trézéguet. Les microcontrôleurs pour cartes à puce. *Electronique*, 141 :72–81, 11 2003.