



MISC

Multi-System & Internet Security Cookbook

HORS - SERIE

PROGRAMMATION

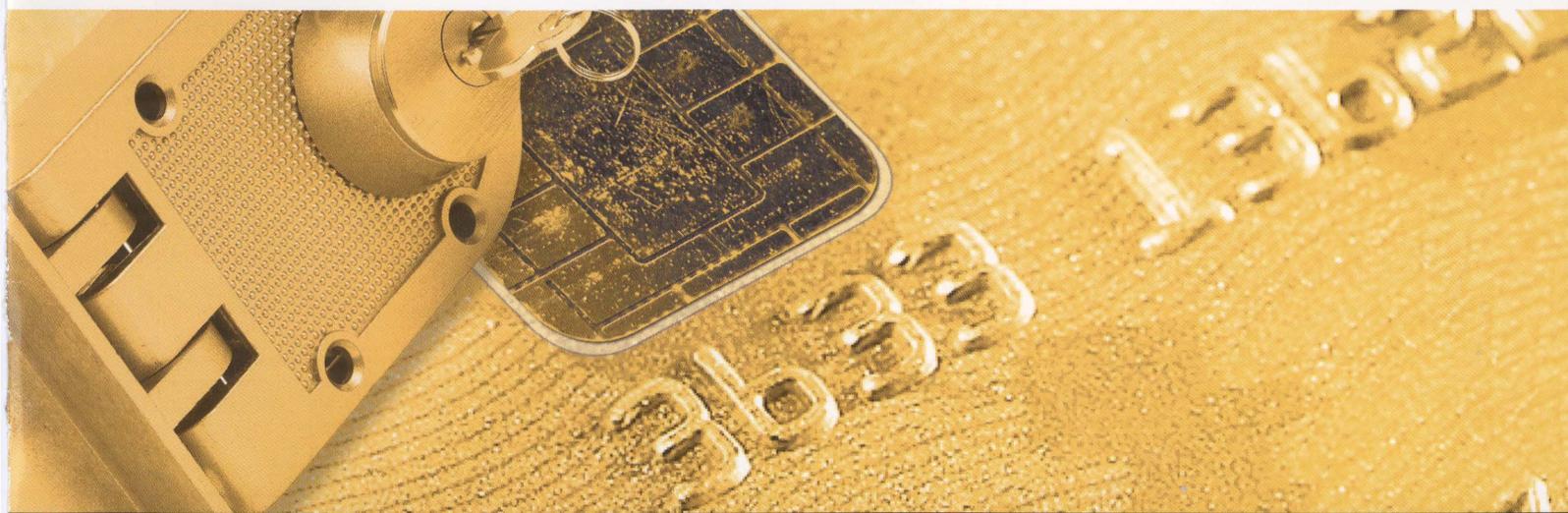
JavaCard ou
comment programmer
une vraie carte à puce



DOSSIER

CARTES À PUCE

DÉCOUVREZ LEURS FONCTIONNALITÉS ET LEURS LIMITES



HISTORIQUE

Retour sur la YesCard, un aperçu
du système bancaire français

L 16844 - 2 H - F : 8,00 € - RD



SÉCURITÉ

Mise en place
d'infrastructures
de gestion de clés
(PKI) utilisant des
cartes à puce

TECHNOLOGIE

MIFARE Classic,
comment une faille
compromet la
sécurité de milliards
de cartes !

Après une longue période de gestation de neuf mois, c'est un beau hors-série de 80 pages qui est là. Le géniteur est fatigué, mais heureux.

Cela faisait longtemps que l'idée d'un dossier *MISC* spécial « carte à puce » était dans l'air. Une gestation en entraînant une autre (...), c'était le moment où jamais de le faire. À ma grande surprise et satisfaction, l'engouement suscité a été tel que le dossier s'est finalement mué en numéro hors-série, puis au final en deux numéros hors-série ! Je vous recommande à ce propos l'excellent hors-série « carte à puce » de *GNU/Linux Magazine France*, en kiosque actuellement. Véritable numéro dual à ce hors-série-ci, il est axé plus pratique et moins théorique.

Je suis honoré de la confiance accordée par Frédéric Raynal, rédacteur en chef de *MISC*, qui m'a laissé gérer de manière totalement libre la réalisation de ce numéro qui me tenait tant à cœur. J'espère que le résultat le satisfait comme il me satisfait.

Mon seul regret est de ne pas avoir eu plus de pages à ma disposition pour pouvoir publier tous les articles que j'aurais voulu. Pas de crainte à avoir, ils ne sont pas perdus. Vous les retrouverez au fil du temps disséminés dans les numéros réguliers de *MISC*.

Le monde de la carte à puce est traditionnellement plutôt discret. Il est assez difficile d'en obtenir des informations si l'on n'est pas déjà dedans. Bien que trentenaire, la technologie des cartes à puce reste par conséquent assez méconnue. Avec ce projet de hors-série, j'ai tenu à rassembler divers articles mettant chacun en lumière une petite partie du sujet, de sorte à démystifier cet objet devenu faussement familier.

Je remercie les auteurs, qui ont accepté de collaborer et permis d'aborder autant de facettes de cette technologie, ainsi que les lecteurs minutieux ayant souhaité rester dans l'ombre.

S'il a longtemps été difficile d'avoir des informations sur la technologie des cartes à puce, il l'a été peut-être encore plus d'arriver à se procurer légalement du matériel à des fins d'expérimentation. Aujourd'hui, des kits sont disponibles sur Internet qui offrent la possibilité aux lecteurs qui le désirent d'acquérir du matériel relatif aux cartes à puce pour manipuler. La disponibilité de tels matériels laisse augurer de futurs articles les mettant en oeuvre.

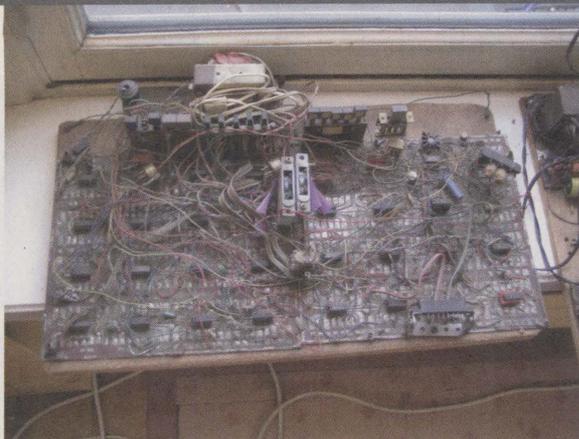
Je tenais également à remercier Roland Moreno, le père de la carte à puce, pour avoir accepté de relire certains articles et traqué l'inexactitude. J'ai été très ému de pouvoir apprécier les pièces originales qui ont été à la source de cette nouvelle industrie. Que de chemin parcouru entre le prototype originel de 1974 et le matériel couramment utilisé aujourd'hui ! (voir les illustrations)

Enfin, je tiens à remercier ma femme qui a supporté mes (trop) longues périodes de travail, finalement nécessaires à la mise au point de ce numéro hors-série, pendant la période difficile de l'apprentissage de la parentalité dans laquelle nous sommes.

Sur ce, je vous souhaite bonnes lectures et manipez bien.

Vincent Guyot

PS : Avec la carte à puce, la sécurité, c'est dans la poche !



SOMMAIRE ▼

HISTORIQUE/INTRODUCTION [4 - 25]

- > Carte à puce : un premier tour d'horizon / 4 → 10
- > Une histoire personnelle de la carte à puce Java / 11 → 15
- > Le marché des cartes à puce / 16 → 17
- > YesCard, entre mythe et réalité ou un aperçu du système bancaire français / 18 → 25

TECHNOLOGIE [26 - 46]

- > La carte SIM ou la sécurité du GSM par la pratique / 26 → 37
- > NXP Mifare Classic : une star déchue / 38 → 46

SÉCURITÉ [48 - 65]

- > Mise en place de PKI et carte à puce / 48 → 57
- > Java Card (U)SIM et applications sécurisées sur téléphones mobiles / 58 → 65

PROGRAMMATION [66 - 80]

- > La Programmation Java Card / 66 → 73
- > Un SDK JavaCard générique ou comment développer une application carte complète pour toutes les cartes Java / 74 → 80

ABONNEMENTS/COMMANDES [47/81/82]

MISC
est édité par **Diamond Editions**
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.ed-diamond.com

www.miscmag.com



Printed in Germany / Imprimé en Allemagne
Dépôt légal : à parution
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K80 190
Périodicité : Bimestrielle
Prix de vente : 8 Euros
A Soleilm

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Relecture : Dominique Grosse
Secrétaire de rédaction: Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08
Impression : Druckhaus Kaufmann (Lahr/Allemagne)
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 61 72 76 24

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans *Misc* est interdite sans accord écrit de la société Diamond Editions. Sauf accord particulier, les manuscrits, photos et dessins adressés à *Misc*, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

Charte du magazine : *MISC* est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de *MISC* une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. *MISC* vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. *MISC* propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

CARTE À PUCE :

UN PREMIER TOUR D'HORIZON

■ mots clés : carte à puce / cryptographie / processeur / sécurité

Le geste est devenu tellement coutumier que nous n'y prêtons même plus attention : nous nous présentons face à un distributeur de billets, introduisons notre carte bancaire dans le lecteur, tapons notre code PIN, et les billets nous tombent dans la main tandis que notre compte bancaire est automatiquement débité. La même carte nous permet d'ailleurs de régler la note d'un restaurant, d'effectuer un achat dans une grande surface, de prendre de l'essence, tandis qu'une autre d'apparence similaire nous permet de nous connecter à l'ordinateur de notre entreprise et qu'une troisième – quoique d'un format un peu différent – protège le cœur de notre téléphone portable. Ce petit objet qui ne pèse que quelques grammes a acquis une telle importance dans nos vies que sa perte s'apparente presque à une catastrophe. Et cette minuscule carte à puce, dont le prix de revient est aujourd'hui ridiculement bas, met pourtant en œuvre une technologie très évoluée, fruit de plusieurs décennies de développement et d'idées ingénieuses.

⇒ 1. Un peu d'histoire

Beaucoup attribuent l'idée originale de la carte à puce au romancier René Barjavel. Dans son roman *La Nuit des temps*, ce dernier imagine en effet pour le peuple de Gondawa une bague permettant – entre autres – de payer automatiquement ses achats. « Chaque fois qu'un Gonda désirait quelque chose de nouveau, des vêtements, un voyage, des objets, il payait avec sa clé. Il pliait le majeur, enfonçait sa clé dans un emplacement prévu à cet effet et son compte, à l'ordinateur central, était aussitôt diminué de la valeur de la marchandise ou du service demandés ». Cela dit, l'histoire des découvertes est rarement linéaire et Roland Moreno, l'un des pères de la carte à puce, n'avait pas lu Barjavel quand l'idée lui est venue. L'idée d'une bague-clé permettant d'accéder à certains lieux se retrouve d'ailleurs dans des romans antérieurs, tels que *Croisière sans escale* (1959) de Brian Aldiss.

Sur un plan moins poétique, la genèse des cartes à puce remonte aux années 40-50, même si l'objet développé à l'époque n'incluait au départ pratiquement aucun artifice technologique.

Il s'agissait simplement de remplacer les cartes en papier ou en carton par un équivalent en plastique, beaucoup plus apte à résister aux détériorations dues à un emploi régulier. Le premier à avoir eu l'idée du principe d'une carte de crédit est un banquier de Brooklyn, John Biggins, qui imagina en 1946 un système permettant aux clients de sa banque de faire des achats « sans cash » auprès des commerçants locaux. Ces derniers transmettaient ensuite les factures à la banque, qui débitait le compte du client. En 1950, fut lancée la carte *Diners Club International*, permettant à ses détenteurs de dîner à crédit dans un certain nombre de restaurants huppés de New York. Outre son indéniable aspect pratique, cette carte, détenue par un cercle très restreint de privilégiés, était également un symbole de prestige, puisqu'elle offrait l'occasion de régler une addition sur la base de sa « bonne réputation ». Des compagnies comme Visa et MasterCard ont ensuite suivi le mouvement, lui donnant un large essor et contribuant largement à l'énorme succès que la carte à puce connaît aujourd'hui.

Au départ, les mesures de protection étaient assez simples : impression en relief (ou « embossage ») des informations propres à l'utilisateur, zone permettant à ce dernier de signer sa carte,... Ces mesures visant à empêcher la contrefaçon reposaient largement sur le soin avec lequel le commerçant allait examiner la carte. Cependant, la prolifération de ce moyen de paiement a rapidement entraîné le besoin de mesures de sécurité plus élaborées, ainsi que de la possibilité de traiter les données de manière automatisée. La première amélioration fut l'apparition d'une piste magnétique au dos de la carte, permettant l'enregistrement de données numériques sur cette dernière et donc le traitement automatique par une machine. Cette piste magnétique souffre pourtant d'une faiblesse cruciale : si elle peut effectivement stocker des données, celles-ci peuvent être lues et même réécrites – et donc modifiées – avec du matériel assez simple. Dès lors, il est par exemple assez facile de lire les données se trouvant sur une carte et de les écrire sur une autre, créant ainsi un clone de la première qui trompera au moins le commerçant peu attentif ou la machine qui ne vérifiera pas les autres informations. Diverses techniques, comme l'utilisation d'un code PIN (*Personal Identification Number*) qui n'est pas inscrit sur la piste magnétique, ont permis de limiter ce risque, mais, à long terme, la solution n'était pas entièrement satisfaisante.

Les développements de la micro-électronique dans les années 70 rendent possible l'intégration de circuits à la carte à puce, et cette idée naît presque simultanément en différents endroits du monde : en 1968, les chercheurs allemands Jürgen Dethloff et Helmut Gröttrup introduisent une demande de brevet pour un tel concept de circuit électronique intégré à une carte. En 1978, un brevet similaire est déposé au Japon par Kunitaka Arimura, tandis qu'en France Roland Moreno dépose, entre 1974 et 1978, 47 brevets dans 11 pays du monde. Beaucoup considèrent d'ailleurs Moreno comme le véritable père de la carte à puce. Michel Ugon et Louis Guillou font ensuite les recherches et démarches de standardisation. La première carte à puce, le CP8 (Circuit Portatif des années 80), est créée par Bull en 1979, mais le premier déploiement d'envergure de cartes à puce n'intervient que vers 1983-1984, lorsque les PTT françaises

...on pourrait définir la carte à puce comme un petit ordinateur blindé...



Le prototype de puce portable initialement imaginé par Moreno

lancent les premières cartes téléphoniques prépayées à mémoire munies du circuit inhibiteur de Roland Moreno.

L'étape suivante consiste à doter la carte d'un véritable microprocesseur, pour lui permettre non seulement de stocker des données, mais également d'effectuer des opérations sur ces dernières. En 1984, le GIE carte bancaire introduisit la fameuse « carte bleue » en France et fut peu à peu suivi par ses voisins européens. Parallèlement, en 1988, les services des postes allemands lancèrent une carte à puce à microprocesseur dans le cadre du réseau de téléphonie mobile C-Netz. Cette première génération de téléphonie mobile ne rencontra qu'un succès assez limité et fut supplantée dans les années 90 par la technologie GSM que nous connaissons actuellement. Cependant, le succès de la carte à puce en tant qu'élément-clé de la sécurité de ce réseau n'est certainement pas étranger à l'introduction de cette même carte, sous la forme d'une carte SIM (*Subscriber Identity Module*), dans tous les téléphones mobiles GSM actuels. Cette utilisation représente l'un des premiers usages massifs de la carte à puce à microprocesseur. Jean-Jacques Quisquater dotera la carte à puce de cryptographie solide, par le DES (*Data Encryption Standard*) en 1985 et le RSA en 1988.



2. La cryptographie à portée de main

La carte à puce à microprocesseur apporte une solution élégante et efficace à un problème extrêmement important. Pour comprendre lequel, prenons par exemple le cas de l'authentification. L'une des plus anciennes techniques permettant de prouver qui je suis consiste à utiliser un mot de passe connu de moi seul. Cette technique présente cependant un défaut majeur, puisque le seul moyen que j'ai de prouver à un tiers que je connais le mot de passe consiste à le lui révéler. Dès cet

instant, comment être sûr qu'il ne va pas à son tour utiliser le mot de passe pour se faire passer pour moi ? La cryptographie moderne résout ce problème en offrant des techniques permettant de prouver la connaissance d'un secret sans révéler le secret lui-même.

Malheureusement, la cryptographie implique la manipulation de très grands nombres et il est assez difficile pour un être humain de retenir par cœur une clé de 300 chiffres ou de multiplier

⇒ Cinq minutes de cryptographie

La cryptographie est la discipline qui étudie les moyens d'assurer la confidentialité et l'intégrité de l'information. Bien que cette question soit un sujet de préoccupation depuis l'antiquité, la cryptographie moderne n'a vu le jour que dans la deuxième moitié du XX^{ème} siècle.

Parmi les (nombreuses) fonctions remplies par la cryptographie, on trouve par exemple le chiffrement, permettant de protéger le contenu d'un message contre les regards indiscrets, la signature, permettant de prouver au destinataire qu'un message a bien été écrit par une personne donnée et n'a pas été modifié entre-temps, et l'authentification, permettant de prouver son identité à un tiers.

La cryptographie moderne est régie par le célèbre principe de Kerckhoffs, qui stipule en substance que la sécurité d'un bon système ne doit pas reposer sur le caractère secret de l'algorithme utilisé, mais uniquement sur le secret d'un paramètre, appelé « la clé ». En d'autres termes, un grand nombre d'utilisateurs peuvent très bien utiliser exactement le même procédé cryptographique : du moment que des clés différentes (et gardées secrètes) sont utilisées, les données manipulées par un groupe sont parfaitement protégées vis-à-vis des autres utilisateurs. Ce principe est réellement appliqué en pratique, et la grande majorité des systèmes de sécurité actuels utilisent des algorithmes standardisés et publiquement décrits, combinés avec des clés connues seulement des personnes autorisées.

Jusqu'au milieu des années 70, les seuls principes connus reposaient sur la cryptographie symétrique, dans laquelle une même clé est utilisée pour chiffrer (ou authentifier,...) et déchiffrer (ou vérifier,...) un message. En 1976, Diffie et Hellman introduisirent la cryptographie asymétrique, basée sur une paire de clés, traditionnellement appelées « clé publique » et « clé privée ». Si n'importe qui est capable de chiffrer un message au moyen de la clé publique, seul le détenteur de la clé secrète correspondante est capable d'effectuer l'opération inverse et donc de retrouver le message clair. Il n'est bien évidemment pas possible de retrouver une clé secrète à partir de la clé publique correspondante. La même dualité est possible avec la signature, la clé privée étant utilisée pour signer un message, dont l'authenticité peut ensuite être vérifiée par toute personne en possession de la clé publique correspondante.

La cryptographie asymétrique offre de nombreux avantages pratiques sur la cryptographie symétrique, mais sa mise en œuvre est malheureusement beaucoup plus coûteuse – typiquement de l'ordre de 1000 fois – en termes de puissance de calcul.

Parmi les systèmes de cryptographie largement répandus à ce jour, citons par exemple AES (*Advanced Encryption Standard*), fonction de chiffrement symétrique, et RSA (du nom de ses inventeurs, Rivest, Shamir et Adleman), utilisée pour la signature ou le chiffrement asymétrique.

mentalement deux nombres de cette taille ! L'utilisation d'un ordinateur résout ce problème, mais, même à notre époque de miniaturisation, avoir en permanence un ordinateur sur soi reste un peu contraignant. En outre, un ordinateur moderne est une machine largement multifonctionnelle : s'il doit protéger mon compte bancaire ou l'accès à mes données médicales, comment puis-je être sûr que l'un des très nombreux programmes qui y sont installés ne contient pas en fait un virus chargé d'espionner les secrets ? Et que faire si je permets à mon voisin de s'en servir pour taper une lettre ou si je laisse la machine quelques minutes sans surveillance ? Un indiscret ne va-t-il pas en profiter pour récupérer en douce les précieux sésames ?

La carte à puce résout ce double problème. En trois mots, on pourrait la définir comme un petit ordinateur blindé. Ordinateur d'abord, puisqu'elle possède bien un processeur, de la mémoire vive, de la mémoire rémanente et un périphérique d'entrée-sortie, comme n'importe quel ordinateur. Blindé ensuite, puisque la carte à puce est typiquement constituée d'une seule pièce de silicium entourée de blindages et de capteurs empêchant d'accéder à ses données internes. Son utilisation est en outre la plupart du temps protégée par un code PIN limitant l'accès à son seul détenteur. Petit enfin, tant par sa taille, qui permet de la transporter aisément dans son portefeuille et donc de l'avoir toujours sous la main, que par ses capacités : la carte à puce a pour vocation de contenir un ou quelques programmes sécurisés, mais pas de devenir une plateforme multifonctionnelle hébergeant des logiciels en tous genres.

Le rôle de la carte à puce est donc d'effectuer des opérations cryptographiques : elle manipule des clés secrètes pour effectuer des opérations sur des valeurs transmises en entrée-sortie, mais les clés secrètes elles-mêmes ne quittent jamais la puce. Cette précision est importante : dans la plupart des cas, même l'utilisateur légitime de la carte n'a pas la possibilité de récupérer la clé secrète qu'elle contient. Il est d'ailleurs à noter que certaines cartes sont capables de générer leurs clés secrètes elles-mêmes : dans certains scénarios, la clé naît, vit et meurt donc dans la puce, sans jamais la quitter.

La carte peut ainsi typiquement exécuter des opérations de chiffrement, générer des signatures électroniques ou participer à un processus d'authentification en pouvant toujours se reposer sur les solides fondations de la cryptographie moderne. À partir de ces quelques briques de base, un très large éventail d'applications sécurisées peuvent être construites, les seules limites étant l'imagination des concepteurs et la puissance des cartes.

3. Les membres de la famille

Comme nous l'avons vu, plusieurs générations de cartes se sont succédées au cours du temps et coexistent encore aujourd'hui. Les cartes à simple piste magnétique ne contiennent qu'une zone de stockage non protégée accessible en lecture et en écriture (la bande noire au dos de la carte). N'ayant aucun composant électronique, il ne s'agit pas véritablement de cartes à puce. Les cartes à puce à mémoire sont en fait une version plus évoluée des précédentes : elles ne peuvent elles aussi stocker des données, mais sont dotées de moyens de protection empêchant leur manipulation. Elles sont typiquement appropriées pour tout usage dans lequel leur détenteur reçoit des « jetons » à dépenser, comme pour la carte téléphonique française ou en tant que carnet de tickets pour les transports en commun. Un autre exemple d'usage est la carte SIS (Système Information Sociale) belge, utilisée pour stocker les informations relatives à la sécurité sociale. Enfin, les cartes à microprocesseur sont dotées d'une véritable puissance de calcul et peuvent donc réellement traiter les données transmises, ce qui ouvre la voie à bien des applications. La carte à microprocesseur est d'ailleurs la seule qui mérite vraiment le nom de *smart card* en anglais et c'est principalement à elle que nous allons nous intéresser.

Une autre distinction fréquemment faite concerne le mode de communication avec la carte : les cartes à contact doivent être

...les cartes à microprocesseur sont dotées d'une véritable puissance de calcul et peuvent donc réellement traiter les données...

physiquement insérées dans un lecteur pour pouvoir fonctionner (comme nous le verrons ci-dessous, ceci n'est pas forcément un inconvénient) ; les cartes sans contact, en revanche, peuvent communiquer par radiofréquence avec un lecteur situé à proximité. Elles s'apparentent alors aux tags RFID (*Radio Frequency Identification*), qui ont fait l'objet d'un dossier dans le numéro 33 de *MISC*. Tracer une frontière nette entre carte à puce sans contact et tag RFID n'est d'ailleurs pas chose aisée. Disons simplement que ces derniers sont capables de communiquer à des distances maximales diverses, de quelques centimètres à quelques mètres, selon le modèle, et opèrent à des fréquences variées, tandis qu'une carte à puce sans contact nécessite d'être à courte distance de son lecteur, typiquement une dizaine de centimètres, et n'opère qu'à une fréquence de 13,56 MHz. Les tags RFID sont aussi typiquement moins puissants que les cartes à puce sans contact,

même si des tags « intelligents » récents tendent à gommer cette différence. En fait, une carte à puce sans contact utilisée pour un protocole d'identification par radiofréquence peut probablement être considérée comme un membre de la famille RFID.

Une carte à puce n'est pas forcément exclusivement ou bien à contact ou bien sans contact : des cartes hybrides, dotées de deux puces, une avec et une sans contact, ou des cartes à interface duale, dont la puce peut fonctionner dans les deux modes, existent également.

4. Structure d'une carte à puce

Intéressons-nous d'un peu plus près à l'intérieur d'une carte à puce. Comme nous l'avons dit, elle est équipée d'un microprocesseur 8-, 16- ou 32-bits, de RAM (entre 128 bytes et 1-2 kilobytes), de quelques kilobytes de ROM pour le stockage de données permanentes telles que le système d'exploitation ou le code des applications, et de quelques kilo-octets d'EEPROM (*Electrically-Erasable Programmable Read-Only Memory*), ou d'une technologie équivalente, permettant l'enregistrement de données qui seront conservées après coupure de l'alimentation. L'EEPROM joue donc le rôle de disque dur de la carte. À ces composants est souvent adjoint un cryptoprocésseur permettant l'exécution plus rapide de primitives cryptographiques telles que l'AES ou le RSA.

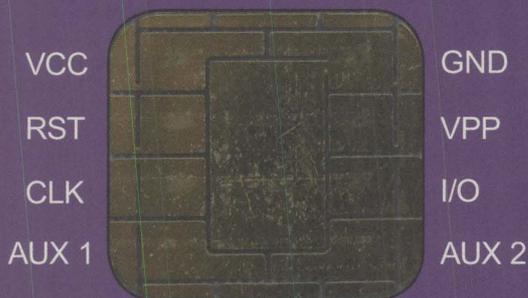
La carte à puce est donc bien un processeur exécutant des logiciels. Typiquement, le développeur désireux de concevoir une nouvelle application choisira une carte à puce dont la puissance, la sécurité et le prix de revient correspondent à ses besoins et, à l'aide d'un kit de développement, écrira le code de cette dernière.

Certaines applications ou certains modules sont développés en langage d'assemblage pour des raisons d'efficacité, mais des compilateurs de langages de haut niveau (C, Java,...) sont également disponibles.

Un ordinateur n'est pas très utile s'il ne peut communiquer avec le monde extérieur. La surface de la puce d'une carte à contact est divisée en huit zones servant de points de connexion lors de l'insertion de la carte dans un lecteur (voir Figure 2). Les fonctions et positions de ces différents contacts sont bien évidemment normalisées, afin de permettre une large interopérabilité entre différents modèles de cartes et de lecteurs. L'un de ces contacts est réservé aux communications avec la carte. Parmi les autres, citons VCC, utilisé par le lecteur pour fournir un courant d'alimentation à la carte, et CLK, permettant de lui transmettre les tops d'horloge cadencant le fonctionnement de son processeur. Une carte à puce n'a en effet typiquement ni batterie, ni horloge interne et dépend entièrement du lecteur dans lequel elle est insérée pour ces deux ressources.

HISTORIQUE

INTRODUCTION



Vcc : tension d'alimentation
RST : signal RESET
CLK : signal d'horloge
GND : connexion à la masse

VPP : tension de programmation de la carte (obsolète)
I/O : port d'entrée/sortie série
AUX1 et 2 : réservés pour usage futur

Les choses sont évidemment différentes pour la carte sans contact, qui devra par définition être capable de communiquer et de traiter les données sans lien physique avec le lecteur. Pourtant, ces cartes ne sont typiquement pas non plus équipées de batteries. Leur alimentation est fournie par le champ électromagnétique émis par le lecteur avec lequel elles communiquent et la transmission de données s'opère via ce même champ électromagnétique, soit directement pour la communication du lecteur vers la carte, soit en modifiant (par exemple en modulant l'amplitude) le champ produit par le lecteur pour la communication de la carte vers le lecteur.

Le prix de revient d'une carte à puce varie entre quelques euros pour les plus performantes et quelques centimes pour les plus élémentaires (dans certains cas, l'impression en couleur de la carte plastique coûte plus cher que la puce elle-même !). Les opérations cryptographiques que la carte sera capable d'exécuter dépendent bien évidemment de sa puissance, mais la cryptographie est en fait une application moins gourmande qu'on ne pourrait le penser. Si seule une carte haut de gamme peut espérer effectuer une signature RSA en un temps raisonnable, même les plus simples seront capables d'exécuter des primitives de chiffrement symétrique comme l'AES, qui correspond au standard actuel de sécurité en matière de chiffrement. Les primitives cryptographiques les plus robustes sont donc à la portée des cartes à puce.

2 Les huit points de contact d'une carte à puce tels que définis par la norme ISO 7816 (l'usure provoquée par les contacts est bien visible sur l'image).

5. Un outil très polyvalent...

Les domaines d'application de la carte à puce sont très variés. Parmi les principaux, citons par exemple :

- ⇒ le paiement et la gestion de données bancaires ;
- ⇒ l'identification, via une carte d'accès d'entreprise, une carte d'identité ou un passeport électronique,...
- ⇒ l'authentification, un nombre croissant d'ordinateurs étant par exemple équipés d'un lecteur de cartes à puce permettant la *login* et la connexion à des sites sécurisés ;
- ⇒ la téléphonie mobile, où elle permet de séparer le téléphone portable de l'abonnement téléphonique (tous ceux qui ont utilisé leur carte SIM dans le téléphone d'un ami pour résoudre un problème de batterie déchargée en ont fait l'expérience) ;
- ⇒ le porte-monnaie électronique, variante de la carte de paiement dans le sens où elle n'ordonne pas un transfert de fonds à un

serveur central, mais est « chargée » d'argent électronique qu'elle dépense ensuite en transférant les « pièces de monnaie » virtuelles dans une carte similaire située dans le lecteur ;

- ⇒ le domaine de la santé, soit pour identifier plus rapidement un patient, soit pour stocker directement ses données médicales (bien que les capacités mémoire des cartes actuelles soient probablement encore un peu faibles pour cet usage) ;
- ⇒ la signature électronique, permettant l'authentification de documents numériques, comme le propose par exemple la nouvelle carte d'identité belge.

Cette liste est bien entendue loin d'être exhaustive, et le champ d'application ne cesse de s'étendre, à mesure que la carte à puce fait ses preuves dans les divers domaines et que ses capacités, tant en mémoire qu'en puissance de calcul, continuent de croître.

6. ...mais rien de plus qu'un outil

Si la carte à puce présente d'indéniables et très importants atouts de sécurité, elle n'est pas pour autant la panacée et il y a lieu de bien comprendre les limites de la sécurité qu'elle offre. Ces limites sont de deux ordres. D'une part, il ne s'agit

bien évidemment pas d'une technologie parfaite et les objectifs d'invulnérabilité voulus par ses concepteurs ne sont donc pas totalement rencontrés. En d'autres termes, il est parfois possible d'attaquer une carte à puce et de lui faire révéler ses secrets.

D'autre part, même si l'on considérait la carte à puce comme un objet parfaitement inviolable, son utilisation se place dans un contexte plus large : la carte n'est qu'un élément dans un système de sécurité. Comme nous allons le voir, certaines caractéristiques intrinsèques de la carte à puce créent de possibles failles externes à cette dernière.

⇒ 6.1 Attaques physiques

L'une des principales menaces contre l'inviolabilité de la carte à puce est celle des attaques physiques. L'idée la plus simple d'attaque de ce genre consiste à « gratter » la couche de protection de la carte afin d'observer – par exemple au microscope – les cellules mémoire dans lesquelles est inscrite la clé ou de poser une mini-sonde sur le bus reliant mémoire et processeur afin d'espionner le transfert des données. Les protections typiques contre ce type d'attaque utilisent des capteurs destinés à détecter l'altération du blindage afin de réagir en conséquence (par exemple en effaçant les clés) ou, plus passivement, chiffrent l'ensemble des données stockées et transitant sur la carte, ce qui rend leur observation inutile. Les données ne sont déchiffrées qu'au moment où elles doivent être utilisées.

Des attaques plus subtiles, apparues durant les années 90, visent à exploiter les caractéristiques physiques du processeur pour déduire de son comportement des informations sur les données qu'il manipule, parmi lesquelles la précieuse clé (voir encadré). On peut par exemple exploiter ainsi le temps de calcul (*timing attack*), la consommation électrique de la carte (*power analysis attack*) ou encore les radiations électromagnétiques qu'elle émet durant son fonctionnement (*electromagnetic analysis attack*). Une autre possibilité (attaques par fautes) consiste à induire délibérément des erreurs dans la carte, par exemple en lui fournissant des voltages anormalement bas ou élevés ou en accélérant l'horloge du processeur. L'objectif est d'amener la carte à produire des résultats incorrects révélant de l'information sur les données manipulées. Ici encore, un arsenal de moyens de protection a été élaboré au cours des années par les différents concepteurs de cartes à puce, mais le combat entre des cartes de plus en plus résistantes et des adversaires de plus en plus ingénieux n'est pas encore terminé.

⇒ 6.2 La carte à puce, un composant d'un système

Pour bien comprendre en quoi la carte à puce n'est qu'un élément d'un système de sécurité, prenons par exemple le cas d'un paiement électronique à la caisse d'un magasin. Au moment d'introduire mon code PIN dans le lecteur, je fais bien sûr confiance à ma carte bancaire pour ne pas débiter mon compte de plus que le montant de mon achat et pour ne transmettre au commerçant que les informations dont il a besoin pour être assuré du paiement. Cependant, même si ma carte à puce est la

⇒ Attaques par canaux cachés : une analogie

Pour comprendre l'idée des attaques par canaux cachés, transposons-nous dans un autre contexte et imaginons le jeu suivant : j'ai devant moi deux pots dans lesquels un adversaire va cacher respectivement les sommes de 100 et 37 euros. Je pourrai ensuite ouvrir l'un des deux pots et empocher son contenu, mon objectif étant bien évidemment de gagner les 100 euros. Pour ce faire, j'ai le droit de poser une question à mon adversaire et je lui demande de calculer 46 fois le montant contenu dans le premier pot plus 10 fois le montant contenu dans le second, puis de me révéler si le résultat est pair. Si je me contente d'observer la réponse finale, je n'apprends bien évidemment rien puisque les deux résultats possibles, à savoir $46 \times 100 + 10 \times 37 = 4970$ et $46 \times 37 + 10 \times 100 = 2702$ sont tous les deux pairs. En revanche, si je puis observer mon adversaire pendant qu'il calcule la réponse, j'ai de bonnes chances d'apprendre de l'information : en effet, si le premier calcul est assez simple à effectuer mentalement, le second implique en revanche une multiplication un peu plus compliquée. Si donc je vois mon adversaire transpirer à grosses gouttes et mettre du temps à me répondre, je peux en déduire qu'il est vraisemblablement en train de calculer combien font 46×37 et que j'ai plutôt intérêt à choisir le contenu du second pot.

Les attaques par canaux cachés se basent sur le même principe, à savoir que certaines opérations sont plus exigeantes et nécessitent donc davantage de ressources (en temps, en énergie,...). En observant la consommation des ressources, on déduit des informations sur les opérations effectuées

seule entité capable – avec ma collaboration via l'introduction du code PIN – de générer un ordre de paiement correct à destination de ma banque, les détails dudit paiement ne lui sont pourtant pas directement transmis par moi, mais transitent par le lecteur de cartes du commerçant : c'est le lecteur qui a envoyé la demande de paiement à la carte et c'est ce même lecteur qui se charge à présent d'afficher à mon intention le montant de la transaction. Qui me prouve que le lecteur n'a pas transmis à ma carte un montant dix fois supérieur à celui de mes achats, tout en continuant à m'afficher sagement le montant original ? C'est d'ailleurs encore par l'intermédiaire de ce même lecteur que je vais introduire mon code PIN et, dès l'instant où il le possède, rien ne l'empêche a priori de profiter de la présence de ma carte pour lui ordonner une nouvelle transaction, ou d'en garder copie en attendant qu'un complice ne vienne me dérober ma carte. On le voit, le pouvoir

HISTORIQUE

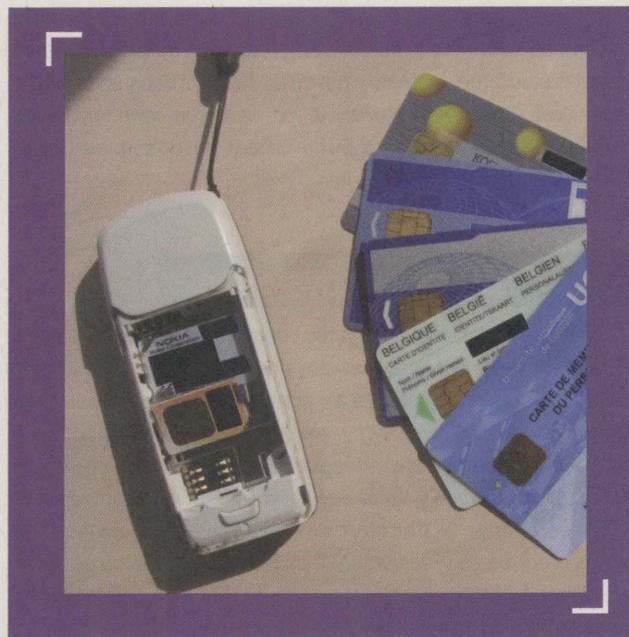
INTRODUCTION

d'un lecteur corrompu est tout simplement énorme et, même s'il n'y a pas lieu de sombrer dans la paranoïa – les lecteurs de cartes à puce doivent répondre à de très exigeants standards de sécurité – il est néanmoins bon de noter que des attaques de ce type se produisent réellement : le *skimming*, par exemple, est l'attaque consistant à modifier un terminal tel qu'un distributeur de billets afin de pouvoir capturer des informations lorsqu'il est ensuite utilisé.

La menace est bien entendu tout aussi réelle, voire pire, quand on considère le cas d'un lecteur de cartes à puce relié à un ordinateur personnel : comme nous l'avons dit plus haut, la carte et son contenu sont normalement à l'abri des virus et autres logiciels malveillants pouvant affecter l'ordinateur. En revanche, le pirate prenant contrôle des informations affichées à l'écran ou encodées au clavier peut à nouveau modifier à sa guise le contenu des transactions. Les attaques par hameçonnage (*phishing*) consistent par exemple à rediriger la communication vers un serveur frauduleux prenant l'apparence du serveur visé, un serveur de paiement en ligne, par exemple. Si cela permet l'exécution d'un programme piraté de communication avec la carte, bien des choses deviennent possibles pour l'attaquant.

Les choses sont encore plus marquées avec la carte sans contact : certes, celle-ci facilite grandement les transactions puisqu'elle n'a pas besoin d'être introduite dans un lecteur, voire sortie du portefeuille, pour pouvoir interagir avec le lecteur. Cependant, cette même absence d'action directe de la part de son détenteur implique qu'elle peut a priori être interrogée sans son consentement, et sans même qu'il n'en ait conscience.

On le voit, une importante fonctionnalité qui manque à la carte à puce est un canal de communication direct avec son utilisateur, permettant à ce dernier d'envoyer des ordres directs à la carte et d'être informé de ce qu'elle fait. Diverses initiatives sont prises en ce sens. Certaines banques fournissent par exemple à leurs utilisateurs de services bancaires en ligne un lecteur autonome de cartes à puce, doté d'un clavier et d'un écran, et réalisant donc une telle interface de confiance. La difficulté consiste alors à pouvoir utiliser cette interface souvent assez rudimentaire (l'écran n'affiche typiquement que quelques caractères et le clavier se



borne à un pinpad) pour afficher un résumé sans équivoque de ce qui se passe.

Nous n'avons soulevé là qu'une partie du problème et de très nombreux autres éléments entrent en jeu dans un système de sécurité basé sur la carte à puce : la manière dont les cartes sont émises, distribuées et détruites, la sécurité des serveurs et des bases de données avec lesquels la carte interagit, la définition des protocoles à suivre et leur respect sur le terrain, ne sont que quelques-uns des aspects entrant en ligne de compte.

Bien évidemment, tout ceci ne signifie pas qu'il faille jeter le bébé avec l'eau du bain : la carte à puce, de par sa polyvalence, son faible prix de revient, la facilité de son déploiement et sa sécurité intrinsèque, est un formidable outil permettant de satisfaire les attentes sécuritaires des plus exigeants. Simplement, et comme pour toute solution technologique, il faut bien l'envisager comme une simple brique de l'édifice et non comme une solution magique à tous les problèmes. ■



Références

- ⇒ RANKL (Wolfgang) et EFFING (Wolfgang), *Smart Card Handbook*, 3rd edition, 2003, John Wiley & Sons, ISBN: 0-470-85668-8.
- ⇒ BARJAVEL (René), *La Nuit des temps*, Ed. Denoël, 1968.
- ⇒ SAUVERON (Damien), « Les cartes à puce pour tous », Conférence proposée par l'IREM, le CIJM, le TML et l'APMEP, 30 janvier 2008 (présentation disponible à l'adresse http://www.unilim.fr/irem/uploads/media/CartesAPucePourTous2008_07.pdf).
- ⇒ Site web de la *Smart Card Alliance* : <http://www.smartcardalliance.org/pages/smart-cards-intro-primer>
- ⇒ U.S. General Service Administration, *Government Smart Card Handbook*, disponible à l'adresse <http://www.smart.gov/information/smartcardhandbook.pdf>

UNE HISTOIRE PERSONNELLE DE LA CARTE À PUCE JAVA

■ mots clés : *confiance / confidentialité / innovation*

Qui a jamais participé à un évènement historique ne peut qu'être surpris à le voir relaté dans des articles de journaux, des livres ou des émissions diverses. Il apparaît presque que leurs auteurs ont vécu une tout autre expérience, parfois et peut-être souvent même incompatible avec sa propre impression.

En 2005, j'ai eu l'honneur de recevoir le prestigieux trophée du visionnaire du magazine Card Technology, à l'époque la référence en matière de publications dans le monde des cartes à puce. Dans un long article, le journaliste Dan Balaban a fourni une histoire assez exacte de la carte à puce Java et de mon implication. Mais, naturellement, je ne pouvais pas à l'époque lui dire ce que je vais maintenant raconter, car il était trop tôt. Comme j'ai toujours été curieux de savoir comment l'Histoire sonne, écrite par ses auteurs, j'ai été très heureux quand il m'a été demandé d'écrire cet article.

⇒ 1. Les débuts

Tout commença pour moi en 1995. J'étais un jour dans mon bureau à Austin, Texas, où j'officialiais à Schlumberger comme responsable d'un département de développement de logiciels techniques pour l'exploration et production pétrolière. Surgit derrière moi mon patron de l'époque, Bill Preeg (tous les noms de ce récit sont authentiques, mais les mémoires sont miennes, et pourraient n'être pas corroborées ; comme nous l'avons dit, ainsi va l'Histoire). Il fut surpris de me voir en contemplation d'un petit hélicoptère vaguement tracé évoluant sans grâce sur l'écran de mon ordinateur. Je me souviens lui dire : « Voila qui va changer notre monde ; ceci est un applet Java, le premier exemple de logiciel amené du réseau à notre service immédiat ». J'aurais dû dire : « changer *mon* monde ».

À la fin de 1995, Olivier Piou (le futur président de Gemalto) était en charge du marketing et des produits de la division cartes à puce de Schlumberger, à Paris. Schlumberger avait acheté à la fin des années 70 les brevets de Roland Moréno, un des inventeurs de la carte à puce, et, au cours des années, la division cartes

à puce s'était établie, mais sans l'ambition ni les moyens que la compagnie avait dans d'autres domaines. Elle s'était laissée dépasser par une plus ambitieuse jeune société, Gemplus, qui avait réussi à se hisser au premier rang mondial, en manœuvrant habilement autour de la fourniture de cartes de téléphone de cabine publique, au début en France, puis à l'étranger. À l'époque de notre récit, certains dans l'industrie prédisaient même la fin de Schlumberger dans les cartes à puce. Mais Olivier Piou réussit à convaincre le président de Schlumberger à l'époque, Euan Baird, d'investir quelques millions de dollars dans un programme de trois ans de développement d'un interpréteur (nous reviendrons pour une explication un peu plus loin) pour cartes à puce et de logiciel associé permettant son utilisation dans les ordinateurs portables. Euan Baird ne mit qu'une condition à l'opération, qu'elle soit conduite à Austin, Texas, et non pas en France. C'est ainsi qu'après quelques pérégrinations, Olivier Piou et Bill Preeg me proposèrent un jour de prendre à mon compte ce programme dans un domaine que je ne connaissais absolument pas. Mais par contre, je connaissais très bien le fonctionnement de la

HISTORIQUE

INTRODUCTION

compagnie après presque vingt ans de maison dont quinze ans passés aux États-Unis. Je n'acceptais qu'à une seule condition : que mon patron pour la partie cartes à puce de mon activité (car je gardais mes autres fonctions) soit Olivier Piou et personne d'autre. Bien que ce ne fut pas dans les plans, ma condition fut acceptée. De fait, j'avais anticipé les difficultés considérables qui ne pouvaient que m'attendre à travailler loin des centres de développement français, et de plus en dépit de la volonté initiale et naturelle de ces centres d'entreprendre eux-mêmes ce programme qu'ils avaient imaginé. Dans toute société le pouvoir d'un patron est lié au pouvoir du niveau hiérarchique supérieur, et je savais ainsi que je pourrais avoir l'appui direct de Jacques Connefroy, le directeur général de la division cartes à puce ; voilà une puissance de frappe qui me satisfaisait. Le 1^{er} février 1996 donc, je démarrais sur une route qui me mena bien au-delà de ce que quiconque pouvait avoir imaginé.

Ma première initiative fut d'embaucher les ingénieurs que je pus trouver dans la compagnie. Sachant que j'avais le support du siège, j'essayai de recruter les cinq meilleurs pour la tâche à entreprendre. Je réussis à rassembler quatre d'entre eux : Ed Dolph, un ingénieur spécialisé dans les systèmes dont je connaissais le talent à deviner avant d'autres les écueils ; Scott Guthery, un innovateur bouillonnant avec qui j'avais partagé d'autres aventures, qui couvrirait la partie créative de l'opération ; Tim Jurgensen, calme et posé, un penseur profond qui veillerait à ce que nous réfléchissions avant de nous engager dans des développements d'importance (plus tard, Tim et moi écrivons ensemble *Computer Theology : Intelligent Design of the World Wide Web*, une vaste entreprise de description des phénomènes religieux et digitaux directement inspirée de notre expérience) ; et Mike Montgomery, qui, avec un nombre de brevets impressionnants et une grande expertise dans le domaine des semi-conducteurs, saurait couvrir la partie électronique ainsi que les questions de propriété intellectuelle de l'aventure. J'appliquai ensuite pour mes autres recrues un principe de base reposant sur trois questions : 1) Êtes-vous qualifié(e) techniquement ? 2) Êtes-vous qualifié(e) politiquement ? 3) Avez-vous déjà été dans la mouise (j'utilisais un mot plus fort) ? La deuxième question me permettait de m'assurer que je pourrais m'occuper de mes propres problèmes plus que des problèmes de mon équipe, et la troisième me donnait l'assurance que les obstacles ne ralentiraient pas notre marche.

C'est ainsi que j'engageai Ksheerabdi Krishna, un spécialiste des compilateurs des ordinateurs de grande puissance. Et Marc Valderrama, un ingénieur touche-à-tout que je ne connaissais en fait que pour jouer au football à midi avec lui une fois par semaine ; à voir son style de jeu mêlant l'individualité et le sens de l'équipe, je pensais qu'il conviendrait à la tâche, et l'avenir me donnerait raison. Ly Thanh Phan vint de France et nous introduisit à la technologie des cartes à puce tout en servant de liaison avec les équipes françaises. Et comme j'avais besoin d'un spécialiste du marketing, je scrutais les horizons jusqu'au jour où un docteur en physique nucléaire, Merzad Mahdavi, débarqua dans mon bureau en me demandant si je le voulais pour ce poste. Je lui demandai quelle était son expérience en marketing ; il me dit « aucune ». Je lui dis « vous êtes embauché ». Naturellement, le président pour les États-Unis de la division cartes à puce,

Bob Davis, sur lequel les opérations de marketing reposaient, demanda à avoir une entrevue avec cet individu sans pedigree. Je ne sais comment, Merzad passa l'examen, mais, comme nous allions voir, ce n'était qu'un début dans les acrobaties. D'autres joignirent l'équipe et jouèrent des rôles importants, mais j'ai choisi de ne mentionner que ceux qui apparaissent dans les épisodes que je décris dans cet article.

La première décision que nous prîmes fut d'utiliser pour notre interpréteur de carte à puce le langage Java. Pour moi, ce choix était clair de par mon expérience, mais il se trouva que Scott Guthery avait été commandité quelques mois auparavant par la même division de cartes à puce pour faire une étude des choix possibles pour un interpréteur. Donc, pas de surprise si Scott était préparé, puisqu'il avait inclus Java dans les possibilités. Cependant, nous avons initialement décidé de n'utiliser Java que comme moteur de la carte à puce, et non pas comme langage de programmation. En effet, pour le développement de programmes pour la carte, nous avons tout d'abord opté pour Basic, un langage favori de Microsoft, la raison étant que nous n'étions pas sûrs à l'époque que Java aurait un succès durable. Puisque Basic était bien reconnu, il nous semblait que notre stratégie, qui consistait à écrire du logiciel dans ce langage, puis à le traduire pour le mettre en carte dans la forme nécessaire pour son moteur Java, serait optimale. Ceci établi, nous fîmes en mars un voyage à Paris pour présenter l'équipe. Jacques Connefroy nous invita à dîner dans un restaurant à Montrouge, et nous posa la question : « Quand l'interpréteur sera-t-il prêt ? » À sa grande surprise, je répondis intuitivement « Dans neuf mois ». Je n'avais aucune idée de comment nous ferions ça, mais je savais qu'à Schlumberger les programmes de trois ans ont beaucoup plus de chance d'être soutenus s'ils donnent rapidement des résultats sensibles. Pendant ce voyage, nous eûmes le plaisir de rencontrer Vincent Rigal, le chef du développement de la division, qui nous introduisit à notre nouveau domaine et à ses participants. Je retirai de ses présentations que nous n'aurions pas de support immédiat de toute son organisation, chez qui certains nous verraient naturellement plus comme compétiteurs extra-terrestres que comme membres de l'équipe : conséquemment, j'enregistrai mentalement qu'il s'agirait de veiller soigneusement aux relations internes. Quelque peu après notre retour aux États-Unis, j'interrogeai Ed Dolph, qui avait commencé à travailler sur la partie Basic du projet, sur son progrès. Après cette discussion, je conclus que nous n'avions aucune chance de finir cette partie de l'effort en neuf mois. Je vins dans le couloir avec ma décision en tête et rencontrai Scott s'appêtant à entrer dans les toilettes. Alors qu'il avait la main sur la porte, je lui proposais d'éliminer Basic et de tout miser sur Java. Il dit « d'accord », je partis content, et il put procéder aux affaires urgentes.

Ici, je dois expliquer qu'un interpréteur, dans la vie courante, est naturellement quelqu'un qui traduit, disons de l'anglais au français. C'est la même chose dans les ordinateurs. Je vais simplifier un peu dans ce qui suit, mais les experts me pardonneront en effectuant les corrections nécessaires. Un interpréteur traduit le langage Java dans un autre langage que la carte à puce comprend. Vous me direz, pourquoi ne pas parler directement dans ce dernier langage ? De fait, c'est ce qui était fait auparavant (et même encore aujourd'hui dans certains projets). Mais, en utilisant un

interpréteur Java, nous avons deux avantages : le premier est que les ingénieurs informatiques sont en grand nombre à connaître Java, et peu à connaître les cartes à puce ; le second est qu'en utilisant un interpréteur, nous contrôlons ce qui entre dans la carte à puce. Si l'interpréteur pense que le code Java qui lui est proposé présente des risques de sécurité ou autres, il refuse simplement de traduire le programme et de l'entrer dans la carte (la réalité est un peu plus compliquée, mais nous nous en tiendrons là). Pour finir, je dois indiquer que le langage que la carte à puce comprend, que nous appellerons ici le système d'exploitation pour faire simple, est dépendant de l'électronique de la carte à puce. Quand l'électronique change, le langage change. Il s'agit donc d'un autre avantage de l'interpréteur : il prend en charge les particularités du système d'exploitation. L'ingénieur qui écrit en Java n'a pas à s'en soucier.

Maintenant que nous avons les bases en place, je peux indiquer que Scott me proposa trois contrats extérieurs allant jusqu'à 100000 dollars chacun. L'un pour développer un traducteur automatique de programme qui permettrait de passer automatiquement d'un système d'exploitation à un autre pour changer aisément l'électronique sous l'interpréteur. Le deuxième pour évaluer automatiquement la sécurité d'un système d'exploitation combiné avec Java. Le troisième pour demander à Tim Wilkinson, un ingénieur anglais qui avait développé un système Java en compétition avec celui de l'inventeur de Java, Sun Microsystems, d'évaluer comment on pourrait réduire Java pour le mettre dans une carte à puce : il vous faut savoir, qu'à l'époque, Java demandait 80 mégaoctets de mémoire. Nous n'avions que 4 kilo-octets disponibles sur notre carte à puce, soit 20000 fois moins ! Je pensais immédiatement que le premier contrat ne marcherait pas, que le deuxième était douteux, mais que le troisième était intéressant. Mais comme j'avais comme principe que l'on ne doit jamais freiner la créativité, je proposai à Bill Preeg les trois programmes : il les accepta tous, établissant une conduite qui fut de toujours supporter ce que nous faisons, même dans les pires moments ; sans Bill, la carte Java n'aurait pas existé. Des trois programmes, les deux premiers échouèrent, mais le troisième fut un succès sans équivoque et au crédit de Scott. Tim Wilkinson proposa un sous-ensemble des instructions Java qui se révéla être presque entièrement final pour une carte à puce à mémoire très réduite. Par la suite, Ksheerabdi Krishna prendrait en charge l'interpréteur, avec une excellente relation personnelle avec Tim Wilkinson qui permit au projet de progresser très rapidement. Quand à l'idée de traduction automatique de système d'exploitation, Marc Valderrama, qui avait anticipé son échec, décida d'écrire un système d'exploitation adapté à supporter Java ; et, ce faisant, il évita un écueil qui était le résultat du progrès initial lent des relations internes que nous avons mentionnées. Au lieu d'utiliser la puce Texas Instruments expérimentale qui nous avait été proposée pour supporter l'interpréteur, il me dit que nous ferions mieux d'utiliser une puce établie, et il choisit pour cela un composant Motorola qui était déjà installé dans d'autres cartes à puce. Il avait entièrement raison, et sa décision fut un facteur critique de notre succès.

...Bertrand vous aime, mais il n'est pas heureux...

Assez pour la technique. Nous avons maintenant (en avril 1996) un grand problème. Java avait été inventé par Sun, qui n'avait aucune intention de le laisser disponible à tout un chacun. Pire, Sun avait publiquement annoncé qu'ils ne laisseraient personne réduire Java. Donc, nous avions tout d'abord à convaincre Sun de nous laisser opérer une réduction, et ensuite de nous accorder une licence. Merzad Mahdavi visita Sun, et fut renvoyé dès qu'il mentionna ce que nous voulions faire. Nous étions dans une impasse. Mais Merzad était le fils d'un général iranien dont la famille a été forcée à immigrer d'Iran lors de la perte de pouvoir du Shah, s'établissant à Houston. Il connaissait les difficultés, et n'était pas prêt à accepter « non » comme réponse. Il était sorti par la porte, il entrerait par la fenêtre, ce qu'il fit en contactant non plus les services de développement de Sun, mais leurs services de vente. Il finit par convaincre un dirigeant haut placé de la valeur financière qui proviendrait de notre travail. Et ce fut cette personne qui nous permit d'avoir une seconde réunion avec Sun, cette fois avec d'une part une équipe d'affaire, et, d'autre part, une équipe technique qui incluait James Gosling, l'inventeur de Java. Nous avions notre chance : nous expliquâmes à James ce que nous voulions faire, et il acquiesça à la seule condition de remplacer une des instructions Java que nous avions proposée pour la forme réduite par une autre. Cela fait, le groupe d'affaire était prêt à négocier une licence, avec l'idée que nous ferions une annonce publique le 29 octobre 1996. Sun s'engageait à contacter nos concurrents pour les engager dans la même voie, et ainsi nous verrions toute une industrie prendre une nouvelle direction. Dans mon esprit, le changement proposé était si important que les désavantages

à aider nos concurrents à progresser technologiquement seraient compensés par l'impact qu'aurait sur l'industrie l'utilisation d'un interpréteur Java standard, propulsant la carte à puce dans le monde de l'informatique traditionnelle.

Naturellement, cela ouvrait un troisième front, notre propre organisation, chez qui certains pensaient que notre interpréteur devait être propriétaire, et non pas écrit sous des normes partagées avec nos concurrents. Heureusement, grâce au soutien de Jacques Cosnefroy qui pouvait balayer les résistances internes, nous fûmes prêts à signer le contrat avant l'annonce du 29 octobre. Je passerai les péripéties associées, pour deux raisons : la première est que Sun demandait que la licence fut confidentielle, et la deuxième parce que ces péripéties sont communes à tout contrat important. Cependant, je dois mentionner un point central à notre histoire. L'avocat de notre groupe, Danita Maseles, me suggéra un matin de l'été 1996 de prendre un brevet sur notre invention de réduction de Java pour une carte à puce. Mike Montgomery pourrait entrer en action, et Danita nous avait rendu un grand service, car, comme nous allons le voir, le brevet nous servirait bien dans le futur. Le 29 octobre donc, une annonce officielle de la carte Java fut faite conjointement par Sun et la plupart des grands de la carte à puce, y compris Gemplus. Simultanément, Schlumberger annonça sa nouvelle carte Java, Cyberflex, pour 1997. L'industrie fut prise d'un énorme bouleversement qui changea toutes les perspectives.

2. L'adolescence

Le 2 février 1997, nous reçûmes de l'usine la première carte Java, qui est toujours en ma possession, signée par les membres de l'équipe. Je la lèguerai au musée qui me la demandera s'il s'engage à lui donner la présence qu'elle mérite. Miraculeusement, et à notre grande surprise, elle fonctionna immédiatement. Pour mesurer la rareté de l'évènement, il faut savoir qu'il nous fallut ensuite un an pour faire la seconde, avec deux échecs entre-temps : les cartes à puce sont de très délicats mécanismes à cause des mesures de sécurité qu'elles comprennent. Une erreur, qui peut venir de plusieurs horizons, et la carte décide de ne pas fonctionner. Donc, réussir du premier coup était étonnant. Nous décidâmes immédiatement de la mettre en vente avec un système de développement pour familiariser la communauté informatique avec la technologie. Pendant ce temps, Olivier Piou avait engagé les procédures d'industrialisation et avait commencé une campagne d'information tout en proposant avec sa conviction habituelle la nouvelle carte aux plus grands utilisateurs de cartes à puce, les opérateurs de téléphonie mobile. La première vente en volume fut de 500000 cartes à l'opérateur Swisscom, ce qui valida la proposition commerciale.

Cependant, les nuages s'accumulaient. Gemplus n'avait aucune intention d'abandonner sa première place. Nous avions réussi à les entraîner dans une aventure dont je crois ils se seraient peut-être passés ; ils allaient réagir. D'autre part, les équipes de Sun, émulsionnées par un succès que nous leur avions apporté sur un plateau d'argent, commençaient à annoncer à qui voulait les entendre qu'ils seraient à l'industrie de la carte à puce ce que Microsoft était à l'industrie des ordinateurs : une force dominante qui nous aurait réduits à néant. Il faut savoir que les fabricants d'ordinateurs peuvent fournir une valeur ajoutée grâce aux nombreuses options possibles. Par contre, sur les cartes à puce, c'est principalement la combinaison de matériel et de logiciel sécurisés qui fait la proposition du fabricant, lui-même dépendant des fournisseurs de puce électronique, et se présentant comme fournisseur des grands distributeurs. Si Sun comblait son désir, l'industrie des fabricants était en danger de perdre son rôle et d'être réduite à des fonctions subalternes. Par conséquent, je téléphonai à Vincent Rigal pour lui demander s'il savait comment organiser un consortium de tous les fabricants de cartes à puce qui prendrait en main la définition des cartes Java pour former un contrepoint à Sun. Naturellement, je pensais aussi qu'en travaillant sur le sujet avec Gemplus, j'aurais plus de chance de comprendre leur stratégie de contre-attaque. Je suis sûr que Gemplus eut la même pensée et suite à l'annonce de la création du Java Card Forum le 12 février 1997, nous eûmes début mai 1997 à Bruxelles la première réunion du Java Card Forum, avec en particulier Bull, Gemplus, Schlumberger et Sun en présence.

Je ne connaissais pas Philippe Maes, un fondateur de Gemplus habitué des télévisions françaises, ni Christian Goire, un vétéran de la carte chez Bull, ni la trentaine d'autres présents à part naturellement le contingent Schlumberger, qui incluait Olivier Piou. En ce qui me concerne, le moment qui m'amusa le plus est quand nous fîmes un tour de table où chacun devait exprimer ses priorités. Je proposai

« Amour et bonheur ». Plus tard, Arthur Coleman, de Sun, essaya de nous expliquer que nous faisons face à un « changement de paradigme ». Je lui demandai ce que c'était, et il répondit « Et bien c'est simple ; par exemple, avant, on pensait que tout tournait autour de la terre ; le changement de paradigme, c'est quand on découvre que tout tourne autour du soleil ». Comme naturellement, « Sun » veut dire « Soleil » en anglais, la salle s'indigna, et moi en particulier ; et là, Philippe Maes eut ce mot superbe : « En résumé, Bertrand vous aime, mais il n'est pas heureux ». La glace était brisée. Le lendemain, Christian Goire prenait la présidence du Java Card Forum, Michel Roux, un dirigeant de Gemplus en charge de leur centre de Californie dans la Silicon Valley prit la direction du groupe affaires du forum, et je pris la direction du groupe technique.

La première action de Gemplus fut de déclarer que les spécifications originales de Schlumberger pour Java qu'avait adoptées Sun devaient être remplacées par une nouvelle version, qu'ils appellèrent version 2. Avec la puissance de leur service de communication, ils réussirent à faire que le Java Card Forum passerait un an en réunions mensuelles à travers le monde pour réécrire ce que nous avions défini, leur donnant le temps d'effectuer leurs propres développements. Grâce à Christian et à Michel du côté affaires, le Java Card Forum devint pendant cette année la grande voix de l'industrie en matière de programmation et de système d'exploitation de cartes à puce, avec en particulier une énorme réunion à Paris de tous les grands participants de l'industrie de la carte dans la finance, la téléphonie et l'informatique. À Austin, nous reçûmes en retour des trains de visiteurs du monde entier. J'étais en particulier étonné de voir que de nombreuses sociétés envoyaient des délégations dirigées par de très importants représentants de leur compagnie. Pendant que Christian et Michel travaillaient de leur côté, il me fallait trouver avec mon comité ce qu'était cette version 2 qui satisferait tout le monde, étant donné que j'avais établi une règle que toutes les propositions techniques devraient être adoptées à l'unanimité, par consensus et non pas par vote. Comme il était clair que les ingénieurs des diverses compagnies n'étaient pas encore prêts à collaborer, soit par ordre de leurs supérieurs, soit simplement à cause de la tradition de secret de l'industrie, il me fallait trouver un moyen d'asseoir tout ce monde à la même table. Je choisis de nous trouver un ennemi commun : Microsoft.

Il était facile de faire de Microsoft un vilain, pour les raisons expliquées plus haut. Si quelqu'un prenait le contrôle des logiciels de cartes à puce indépendamment des fabricants, ceux-ci étaient réduits à néant. Or, les dirigeants de Microsoft, qui voyaient Sun leur prendre avec Java leur rôle traditionnel de développeur de système d'exploitation, craignaient que, s'ils les laissaient faire, cela pourrait créer un précédent, et Java pourrait étendre son influence dans le bastion sacré de Microsoft, les ordinateurs portables et autres. Nous savions cela parce que le chef de la stratégie de Microsoft dans le domaine de Java, Charles Fitzgerald, nous avait visité à Austin. De fait, Microsoft décida plus tard de créer un programme « Carte à puce pour Windows ». La menace était réelle, et, en fait, il se révélera que Microsoft embaucherait rapidement notre Scott Guthery comme architecte principal du projet. Il m'était ainsi

facile de créer un ennemi commun pour mon comité technique, et ils se mirent au travail. Naturellement, j'avais de la diplomatie à faire, parce que je devinais (je n'ai jamais eu de confirmation définitive à ce sujet) que Michel Roux était le négociateur de Gemplus avec Microsoft ; je soupçonnais qu'un plan de Gemplus pour diminuer notre influence avec Java incluait une alliance privilégiée avec eux. Quoi qu'il en soit, après un an la version 2 de la carte Java était bien avancée, et Sun mit toute son énergie de communication à la promouvoir : il faut savoir qu'en 2007 le cap des cinq milliards de cartes Java de version 2 était dépassé,

en faisant l'ordinateur le plus vendu au monde. Pour ma part, j'en retirai une nomination au poste prestigieux de Schlumberger Fellow, la plus haute distinction que la compagnie confère à ses ingénieurs. Les membres de mon équipe en retirèrent également, je l'espère, des bénéfices importants ; en particulier, Scott Guthery et Tim Jurgensen écrivirent plusieurs livres. Ce fut une équipe applaudie officiellement par plusieurs médailles et trophées, et je sais que chacun gardera de cette aventure le sentiment d'avoir eu la chance d'une vie d'avoir participé à l'histoire de l'informatique.

⇒ 3. La maturité

Un changement aussi important que l'introduction de Java dans la carte à puce ne pouvait pas se passer sans créer d'énormes tensions dans l'industrie. À l'intérieur de Schlumberger, il fallait changer les plans de vente et de marketing, il fallait rediriger les efforts de développement, et il fallait supporter le nouvel essor de la compagnie qui soudain se retrouvait dans le peloton de tête des fabricants. Nous savons maintenant que, dans les autres sociétés, les nouvelles orientations étaient au moins aussi difficiles. Dans certaines, comme Bull, la résistance à Java était quasi-totale ; Bull en finit par être vendue à Schlumberger. Gemplus avait créé en Californie une société de développement dirigée par Patrice Peyret qui avait développé son propre interpréteur, et qui se trouva soudainement en compétition avec Java ; la société fut finalement achetée par Sun pour son expertise en cartes, faisant au passage de Patrice un homme riche. Des fabricants de puces essayèrent de bénéficier des changements en cours pour étendre leurs affaires au-delà des composants électroniques, vers la fabrication même de cartes à puce, en directe compétition avec nous, leurs clients. Motorola, en particulier, établit publiquement un programme de 100 millions de dollars à cet effet, mentionnant Schlumberger nominalement comme concurrent. Cet argent fut perdu quand l'effort fut totalement abandonné. Des compagnies électroniques japonaises assistèrent assidûment au Java Card Forum et travaillèrent avec Sun pour s'établir comme fabricants de cartes Java ; leurs efforts furent limités, tant était leur retard. En Chine, un effet similaire fut observé. La technologie des cartes Java était devenue extrêmement sophistiquée, et peu étaient prêts à investir aux niveaux requis. Finalement, la division cartes de Schlumberger devint une compagnie indépendante sous le nom d'Axalto, et plus tard Axalto et Gemplus fusionnèrent en Gemalto. À ce point, je quittai l'industrie pour me consacrer à nouveau pleinement à l'informatique pétrolière, tout en écrivant *Computer Theology* avec Tim Jurgensen comme ode à une industrie que j'avais aimée et qui me le rendit bien.

La carte à puce, grâce à sa portabilité, sa puissance de connexion aux réseaux, sa sécurité intrinsèque et sa mémoire protégée qui permet de lui confier nos plus profonds secrets, nous sert de représentant dans le monde digital. Qu'elle soit dans notre téléphone portable sous forme de SIM, dans notre portefeuille sous forme de carte de crédit, en pendentif sous forme de badge d'entreprise ou dans notre main à l'entrée du

métro, elle dit au réseau informatique : « Je représente mon propriétaire, qui a confiance en moi, et vous pouvez aussi avoir confiance en moi et lui donner accès à vos ressources ». Dans ce rôle, elle permet aux ordinateurs de créer des réseaux de confiance alors qu'ils s'assemblent en communautés de plus en plus importantes. Ceci nous rappelle l'expansion qui commença il y a quelques 50000 ans, où les groupes humains apprirent à établir suffisamment de confiance réciproque pour s'assembler plutôt que de s'annihiler et pour construire ensemble les défenses et bénéfices que la nature requiert et présente. Comme on sait que tous ces groupes humains ont à ce stade établi des religions, il n'y avait qu'un pas à franchir pour étudier le rôle possible de la théologie dans les ordinateurs. De par mon rôle dans le développement de la carte Java, j'eus l'honneur d'être invité à présenter cette nouvelle approche des réseaux informatiques et de la religion à l'*Association for the Advancement of Artificial Intelligence*. Le gotha de l'intelligence artificielle fut ainsi introduit à la carte à puce, et je présentai ensuite mon histoire dans les universités du monde entier, ce qui conduisit à la publication de mon livre. De Java à une meilleure compréhension des ordinateurs par le monde religieux et du monde religieux par la communauté scientifique, peut-être que cette aventure industrielle et intellectuelle aura-t-elle conduit à un plus paisible monde.

Et Microsoft dans tout cela ? Eh bien, après quelques années d'essai, ils déduisirent de notre brevet et d'autres que l'industrie de la carte à puce maintiendrait son indépendance envers Sun, et qu'ils n'avaient pas besoin de développer un système d'exploitation différent dans un domaine qui ne leur était pas central ; ils préférèrent travailler de concert avec les fabricants de cartes à puce pour les installer au mieux dans Windows. Un nouvel ordinateur était né.

Bertrand du Castel est Schlumberger Fellow et vit à Austin, Texas. Directeur de la recherche de la division cartes à puce de Schlumberger, puis d'Axalto (maintenant Gemalto) de 1996 à 2006, il obtint en 2005 le trophée de visionnaire de l'année de Card Technology Magazine pour la carte à puce Java. Bertrand est auteur avec Tim Jurgensen de Computer Theology : Intelligent Design of the World Wide Web (Midori Press, 2008, ISBN : 978-0-9801821-1-8). Bertrand est diplômé de l'école polytechnique et docteur en informatique théorique de l'université de Paris 7. ■

LE MARCHÉ DES CARTES À PUCE

mots clés : SIM / carte à microcontrôleur / télécarte / Java Card / NFC / Machine-to-Machine (M2M) / Over-The-Air (OTA) / smart security / smart objects

1. Le cap des 7 milliards d'unités devrait être franchi en 2009

Plus de 4 milliards de cartes à puce ont été livrées dans le monde en 2007, pas loin de 5 milliards devraient l'être en 2008. Selon la société d'études IMS Research, le cap des 7 milliards de cartes utilisées sur « le terrain » devrait être franchi à la fin 2009, dépassant ainsi avec deux ans d'avance les prévisions démographiques du nombre d'habitants de la planète (7 milliards en 2011, selon l'ONU). La prévision faite au début des années 90 – à chaque habitant de la planète sa carte à puce –, par Marc Lassus le fondateur de Gemplus (l'un des premiers grands acteurs de ce marché, devenu aujourd'hui Gemalto, après sa fusion il y a

trois ans avec le numéro 2 du secteur, Axalto) est donc en passe de se réaliser. Depuis ses modestes origines franco-françaises, la carte à puce a en effet connu un mouvement continu d'adoption, d'internationalisation et de standardisation pour des applications de plus en plus nombreuses. La France, qui fut longtemps son premier marché mondial avec ses quelques 30 ou 40 millions de cartes bancaires, a laissé depuis déjà une dizaine d'années la place à la Chine qui compte désormais largement plus d'un milliard et demi de cartes à puce « sur le terrain », dont près d'un milliard de cartes d'identité sans contact, et plus de 500 millions de cartes SIM.

L'évolution des marchés des cartes à microcontrôleur (en millions d'unités)

	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008
Telecom (SIM)	200	370	390	430	670	1050	1390	1800	2650	3000
Bancaire	108	120	140	175	205	280	336	410	510	580
Identité Gouvernement -Santé	30	20	16	32	40	45	60	90	105	150
Transport	3	3	8	15	12	15	20	25	30	30
TV à péage	29	20	25	35	35	55	55	60	85	80
Identité Entreprise	⇨ (avec TV à péage)	5	5	7	7	12	15	15	⇨ (avec autres)	20
Autres	28	3	15	7	10	12	12	15	65	15
Total cartes à microcontrôleur	398	541	599	701	979	1469	1888	2415	3445	3875

Estimation nov. 2007 pour 2008

Source : Eurosmart

2. La téléphonie mobile tire le marché

Trois évolutions importantes ont accompagné cette croissance mondiale « explosive » et favorisent aujourd'hui la voie vers de futurs développements, et sans doute l'avènement d'un nouveau cycle d'innovations dont le maître-mot serait sans doute moins celui de carte à puce (SmartCard) que de « Smart Objects » ou de « Smart Security ». La première évolution concerne les applications. Contrairement à une idée assez largement reçue qui fait de la carte de crédit (en réalité souvent une carte de débit) la figure emblématique de la carte à puce, les applications

de paiement de la carte n'ont toujours représenté qu'une part relativement modeste de ce marché : entre 15% et 20%. L'application qui n'a cessé depuis le début de tirer fortement la croissance du marché en lui donnant son caractère explosif est liée à la téléphonie. La télécarte (carte à mémoire prépayée), puis la carte SIM (carte à microcontrôleur gérant l'accès au réseau et les services d'un abonné) ont toujours représenté environ entre 70% et 78% du marché global de la carte à puce, avec une croissance moyenne annuelle de l'ordre de 30%.

3. Les cartes à microcontrôleur deviennent de petits PC

La deuxième évolution est celle qui a vu petit à petit l'utilisation des cartes à microcontrôleur prendre le pas sur celles des cartes à mémoire. La vitesse et l'ampleur de la migration de la télécarte vers la carte SIM illustrent cette évolution qui a été évidemment tirée par l'adoption de la téléphonie mobile, et du GSM en particulier, qui compte aujourd'hui près de 3 milliards d'abonnés dans le monde. En 1999, les livraisons de cartes à microcontrôleur représentaient 28% du marché de la carte, soit 398 millions de pièces. L'an dernier, sur les 4,455 milliards de cartes livrées, ces cartes « intelligentes » ont représenté plus de 77% (3,445 milliards) de ce marché. Cette évolution, qui en moins

de dix ans a multiplié le nombre de cartes à microcontrôleur par 10, est cruciale. Elle a tracé la route à de nouveaux développements de la carte qui peuvent désormais exploiter l'accroissement des capacités de traitement des nouvelles générations de microcontrôleurs (32 bits), celles des mémoires embarquées de plus en plus importantes (plusieurs Go de mémoire flash) et des interfaces et des protocoles de communication plus rapides et ouverts. L'USB Interchip est ainsi devenu un standard depuis deux ans. La dernière spécification de Java Card (Java Card 3.0) définit une version de Java proche de celle qui tourne sur des PC de bureau.

4. L'émergence du sans contact et la convergence des applications

La troisième évolution majeure – plus récente celle-ci – que connaît le marché de la carte à puce est liée à l'émergence des technologies et des applications sans contact. Pour la première fois, Eurosmart, l'association professionnelle qui regroupe près de 80% des industriels de la carte à puce dans le monde, a publié l'an dernier des chiffres de marché qui comptabilisent ce type de cartes et d'applications sous une rubrique à part. Avec 630 millions de cartes sans contact livrées en 2007 (sans compter les applications de contrôle d'accès physique et les tickets électroniques de transport), principalement pour des applications de transport (carte d'abonnement), de carte d'identité (en Chine), et de paiement (surtout aux États-Unis), le marché des cartes sans contact est encore modeste. Il est néanmoins promis à un bel avenir. Il est tiré en particulier par un mouvement de convergence qui se dessine grâce aux perspectives de nouveaux services offertes par l'implémentation de la technologie NFC (*Near Field Communication*) dans les téléphones mobiles, et

la gestion de ces services (paiement, contrôle d'accès) par une carte SIM devenue aussi sûre qu'une carte bancaire et aussi puissante qu'un petit micro-ordinateur. Mais il est aussi tiré par la montée en puissance du marché de l'identité (applications gouvernementales et entreprises), et des titres de voyages biométriques. Ces derniers ne totalisent aujourd'hui qu'un petit 3% ou 4% du marché global de la carte, mais nettement plus en valeur. Ce marché apparaît de plus en plus comme un futur relais de croissance et surtout comme une opportunité pour l'industrie de la carte, récemment rebaptisée industrie de la « Smart Security » de se développer dans le domaine des services. Ce type d'opportunité se présente d'ailleurs aussi dans la téléphonie mobile avec l'importance prise par les plates-formes de services OTA (*Over-The-Air*), qui permettent de gérer à distance les cartes SIM et leurs applications. Ces services pèsent déjà fortement dans la croissance du marché de la téléphonie mobile que l'on comptabilise à tort encore en nombre de cartes SIM livrées.

5. Vers les « Smart Objects » et une industrie de services

Les prochaines années seront certainement marquées par une accélération de cette migration vers des activités de services appelant une nouvelle segmentation du marché, mais aussi par l'explosion du nombre d'objets intelligents communicants sécurisés (*Smart Objects*) dont le téléphone mobile (Smart Phone, PC ultra-portable ou système M2M) figure aujourd'hui un peu le prototype. En juin dernier, lors du *Technology Forum America*, Lisa Lu (la responsable de la technologie de Freescale) indiquait qu'à l'horizon 2015 chaque individu serait l'utilisateur d'environ un millier de systèmes embarqués plus ou moins personnels. Fin août, à l'occasion de l'*Intel Developer Forum*, Jan M. Rabaey, Professeur au *Department of Electrical*

Engineering and Computer Sciences à l'University de Californie à Berkeley, et spécialiste des technologies sans fil, a avancé également ce chiffre, en évoquant plus spécifiquement le nombre des équipements dotés de fonctions sans fil utilisés par chaque personne. Eurosmart a déjà identifié au moins deux nouveaux vecteurs de développements d'objets « intelligents » sécurisés : celui des équipements *Machine-to-Machine* (M2M) dotés d'une carte SIM spécifique, et celui des clés USB équipées de microcontrôleurs de cartes à puce, et capables d'embarquer des capacités importantes de mémoire flash sécurisée, voire des capteurs biométriques. ■

YESCARD ENTRE MYTHE ET RÉALITÉ OU UN APERÇU DU SYSTÈME BANCAIRE FRANÇAIS

■ mots clés : *carte à puce / carte bancaire / protocole de paiement / B0' (B0 prime) / EMV / YesCard / Serge Humpich*

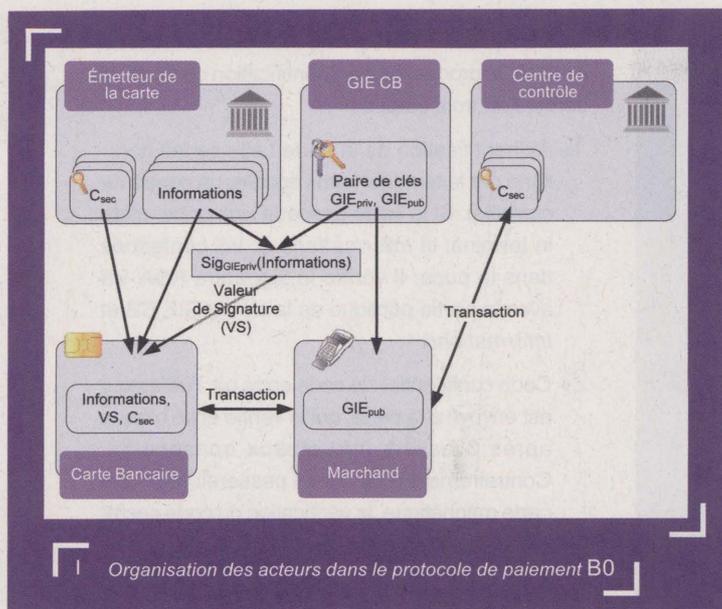
L'idée de paiement sans argent liquide n'est pas récente : « Chaque fois qu'un Gonda désirait quelque chose de nouveau, des vêtements, des objets, il payait avec sa clé. Il pliait le majeure, enfonçait sa clé dans un emplacement prévu à cet effet et son compte, à l'ordinateur central, était aussitôt diminué de la valeur de la marchandise ou du service demandé. » [1] Depuis plus de vingt ans, la carte à puce est au cœur du système bancaire. Elle fait l'objet de toutes les attentions de la part des banques et des utilisateurs, mais aussi, voire encore plus, de la part des pirates. La carte à puce en tant que coffre-fort électronique impénétrable était censée apporter le niveau de sécurité élevé requis pour les applications bancaires. Mais il semble que deux lois de base aient été oubliées par les banques, à savoir que le niveau de sécurité d'un système est caractérisé par son maillon le plus faible et qu'un système sécurisé ne résiste pas éternellement aux attaques. Nous allons tenter dans cet article de décrire le rôle de la carte à puce dans le système bancaire français. Nous détaillerons d'abord le premier protocole qui était encore en vigueur il y a quelques années. Nous verrons ensuite les failles de sécurité que ce protocole présentait et comment elles ont été exploitées par les pirates avec notamment les « YesCard ». Enfin, nous étudierons les principales évolutions de sécurité liées en particulier à ces attaques.

⇒ 1. Mécanisme de paiement par carte

Le protocole qui a été mis en place par le GIE CB pour le paiement par carte bancaire est aussi appelé *B0'*. Il fait intervenir différents acteurs (Figure 1, page suivante) :

- ⇒ la banque du client, noté *Émetteur de la carte* ;
- ⇒ le client, noté *Carte Bancaire* ;

- ⇒ le Terminal de Paiement Électronique (TPE) ou le Distributeur Automatique de Billets (DAB), noté *Marchand* ;
- ⇒ le centre de contrôle, noté *Centre de contrôle* ;
- ⇒ Le GIE cartes bancaires, noté *GIE CB*.



Organisation des acteurs dans le protocole de paiement B0

Une carte à puce a un cycle de vie bien défini :

- 1► Fabrication : inscription d'un programme (ici, un programme de paiement) en mémoire ROM. Ce programme définit les fonctionnalités de base de la carte. Ces fonctionnalités sont les mêmes pour toutes les cartes de cette famille.
- 2► Initialisation : inscription en EEPROM des données communes à l'application.
- 3► Personnalisation : inscription en EEPROM des données relatives à chaque porteur.
- 4► Utilisation : ici, paiement de biens avec la carte.
- 5► Mort : expiration, invalidation logique, saturation de la mémoire, perte, vol, etc.

Nous ne décrivons pas toutes les étapes du cycle de la vie de la carte, mais uniquement les phases 3 et 4.

À la personnalisation d'une carte, des informations spécifiques sont écrites dans la puce (Figure 1) :

- ⇒ Le nom du porteur, le numéro de la carte ou encore la date limite de validité de celle-ci (notés *Informations*).
- ⇒ La valeur de signature (noté VS), signature RSA d'*Informations* générée avec la partie privée de la clé du GIE CB ($VS = \text{Sig}_{\text{GIE}_{\text{priv}}}(\text{Information})$).
- ⇒ Le code confidentiel transmis au porteur de cette carte (aussi appelé code PIN). Ce code non modifiable est stocké sous forme chiffrée à la fois dans la puce et sur la piste magnétique de la carte.
- ⇒ Une clé secrète (noté C_{sec}), unique par carte, protégée en lecture et en écriture pour du chiffrement symétrique.

Le numéro inscrit sur la carte n'est pas une succession aléatoire de chiffres. Chacun de ces chiffres a une fonction particulière et peut donner de nombreux renseignements complémentaires

⇒ Trois types fondamentaux de cartes à puce

- 1► **cartes à mémoire** : elles ne disposent pas de capacité de traitement et ne permettent que de stocker des informations, comme les cartes à piste magnétique. Toutefois, elles possèdent plus de mémoire. Elles sont utilisées pour des applications simples. Ce genre de carte ne possède pas un degré de sécurité important : les données sont simplement stockées sur la carte et le traitement est fait au niveau du lecteur de carte.
- 2► **cartes à mémoire protégée** : les données stockées dans la carte sont protégées contre les accès non autorisés. Les cartes téléphoniques rentrent dans cette catégorie. Certaines zones sont accessibles seulement en lecture ou seulement lors de la personnalisation de la carte. Cette protection peut être effectuée à l'aide d'une logique câblée ou encore d'un code secret.
- 3► **cartes à microprocesseur** : contrairement aux cartes à mémoire, elles possèdent un microprocesseur capable de traiter les informations. Elles possèdent plus de mémoire ainsi qu'un degré de sécurité plus élevé. Ce genre de carte peut être utilisé pour de nombreuses applications comme dans le monde bancaire (ex : carte bancaire, carte de retrait, porte-monnaie électronique), le transport (en remplacement des tickets, des coupons, par exemple), la santé (ex : les projets « Sésam Vitale » ou « Santal »), les télécommunications (ex : téléphone cellulaire avec les cartes SIM) ou encore la télévision (ex : télévision à péage avec le système Viaccess).

sur le type de carte. Un numéro complet de carte est composé de 4 groupes de 4 chiffres (soit 16 chiffres en tout). Un numéro de carte a donc le format suivant : ABCD EFGH IJKL MNOP où chaque lettre correspond à un chiffre.

- ⇒ A désigne le type de carte (3 : American Express, 4 : Visa, 5 : Eurocard/MasterCard, 6 : Discover).
- ⇒ BCD, BCDE ou BCDEF donne le numéro de la banque (la longueur de ce numéro est variable selon l'organisme bancaire).
- ⇒ Les numéros qui suivent le numéro de la banque (O compris) identifient la carte (l'algorithme de construction de ce numéro est inconnu).
- ⇒ P correspond à la clé de Luhn qui permet de vérifier la validité du numéro complet de la carte.

⇒ Algorithme de Luhn

Du nom de l'inventeur allemand Hans Peter Luhn [<http://www.research.ibm.com/journal/rd/024/ibmrd0204H.pdf>], cet algorithme est utilisé pour la vérification de somme (*Checksum*) pour des valeurs numériques en base 10. Il comporte trois étapes :

- 1► Multiplication par deux d'un chiffre sur deux, en commençant par l'avant-dernier et en se déplaçant de droite à gauche. Si un chiffre multiplié par deux est supérieur à 9, alors ce nombre est modifié indifféremment par une de ces deux règles (le résultat est identique) :
 - ⇒ en additionnant les chiffres composant le nombre ;
 - ⇒ en soustrayant 9 au nombre.
- 2► Addition des chiffres obtenus et des chiffres non multipliés.
- 3► Si la somme est multiple de 10, alors le nombre initial respecte l'algorithme de Luhn.

Exemple avec le nombre 972487086 : $6 + 7 (8 \times 2 = 16, 1 + 6 = 7) + 0 + 5 (7 \times 2 = 14, 1 + 4 = 5) + 8 + 8 (4 \times 2 = 8) + 2 + 5 (7 \times 2 = 14, 1 + 4 = 5) + 9 = 50$, or 50 est divisible par 10, donc ce nombre vérifie l'algorithme de Luhn.

HISTORIQUE

INTRODUCTION

⇒ Exemple de normes dans les cartes à puce

Dans le domaine des cartes à puce, la demande de normalisation est très importante. En effet, un émetteur peut être réticent à investir dans une technologie qui ne sera pas forcément compatible avec les futures générations de cartes à puce. C'est pour cela que de nombreuses normes existent dans les différents domaines d'application.

⇒ **ISO 7816** : norme ISO (*International Standardization Organisation*) de référence pour les cartes à puce à contact. Elle spécifie entre autres les éléments physiques des supports plastiques, la signification et la localisation des contacts, les protocoles et contenus des messages de bas et haut niveau échangés avec les cartes. Notons que si les parties 1 à 3 de la famille des normes ISO 7816 sont spécifiques aux cartes à contact, les parties 4 et suivantes s'appliquent aussi aux cartes sans contact à microprocesseur.

⇒ **ISO 14443** : norme de référence pour les cartes à interface sans contact, pour des distances carte/lecteur ne dépassant pas quelques centimètres.

⇒ **ISO 21549** : norme de référence pour les cartes de santé.

⇒ **NF EN 15320** : norme française de référence pour les cartes de transport.

⇒ **EMV** : série de normes définie par Europay, Mastercard et Visa dans le cadre des institutions financières en 1996. Elle couvre les parties protocoles, données et instructions ainsi que les transactions impliquant des cartes à microprocesseur dans le domaine bancaire. Elle définit aussi un ensemble de mécanismes internes de protection des données assurant l'intégrité des secrets stockés dans la carte. Cette norme est conforme aux standards ISO 7816 et ISO 14443. En 2008, la version de référence de ces normes est 4.2 [2, 3, 4 et 5].

⇒ **C-SET** : norme mise en place en France, parallèlement à SET (pour *Secure Electronic Transaction*). Elle utilise les cartes à microprocesseur bancaires françaises (C-SET pour *Chip-Secure Electronic Transaction*). Ce système a été défini par le GIE Cartes Bancaires. Il utilise un lecteur de carte connecté à un micro-ordinateur. L'utilisateur « signe » son achat en entrant son code confidentiel. SET et C-SET sont interopérables.

⇒ **CEN** : normes CEN/TC224 et CEN/WG10 (CEN pour *Comité Européen de Normalisation*) édictées dans un cadre financier pour les porte-monnaie électroniques. Elles décrivent les données et instructions des cartes ainsi que les transactions et applications impliquant des porte-monnaie électroniques rechargeables.

⇒ **ETSI** : dans le domaine des télécommunications, l'ETSI (pour *European Telecommunications Standards Institute*) a établi des recommandations précises et exhaustives du système européen « GSM » (pour *Global System for Mobile communications*). Tous les réseaux de télécommunication en Europe les ont suivies et elles sont désormais imposées partout dans le monde.

Lorsqu'une carte est introduite dans un TPE ou un DAB, le processus d'authentification complet se déroule en trois étapes :

1► **Authentification de la carte** : elle se fait hors-ligne par le terminal (sans appeler un centre de contrôle). Si la carte possède une puce, alors le terminal lit *Informations* et *VS* contenues dans la puce. Il vérifie la signature RSA *VS* avec la partie publique de la clé du GIE CB et *Informations*.

2► **Code confidentiel** : le code entré par l'utilisateur est envoyé à la carte, qui le vérifie et se bloque après 3 essais infructueux consécutifs. Contrairement à ce qui se passerait pour une carte magnétique, la vérification du code secret est réalisée localement.

3► **Authentification en ligne** : cette étape n'est plus obligatoire. Elle est essentiellement réalisée pour des transactions dépassant un certain montant défini dans les contrats bancaires (il reste néanmoins possible d'effectuer cette étape quel que soit le montant de la transaction.). Si la carte possède une puce, alors le terminal interroge un centre de contrôle à distance qui retourne à la carte une valeur aléatoire x . La puce calcule alors le chiffré de cet aléa avec sa clé C_{sec} . Cette valeur est transmise au centre de contrôle qui effectue le même calcul. Si les deux valeurs sont identiques, le centre valide l'authentification. Cette étape implique que le centre de contrôle connaisse la clé secrète C_{sec} de toutes les cartes en service.

⇒ 2. YesCard et autres

Cette section décrit diverses attaques qui ont été utilisées contre le schéma de paiement B0. Les différentes parades mises en place suite à ces attaques sont décrites dans la section suivante.

Pour réaliser une fausse carte de paiement, il faut être capable de réaliser une carte qui va respecter l'algorithme de paiement en utilisant des informations plus ou moins erronées. Il est nécessaire de trouver ou de construire les informations qui doivent être présentes sur la carte et ensuite de les utiliser correctement à chaque étape du protocole. Nous donnons, pour chacune de ces étapes, les techniques utilisées pour la contourner.

1► **Authentification de la carte** : la première étape est l'authentification de la carte où la carte doit fournir les

Informations et un *VS* valide. Il existe deux possibilités pour les obtenir : utiliser des valeurs existantes, correspondant à un véritable compte bancaire (clonage de carte) ou en construire

de nouvelles, non liées à un compte.

Dans ce dernier cas, il est nécessaire d'avoir accès à la clé privée de signature (attaque de Serge Humpich). Le clonage de carte est possible, car *Informations* et *VS* sont lisibles avec un simple lecteur de

carte, celles-ci n'étant pas protégées par un code. Mais cette technique nécessite d'avoir accès à une véritable carte, ce qui n'est pas le cas pour l'attaque mise en œuvre par Serge Humpich. *VS* est calculé avec une clé RSA de 321 bits (carte bancaire de première génération). Une attaque connue sur une clé RSA consiste à factoriser son module en deux nombres premiers. Cette factorisation est un problème classique de

...YesCard, une attaque parmi d'autres...

cryptographie, très difficile à résoudre et d'autant plus difficile que la taille de la clé est grande. Dès 1988, des experts estimaient qu'une clé de 320 bits ne résisterait pas longtemps à cette attaque par factorisation et préconisaient une taille de 512 bits [6]. En 1997, Serge Humpich a donc exploité cette faiblesse en utilisant un outil de factorisation japonais afin de retrouver la clé privée du GIE CB [7]. Ceci fait, il lui suffisait de choisir *Informations* et de calculer le VS correspondant.

- 2 ➤ **Code confidentiel** : la deuxième étape est celle du code confidentiel. Le code porteur empêche l'utilisation frauduleuse d'une vraie carte, mais pas la simulation ou le clonage, car c'est la carte qui valide le code rentré par l'utilisateur. Il suffit que la carte réponde oui quelle que soit la valeur donnée, d'où son nom de YesCard. Ceci est vrai quelle que soit la technique employée pour la première étape.
- 3 ➤ **Authentification en ligne** : la dernière étape du protocole est une authentification en ligne avec une clé qui ne peut pas sortir de la puce. Cette étape n'a jamais pu être falsifiée, la clé n'a jamais fui ni de la puce, ni du GIE CB. Le fait que cette clé soit encore inconnue aujourd'hui a limité l'utilisation des YesCard. En effet, sans cette clé, il est impossible de tromper l'authentification en ligne. De ce fait, les YesCard ne peuvent être utilisées que pour de petites valeurs et sur certains terminaux de paiement uniquement.

La YesCard est sans doute l'attaque la plus connue dans le domaine des cartes bancaires alors qu'elle n'est ni la plus facile à mettre en place, ni la plus utilisée. En effet, il existe une attaque plus simple. Avec seulement les seize chiffres inscrits sur les cartes, il était possible d'acheter sur Internet ou par correspondance. Ce nombre était disponible sur les factures des terminaux de retrait et de paiement. Voilà pourquoi les voleurs raffolaient de ces petits tickets.

Dans la même veine, une attaque très simple consiste à cloner la piste magnétique. En effet, celle-ci est toujours lisible à l'aide d'un lecteur de piste magnétique et d'un petit logiciel. Il suffit d'écrire ces informations sur la piste d'une carte blanche. Cette attaque fonctionne, car le système de paiement électronique n'est pas uniformisé : dans beaucoup de pays, les cartes bancaires n'ont pas de puce et seule la piste magnétique est utilisée par les terminaux de paiement. Cela suppose aussi que le marchand ne soit pas très regardant sur la carte et ne vérifie pas l'hologramme, qui doit être présent, ni la qualité d'impression. Dans les faits, cela ne semble pas correspondre à une situation répandue : en réalité les voleurs préfèrent s'attaquer aux distributeurs de billets ou alors, plus marginalement, le responsable du terminal (caissier) est complice.

Dans les cas précédents, l'intégrité de la puce n'a pas été mise à mal. Les attaques utilisent des failles dans le protocole. Bien sûr, les puces ne sont pas infaillibles. La littérature regorge de cas d'attaques qui ont été mises en œuvre pour extraire les secrets des cartes : attaques physiques en allant inspecter au plus profond du composant les informations directement dans la mémoire ou attaques par canaux cachés en exploitant les informations qui émanent de la carte pendant son utilisation (temps de traitement d'une opération, variation de courant, radiations électromagnétiques...) [8, 9].

➔ Les cartes à puce en quelques dates

- 1967 : René Barjavel écrit *La Nuit des temps* [1] ;
- 1967 : premiers brevets déposés aux USA (jamais utilisés) ;
- 1972 : naissance du GIE Carte Bleue (GIE CB) ;
- 1974-1975 : brevets de Roland Moreno sur les cartes à mémoire ;
- 1977 : brevet de Michel Ugon sur les cartes à microprocesseur ;
- 1978 : brevet « SPOM » (*Self Programmable One chip Microprocessor*) de Michel Ugon sur les cartes mono-composant ;
- 21 mars 1979 : sortie de la première carte à mono-circuit, elle avait deux composants (Michel Ugon, Bull) ;
- 1979 : début du GSM ;
- 1981 : sortie de la première carte mono-composant à micro-circuit ;
- 1983 : début de la télécarte française ;
- 1983 : le centre national d'études des télécoms (CNET) spécifie la première carte bancaire à puce ;
- 1984 : naissance du GIE Cartes Bancaires comme successeur du GIE Carte Bleue (lui aussi abrégé GIE CB) ;
- 1985 : le GIE Cartes Bancaires commande seize millions de cartes à puce ;
- 1988-1992 : première phase du projet de la carte de santé « Santal » ;
- 1992 : le GSM passe du concept de laboratoire au produit commercial ;
- 1992 : toutes les cartes « CB » sont équipées d'un microcircuit ;
- 1994 : développement important des cartes de Sécurité Sociale en Allemagne (environ soixante-dix millions en 1997) ;
- 1995 : début de la deuxième phase du projet « Santal » ;
- 1995 : début du projet « Sésam Vitale » ;
- 1996 : développement important du porte-monnaie électronique ;
- Printemps 1997 : Serge Humpich parvient enfin à percer le secret des cartes bancaires ;
- 1998 : Serge Humpich essaie de négocier sa découverte avec le GIE CB qui porte plainte contre lui ;
- Juin 1999 : l'affaire Humpich éclate publiquement ;
- 1999 : généralisation sur toute la France et les DOM-TOM de la carte « Sésam Vitale » (environ quarante millions de cartes) ;
- 1999 : mise en service de nouvelles cartes bancaires en réponse à l'attaque de Humpich ;
- Février 2000 : Serge Humpich est condamné à 10 mois de prison avec sursis ;
- 2000 : publication des failles de la carte bancaire et développement des YesCard ;
- 2004 : le système Carte Bancaire passe au standard international EMV ;
- 2004 : les premières cartes bancaires sans contact.

⇒ 3. Impact sur la carte bancaire

Toutes ces attaques n'ont laissé personne indifférent. L'attaque proposée par Serge Humpich a fait couler beaucoup d'encre. Cette médiatisation a eu plusieurs effets.

⇒ 3.1 Erreur médiatique

Le premier impact de cette affaire est indéniablement sur l'image de la carte à puce. Les articles ou les reportages qui en ont parlé ont titré sur le cassage de la carte à puce au lieu de préciser que c'était le système de paiement par carte bancaire qui avait été mis à mal (c'est-à-dire le protocole de paiement). La clé secrète utilisée pour l'authentification en ligne n'a jamais été extraite de la carte. Cette erreur dans la communication a eu pour effet de faire baisser la confiance dans la carte à puce alors que c'est bien la puce qui apporte la sécurité dans ce mode de paiement. L'effet de cette erreur de communication de la part du GIE CB et le goût du sensationnel des journalistes ont fait que de nombreuses personnes ont perdu confiance en la carte bancaire, hésitant à s'en servir.

En réaction à ce cafouillage médiatique, Roland Moreno a lancé un pari d'un million de franc le 13 mars 2000. Le pari était le suivant :

« Je mets en jeu une somme de UN MILLION de francs sur la base de l'épreuve suivante :

A/ quiconque parviendrait à écrire un bit dans la zone préservée par mon brevet « Inhibiteur » (17 mars 1975), et

B/ quiconque parviendrait à lire un bit dans la zone préservée par la combinaison de mes deux brevets « Comparateur » et « Compteur d'Erreurs »

recevrait en paiement UN MILLION de francs, somme offerte par Innovatron, la société que je dirige depuis vingt six ans, et qui détient les brevets de base de la carte à puce.

Conditions de l'épreuve : le joueur restera enfermé pendant un mois dans un laboratoire, équipé de tous les instruments logiques dont il souhaitera disposer y compris une liaison, aussi rapide qu'exigé, avec un ordinateur de quelque puissance que ce soit.

Le joueur pourra être un ingénieur, un étudiant ou un amateur.

Offre valable trois mois. »

Personne n'a pu relever ce défi et Roland Moreno a gardé son million. Il faut quand même ajouter qu'étaient exclus du champ de cette épreuve les moyens analogiques de toute nature, ainsi que les moyens chimiques, mécaniques, optiques, électriques ou radioélectriques.

「...pour 1 million,
trouvez la faille...」

⇒ 3.2 Mise à jour du protocole

Le GIE CB a d'abord nié la véracité de l'attaque de Serge Humpich, mais a quand même assez rapidement réagi et mis à jour le protocole en 1999. Le GIE CB a aussi poursuivi Serge Humpich devant les tribunaux pour falsification de moyen de paiement. Les différentes étapes du protocole de paiement ont été conservées. La taille de la clé RSA a été allongée de 320 bits à 768 bits. À l'heure où cet article est écrit, aucun module RSA aussi grand n'a encore été factorisé, le record pour un module RSA étant de 663 bits [10]. Cette factorisation sera possible d'ici quelques années, sachant que le plus grand nombre factorisé aujourd'hui est de 313 digits [10], mais ce nombre possède une propriété mathématique que n'ont pas les modules RSA (pour information, un module RSA de 1024 bits est composé de 309 chiffres). Cette nouvelle clé de 768 bits répond donc temporairement à l'attaque de Serge Humpich, mais n'empêche en aucun cas le clonage de carte. De plus, une VS de 320 bits a été longtemps présente sur les cartes, même sur les nouvelles, car tous les terminaux de paiement ne supportaient pas la version 768 bits. La migration de tous les terminaux s'est terminée en 2003.

Bien que la clé secrète utilisée pour l'authentification en ligne n'ait jamais été trouvée, le GIE CB a quand même profité de cette mise à jour pour changer l'algorithme de chiffrement symétrique en passant du DES (clé de 56 bits) au 3DES (clé de 112 bits). Cette nouvelle clé n'a pas été trouvée et il est toujours impossible de fabriquer des cartes qui soient utilisables lors d'authentifications en ligne.

⇒ 3.3 EMV : nouveau protocole de paiement

La définition d'un nouveau système de paiement à base de carte a d'abord répondu à un besoin de remplacement des nombreuses cartes à piste magnétique encore utilisées dans de nombreux pays. Cette technologie est en effet devenue obsolète et n'offre qu'un très faible niveau de sécurité. Elle est responsable d'une grande partie des fraudes à la carte bancaire. La France, bien sûr, est déjà passée à la carte à puce, mais, pour des raisons d'interopérabilité dans le cas de voyage à l'étranger par exemple, toutes les cartes disposent encore de cette piste magnétique. Les spécifications EMV définissent des clés RSA de taille pouvant aller jusqu'à 1984 bits. Deux protocoles d'authentification sont proposés. L'un est statique et l'autre dynamique. Le choix de l'un ou l'autre des protocoles d'authentification est fait par le terminal en fonction des capacités de la carte annoncées par celle-ci au début de la transaction. Ce nouveau système de paiement

se met progressivement en place dans tous les pays. En 2008, en France, la migration vers EMV est quasiment terminée : 100% des cartes et des DAB et 98% des TPE [11].

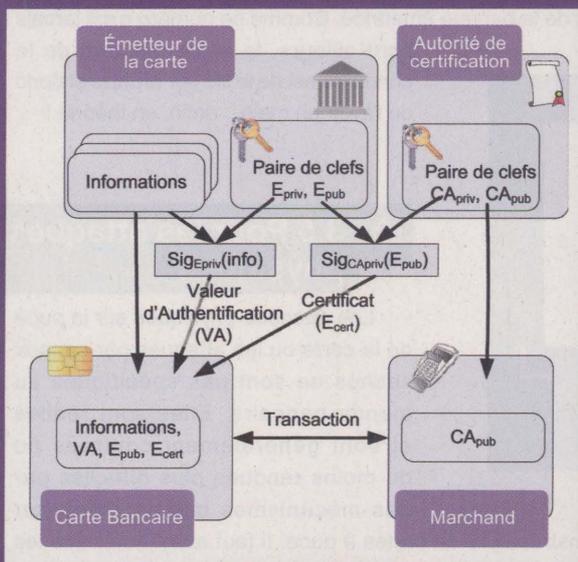
Le protocole EMV fait intervenir à peu de choses près les mêmes acteurs que l'ancien protocole :

- ⇒ la banque du client, noté *Émetteur de la carte* ;
- ⇒ le client, noté *Carte Bancaire* ;
- ⇒ le TPE ou le DAB, noté *Marchand* ;
- ⇒ une autorité de certification, noté *CA*.

L'authentification statique (SDA pour *Static Data Authentication*) consiste pour le terminal à vérifier une donnée signée mise dans la carte durant sa personnalisation.

Pendant la phase de personnalisation, la carte reçoit les informations suivantes (Figure 2) :

- ⇒ le nom du porteur, le numéro de la carte ou encore la date limite de validité de celle-ci (notés *Informations*) ;
- ⇒ une valeur d'authentification (noté *VA*), signature RSA d'*Informations* générée avec la partie privée de la clé de l'émetteur ($VA = \text{Sig}_{E_{priv}}(\text{Information})$) ;
- ⇒ le certificat de l'émetteur (E_{cert}) contenant sa clé publique signée par une autorité de certification ;
- ⇒ le code PIN transmis au porteur de cette carte.



2 Organisation des acteurs dans le protocole de paiement EMV avec SDA

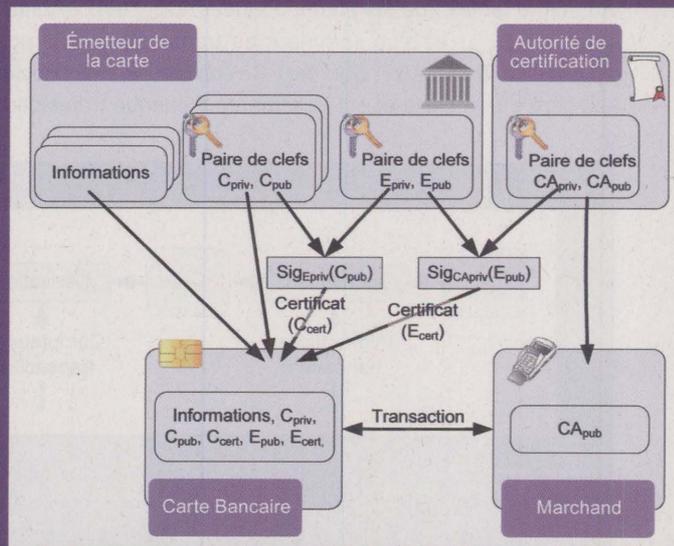
Lorsqu'une carte est introduite dans un terminal, le processus d'authentification se déroule selon les étapes suivantes :

- 1► La carte fournit au terminal *Informations*, le certificat E_{cert} de la banque émettrice, ainsi que la valeur d'authentification *VA*.
- 2► Le terminal vérifie E_{cert} avec la clé publique de l'autorité de certification (CA_{pub}) et vérifie *VA* avec la clé publique de la banque émettrice.
- 3► Le terminal demande à l'utilisateur le code PIN et le transmet (en clair) à la carte pour qu'elle le vérifie.

L'authentification dynamique (DDA pour *Dynamic Data Authentication*) permet, en plus d'une authentification statique, de vérifier que la carte possède un secret délivré par l'émetteur de la carte.

Pendant la phase de personnalisation, la carte reçoit les informations suivantes (Figure 3) :

- ⇒ le nom du porteur, le numéro de la carte ou encore la date limite de validité de celle-ci (notés *Informations*) ;
- ⇒ une paire de clés RSA (C_{pub} et C_{priv}) ;
- ⇒ un certificat (C_{cert}) contenant C_{pub} signée par l'émetteur ;
- ⇒ le certificat de l'émetteur (E_{cert}) contenant sa clé publique (E_{pub}) signée par une autorité de certification ;
- ⇒ le code PIN transmis au porteur de cette carte.



3 Organisation des acteurs dans le protocole de paiement EMV avec DDA

Lorsqu'une carte est introduite dans un terminal, le processus d'authentification se déroule selon les étapes suivantes :

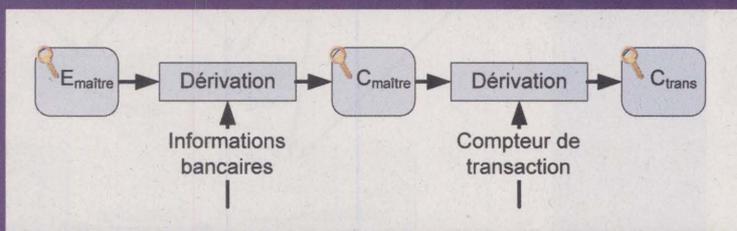
- 1► La carte fournit au terminal *Informations*, le certificat E_{cert} de la banque émettrice et son certificat C_{cert} .

HISTORIQUE

INTRODUCTION

- 2▶ Le terminal génère une valeur aléatoire T_{alea} et l'envoie à la carte.
- 3▶ La carte génère une valeur aléatoire C_{alea} . Puis, elle signe T_{alea} et C_{alea} avec sa clé privée E_{priv} . Elle envoie le résultat de la signature et C_{alea} au terminal.
- 4▶ Le terminal vérifie E_{cert} avec CA_{pub} et vérifie C_{cert} avec E_{pub} . Puis, il vérifie la signature des aléas avec C_{pub} .
- 5▶ Le terminal demande à l'utilisateur le code PIN et l'envoie à la carte pour qu'elle le vérifie. Contrairement à l'authentification statique, le code PIN est envoyé chiffré par C_{pub} . Le code PIN est d'abord concaténé avec deux nouvelles valeurs aléatoires fournies par la carte et le terminal, ceci afin d'éviter les attaques par rejeu.

La transaction peut ensuite être finalisée hors ligne ou en ligne. Le choix peut être fait par la carte et/ou le terminal selon une politique de gestion de risques : sélection aléatoire, validation en ligne pour n validations hors ligne, en fonction du montant de la transaction, du montant cumulé des transactions déjà effectuées hors ligne ou d'un plancher fixé par le marchand. Dans les deux cas, une clé secrète (clé 3DES de 112 bits) est utilisée. Celle-ci est unique par transaction et notée C_{trans} (Figure 4). La carte possède une clé maître ($C_{maître}$) qui est générée par la banque émettrice à partir d'une clé maître de la banque ($B_{maître}$) et des informations bancaires. $C_{maître}$ est mise dans la carte lors de la personnalisation en même temps que les autres données. C_{trans} est calculée à partir de $C_{maître}$ et d'un compteur de transaction (ATC pour *Application Transaction Counter*). Ce compteur sur deux octets est géré par la carte et est incrémenté à chaque transaction.



4 Algorithme pour la dérivation de la clé de transaction

Dans le cas hors ligne, le terminal envoie à la carte les détails de la transaction. La carte produit alors un certificat de transaction (TC) en signant ces données (algorithme *DES CBC-MAC*) à l'aide de C_{trans} . Le terminal ne peut pas vérifier TC, mais le garde pour validation ultérieure auprès de sa banque. Une variante CDA (pour *Combined Data Authentication*) de l'authentification DDA utilise également TC. Elle consiste à inclure TC dans le bloc de donnée signé par la carte. Si la transaction doit être approuvée en ligne, le terminal envoie à la banque émettrice le cryptogramme généré

par la carte (ARQC pour *Authorization ReQuest Cryptogram*). La banque le vérifie et génère un cryptogramme réponse (ARPC pour *Authorization ResPonse Cryptogram*) qui est renvoyé à la carte via le terminal. Le terminal redemande alors à la carte de lui générer un certificat de transaction qui inclut, cette fois-ci, l'autorisation de la banque émettrice.

⇒ 3.4 Et sur Internet ?

Il est clair que la seule utilisation des seize chiffres visibles de la carte ne peut pas être un protocole de paiement sûr. La première mesure a été de ne plus imprimer ce numéro sur les factures. Étant donné l'algorithme utilisé pour la fabrication de ce numéro à seize chiffres, sachant que neuf chiffres sont encore inscrits sur les factures et qu'ils correspondent plus ou moins à l'aléa choisi par la banque, il est possible de reconstituer le numéro entier dans de nombreux cas.

En 2001, un cryptogramme visuel a été ajouté et imprimé uniquement sur la carte (au niveau de la signature au dos). Il s'appelle CVV chez Visa (pour *Card Verification Value*) et CVC chez MasterCard (pour *Card Validation Code*). Ce code supplémentaire est généré par la banque émettrice à partir des informations bancaires du client et de données secrètes de la banque. Ce nombre de trois ou quatre chiffres ne peut pas être reconstruit, car l'algorithme est encore secret et toutes les données ne sont pas encore connues. Depuis 2004, les sites marchands ont l'obligation de demander ce cryptogramme en plus du numéro de carte afin de valider toutes transactions à distance auprès de la banque émettrice. Comme ce numéro n'est jamais écrit ailleurs, la seule manière de le connaître est de le lire sur la carte et donc de l'avoir en main... enfin, en théorie !

⇒ 3.5 Pour les attaques physiques

Les attaques physiques sur la puce de la carte ou les attaques par canaux cachés ne sont pas spécifiques au monde bancaire. Elles sont réelles et sont généralement contrées ou du moins rendues plus difficiles par des mécanismes mis en place par

les constructeurs de cartes à puce. Il faut aussi noter que les attaques physiques sont généralement coûteuses et que seules de grandes organisations peuvent les mettre en œuvre dans un processus quasi industriel. Voici quelques exemples de contre-mesures physiques :

- ⇒ mise en place de couches métalliques supplémentaires sur la puce empêchant des sondes microscopiques ou des faisceaux lumineux d'atteindre les parties actives de la puce ;

⇒ réseaux de détecteurs de conditions anormales de fonctionnement (température, fréquence horloge, tension d'alimentation, luminosité...).

Des mécanismes logiciels existent également :

⇒ redondance au niveau variables ou opérations ; utilisation de plusieurs variables pour la même donnée ou exécution multiple de la même opération afin de contrer les attaques par fautes ;

⇒ dispersion des données secrètes dans plusieurs emplacements en mémoire.

⇒ Conclusion

Nous avons expliqué dans cet article les faiblesses du système de paiement par carte bancaire et les solutions qui ont été mises en place pour les éliminer. Le protocole EMV, qui est maintenant déployé dans de nombreux pays afin de contrer les fraudes, présente aussi des vulnérabilités [12] :

⇒ L'authentification DDA n'est pas obligatoire et l'authentification SDA présente la même faiblesse que l'ancien protocole ; elle utilise effectivement une clé RSA suffisamment longue (jusqu'à 1984 bits), mais la donnée d'authentification peut être lue sans la présentation d'un code et peut donc être recopiée pour cloner la carte. De plus, le code PIN est envoyé en clair à la carte et peut être intercepté entre le lecteur et la carte (la difficulté étant de le faire discrètement !). L'authentification DDA est plus robuste, mais nécessite une carte capable de générer une signature, c'est-à-dire une carte avec une puce plus évoluée et donc plus chère. Il semble que, depuis peu, la majorité des cartes bancaires en circulation supportent l'authentification DDA.

⇒ Il manque des liens entre l'authentification, la vérification du code PIN et la génération de TC. Les requêtes envoyées

par le terminal peuvent être interceptées et des réponses forgées peuvent être renvoyées : OK à une vérification du code PIN, TC modifié lors d'une transaction hors ligne de telle sorte que le paiement soit refusé par la banque...

⇒ Les terminaux de paiement ne présentent pas toujours un niveau de sécurité suffisant pour contrer des attaques physiques.

Une nouvelle tendance se développe pour le paiement. Il s'agit du paiement sans contact où la puce de la carte communique avec le terminal par radio-fréquence, conformément à la norme ISO 14443 ; la carte est simplement passée devant le terminal pour effectuer la transaction. Cela concerne des paiements de petites valeurs et met en œuvre un nouveau protocole. D'autres systèmes de paiement voient le jour, dans lesquels la puce n'est plus sur une carte, mais intégrée à un système complet, comme un téléphone portable.

Nouveaux systèmes, nouveaux protocoles et nouvelles failles ? Le problème est encore et toujours de trouver un compromis entre le coût, la facilité d'usage et la sécurité... sans que cela ne se fasse au détriment du troisième ! ■

i Références

⇒ [1] BARJAVEL (René), *La nuit des temps*, Édition Denoël, 1967.

⇒ [2] EMV: Book 1 - *Application Independent ICC to Terminal Interface Requirements*, 2008, version 4.2.

⇒ [3] EMV: Book 2 - *Security and Key Management*, 2008, version 4.2.

⇒ [4] EMV: Book 3 - *Application Specification*, 2008, version 4.2.

⇒ [5] EMV: Book 4 - *Cardholder, Attendant, and Acquirer Interface Requirements*, 2008, version 4.2.

⇒ [6] GUILLOU (Louis C.), *Techniques à clé publique*, Annales des Télécommunications, volume 43, n°9-10, 1988.

⇒ [7] HUMPICH (Serge), *Le cerveau bleu*, Édition Xo, 25 janvier 2001.

⇒ [8] SKOROBOGATOV (Sergei P.), *Semi-invasive attacks - A new approach to hardware security analyse*, Université de Cambridge, avril 2005.

⇒ [9] Flylogic Engineering's Analytical Blob : <http://www.flylogic.net/blog/> (site consulté en juillet 2008)

⇒ [10] ZIMMERMAN (Paul), *Records de factorisation de nombre entier [en ligne]*, disponible sur : <http://www.loria.fr/~Ezimmerma/records/factor.html> (page consultée en juillet 2008).

⇒ [11] Observatoire de la sécurité des cartes de paiement, *Rapport annuel 2007*, juillet 2008, <http://www.observatoire-cartes.fr> (site consulté en juillet 2008).

⇒ [12] DRIMER Saar, MURDOCH (Steven J.) et ANDERSON (Ross), *Thinking inside the box: system-level failures of tamper proofing*, Computer Laboratory, Université de Cambridge, février 2008.

LA CARTE SIM OU LA SÉCURITÉ DU GSM PAR LA PRATIQUE

■ mots clés : GSM / sécurité / SIM / USIM / carte à puce

La carte SIM (ou Subscriber Identity Module, mot à mot le module d'identité d'un abonné) est le type de carte à puce le plus vendu de par le monde. Environ trois milliards d'unités ont été fabriquées en 2007, soit une carte pour deux habitants de la planète. Ce produit constitue véritablement le poumon de l'industrie de la carte à puce, dont la première entreprise mondiale, Gemalto, réalise un chiffre d'affaire d'environ deux milliards d'euros.

La première génération de téléphonie mobile déployée au cours des années 80 utilisait du matériel électronique cher et encombrant ; elle s'adressait à une gamme très restreinte de clientèle VIP. L'étude de la deuxième génération de mobiles, ou GSM (*Global System for Mobile communications*), démarra en 1982 sous l'égide de la Conférence Européenne des Postes et Télécommunications (le CEPT). Le succès et l'adoption massive de cette technologie (il y a plus de mobiles en France que d'habitants) paraissait à cette époque très incertaine. Depuis 1989,

l'ETSI (*European Telecommunications Standards Institute*), dont le siège est localisé à Sophia Antipolis, près de Nice, édite les spécifications du GSM et de son successeur (ou réseau de troisième génération) l'UMTS (*Universal Mobile Telecommunications System*). Parce que la France est le berceau des cartes à puce, l'idée d'utiliser

cette pastille de silicium s'imposa dès 1988, afin de prévenir les futurs opérateurs mobiles d'une fraude massive, source potentielle de pertes financières. Car les ondes radio forment un brouillard diffus, et, en l'absence de mesure de sécurité particulière, tout à chacun peut accéder librement et gratuitement à cette ressource. Cependant, outre-atlantique le réseau mobile CDMA (l'équivalent du GSM) ne possède pas de module SIM. Cette opposition se vérifie également pour les cartes bancaires, fonctionnant uniquement avec des pistes magnétiques et exemptes de puces électroniques.

Cet article est dédié à la carte SIM. Il propose une approche pratique quant à la compréhension de ses fonctionnalités, à des fins de curiosité intellectuelle ou même d'enseignement. Pour des raisons de concision, nous n'introduisons pas la carte USIM, déployée dans les réseaux UMTS, dont la structure est très proche de celle de la SIM, mais qui met en œuvre un algorithme d'authentification plus complexe, nommé AKA. Un lecteur de carte compatible avec les ordinateurs personnels usuels (sous Windows en particulier) coûte environ 25€. On se procure facilement dans le commerce des cartes SIM prépayées (autour de 15€) offrant une base d'exploration idéale (cf. Figure 1). Une ancienne SIM réalise également un bon support expérimental. Les versions JAVA 1.6 et supérieures permettent d'accéder aux cartes SIM à l'aide d'un lecteur et d'un PC ; nous proposons quelques codes sources très simples, destinés à la pratique des concepts exposés.



Une carte SIM prépayée et un lecteur de cartes (pour un coût d'environ 40 €.)

1. Les fonctions d'une carte SIM

Afin d'assurer la sécurité des opérateurs mobiles et de leurs abonnés, quatre classes de méthodes de protection sont proposées dans le document GSM 02.09 facilement téléchargeable depuis le site de l'ETSI :

- ⇒ La protection de l'identité d'un abonné. Ce dernier possède un identifiant (ou IMSI ou *International Mobile Subscriber Identity*) permettant de retrouver dans la base de données des comptes clients (HLR, *Host Location Register*) les paramètres de son abonnement. Parce que la cellule GSM s'étend de quelques centaines de mètres à plusieurs dizaines de kilomètres, il est vital de limiter la diffusion de cette information afin de garantir le respect de la vie privée. Dès que possible, le réseau délivre un pseudonyme (le TMSI, *Temporary Mobile Subscriber Identity*), modifié à chaque nouvel appel afin d'interdire la traçabilité des communications par des oreilles indiscrettes écoutant le trafic GSM.
- ⇒ L'authentification d'un abonné. L'opérateur souhaite limiter les fraudes sur son réseau et ses clients désirent que leurs forfaits ne soient pas consommés par des pirates. En conséquence, le réseau GSM met en œuvre un mécanisme d'authentification « fort » ; l'algorithme A3, associé à une clé secrète K_i de 128 bits.
- ⇒ La confidentialité des données utilisateur. Dans un téléphone fixe, la voix est transmise à l'aide de câbles ; des écoutes sont possibles, mais elles sont difficiles (des branchements sont nécessaires sur le boîtier de l'abonné ou au central téléphonique) et de surcroît illégales. Dans un réseau cellulaire radio, l'information est transmise par des ondes électromagnétiques (*Over The Air*) entre le téléphone portable et une station de base ; des récepteurs électroniques bon marché sont capables de recevoir et d'enregistrer ces signaux. Les conversations entre mobile et station de base sont donc chiffrées à l'aide d'un algorithme nommé A5 utilisant une clé de chiffrement K_c . Cette dernière n'est pas fixe, mais mise à jour au cours de la procédure d'authentification (et donc à chaque appel), grâce à l'algorithme de génération de clé A8. Généralement, les algorithmes A3 et A8 sont confondus, une seule fonction cryptographique, nommée A38 (ou A3A8) assure ces calculs.
- ⇒ La protection de certaines informations de signalisation, par exemple l'IMSI, les numéros appelés ou appelants. Le numéro de série du portable (ou IMEI *International Mobile Equipment Identity*) constitue également une donnée sensible.

L'infrastructure d'authentification du GSM se divise en cinq entités fonctionnelles : la carte SIM, le mobile, une entité associée à plusieurs stations de base, le VLR (pour *Visitor Location Register*), la base de données clients (HLR, *Host Location Register*) et, enfin, le centre d'authentification (AuC, *Authentication Center*).

La norme GSM 03.20 décrit de manière concise les principes de mise en œuvre des éléments de sécurité dans un environnement GSM ; ces notions sont illustrées par la figure 2. Une cellule ou un ensemble de cellules sont identifiées par une étiquette LAI (*Location Area Identity*).

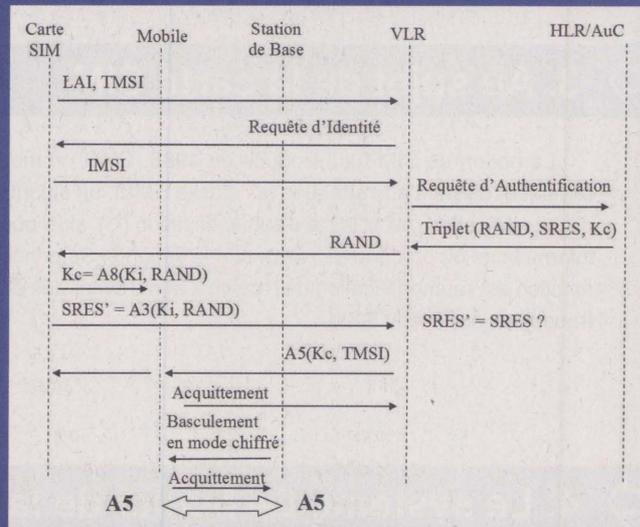
Suite à un précédent appel, un abonné dispose d'un couple de valeurs LAI-TMSI stocké dans son module SIM ; la validité de ces informations est incertaine, en particulier après une mise hors tension du portable. Le mobile transmet au VLR les indications LAI et TMSI ; ce dernier tente de retrouver l'IMSI identifiant de manière univoque un abonné. En cas d'échec de cette opération, il délivre au mobile une requête d'identification qui récupère en clair l'IMSI mémorisé dans la carte SIM.

À ce stade, le VLR connaît l'IMSI de l'abonné. Il transmet au HLR (qui est généralement regroupé avec le bloc AuC) une demande d'authentification. Si le compte utilisateur est valide dans la base de données de l'opérateur, l'AuC produit une suite de valeurs connue sous l'appellation « triplet du GSM », trois nombres notés RAND, SRES et K_c .

RAND est un nombre aléatoire de 16 octets, SRES (*Signed Response*, mot à mot réponse signée) est calculé par l'algorithme A3 associé à la clé K_i , soit $SRES = A3(K_i, RAND)$.

K_c est la clé utilisée pour le chiffrement des communications. Elle est déduite de l'algorithme A8 associé à la clé K_i , soit $K_c = A8(K_i, RAND)$.

Ce mécanisme est une caractéristique originale du GSM ; en effet, le VLR obtient de la part du HLR un ou plusieurs triplets d'authentification. Une fois en possession d'un triplet (RAND, SRES, K_c)



2 Principes de sécurité d'un réseau GSM

le VLR transmet au mobile un défi **RAND**. La carte SIM exécute alors la fonction **A3** dont le résultat ($SRES'=A3(Ki, RAND)$) est renvoyé au HLR qui vérifie alors l'égalité entre **SRES** et **SRES'**. En cas de succès, la légitimité de l'abonné est confirmée.

Le VLR choisit un nouveau TMSI, réalise son chiffrement avec l'algorithme **A5** muni de la clé **Kc**, puis transmet ce paramètre au mobile qui effectue l'opération de déchiffrement.

C'est la station de base dont dépend le mobile, qui décide du basculement en mode chiffré (basé sur l'algorithme **A5** muni de la clé **Kc**)

des communications. Le mobile (et non la carte SIM) est en charge des opérations de chiffrement et de déchiffrement qui s'appliquent uniquement aux signaux radio. Au-delà des stations de bases, dans le réseau câblé de l'opérateur, les informations sont transportées sans garantie de confidentialité.

La carte SIM réalise le calcul **A3A8** dans un espace sûr, un coffre-fort numérique en quelque sorte. Cependant, les algorithmes utilisés par le réseau GSM ont subi de multiples attaques qui ont permis le clonage de certaines SIM.

2. Les attaques de la SIM

En 1998, des chercheurs de la célèbre université de Berkeley (Californie), Mark Briceno, Ian Goldberg et David Wagner ont cassé **A3A8**. Bien que la norme GSM ne recommande aucun algorithme particulier, la quasi-totalité des opérateurs utilisait une même procédure « secrète », le **COMP128-1**. Le trio en réalisa un *reverse engineering* et mit au point un procédé capable de retrouver la clé secrète **Ki** en 2^{19} calculs (environ 500 000 essais), ce qui implique cependant l'usage à 100 % d'une carte SIM pendant 24h ! En 2002, un groupe de chercheurs du *Watson Research Center* d'IBM et de l'Institut de Recherche Fédéral Suisse publia une méthode d'attaque (*Partitioning Attack*) permettant d'extraire la clé **Ki** avec 8 (2^3) calculs seulement ! Cependant, ce procédé, impliquant des écoutes électromagnétiques, nécessite un équipement matériel particulier et exploite l'absence de protection physique pour des SIM bas de gamme.

Afin de contrecarrer l'attaque de Berkeley, les composants qui intègrent le **COMP128-1** sont munis d'un compteur limitant le nombre d'appel, par exemple à 100 000. Pour cette raison, les logiciels de

cassage qui implémentent l'attaque de Berkeley peuvent mettre hors service une SIM, lorsque le compteur dépasse la limite fixée par le constructeur. Le parc SIM est supposé être majoritairement basé aujourd'hui sur un algorithme **COMP128-2** dont les détails ne sont pas publiés.

Les algorithmes de chiffrement des paquets GSM, le **A5/1** (qualifié de « fort », car restreint à l'exportation) et le **A5/2** (qualifié de faible et libre d'usage) sont également cassés. En 1999, l'institut israélien Weizmann associé à l'université de Berkeley décrit une méthode d'attaque de **A5/1** permettant de déduire la clé **Kc** après seulement 2 secondes d'enregistrement des paquets GSM (soit une trame de 114 bits, chiffrée avec **A5**, émise à chaque intervalle de 4.6 millisecondes). La même année, l'université de Berkeley réussit une première extraction d'une clé (**Kc**) associée à **A5/2** avec 2048 paquets (soit environ 6 secondes de trafic) ; en 2003, un groupe de chercheurs de l'Institut Israélien de Technologie (*Technion*) démontra une nouvelle méthode efficace en seulement quelques douzaines de millisecondes.

3. Au sujet de la carte SIM

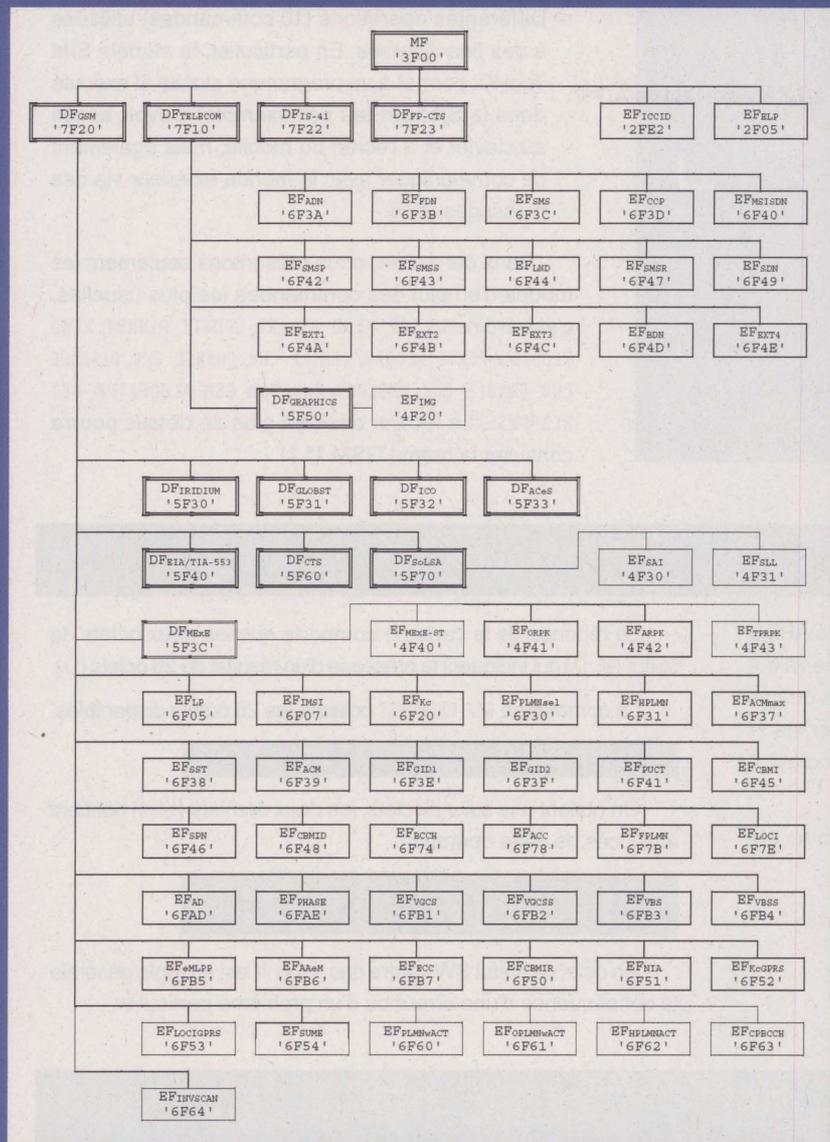
La notion de SIM fut introduite en 1988. C'est un module physique résistant aux attaques qui stocke l'IMSI, un algorithme d'authentification (**A3A8**), la clé d'authentification (**Ki**), ainsi que les informations ou procédures relatives à la sécurité. Sa principale fonction est l'authentification de l'abonné, afin d'éviter des accès frauduleux au réseau GSM.

Au début des années 90, une SIM comportait un CPU de 8 bits, 128 octets de RAM (!), 7 Ko de ROM et 3 Ko d'EEPROM (mémoire non volatile réinscriptible). En 2008, c'est un petit ordinateur puissant équipé d'un CPU de 32 bits, d'une ROM de 512 Ko, d'une RAM de 16 Ko et d'une mémoire non volatile (FLASH ou EEPROM) de l'ordre de 512 Ko. Les calculs cryptographiques sont facilités grâce à un coprocesseur dédié. L'ensemble occupe une surface de silicium inférieure à 25 mm².

4. Le système de fichier de la SIM

Conformément aux standards ISO 7816, le système de fichiers d'une carte SIM (cf. Figure 3) possède un répertoire racine **3F00**.

Les deux sous-répertoires les plus importants sont le répertoire **GSM (DF-GSM, 7F20)** et le répertoire **TELECOM (DF-TELECOM, 7F10)**.



3 Le système de fichiers de la SIM

Le répertoire GSM abrite les fichiers IMSI (EF-IMSI, 6F07), le fichier LOCI (EF-LOCI, 6F7E) qui stocke les paramètres TIMSI et LAI évoqués précédemment, ainsi que la clé Kc (EF-Kc, 6F20). Bien entendu, il existe de nombreuses autres informations que nous ne décrivons pas ici par souci de concision. Par exemple, le fichier SIM Service Table (EF-SST, 6F38) dresse la liste des services disponibles dans la carte. Nous encourageons le lecteur curieux à télécharger gratuitement la norme GSM 11.11 sur le site de l'ETSI (www.etsi.org).

Le répertoire TELECOM comporte trois fichiers qui retiennent plus particulièrement notre attention, un annuaire de numéros abrégés (EF-ADN, 6F3A), un annuaire téléphonique (EF-FDN, 6F3B) et la liste des SMS émis ou reçus (EF-SMS, 6F3C). Ces fichiers sont accessibles en lecture et en écriture et sont de surcroît protégés par le code PIN utilisateur. Ils offrent donc un espace mémoire sécurisé pour des applications diverses.

L'accès aux fichiers se divise en cinq classes de priorités, ALWAYS, CHV1, CHV2, ADM et NEVER :

- ⇒ le fichier est toujours accessible (ALWAYS, code 0) ;
- ⇒ le fichier est protégé par le code PIN du porteur (CHV1, code 1) ;
- ⇒ le fichier est protégé par le code PIN de l'émetteur de la SIM (CHV2, code 2) ;
- ⇒ le fichier est géré par une autorité administrative (ADM, codes de 4 à E) ;
- ⇒ le fichier n'est jamais accessible (NEVER, code F).

5. Les commandes APDU

L'interface logique d'une SIM est décrite par la norme GSM 11.11 publiée par l'ETSI. Elle comporte (cf. Figure 4, page suivante) 22 commandes (ou APDU) que nous classons schématiquement en quatre groupes :

⇒ Accès aux différents fichiers stockés dans la carte (6 commandes). De manière analogue aux ordinateurs personnels, l'information enregistrée dans la SIM est ordonnée grâce à des répertoires et des fichiers. La commande SELECT

permet de naviguer dans cette arborescence (c'est l'équivalent de l'instruction CD en ligne de commande). Les commandes READ et WRITE réalisent les opérations de lecture et d'écriture.

- ⇒ Opérations liées au code PIN (5 commandes), telles que vérification, modification, activation, suppression ou déblocage à l'aide du code PUK.
- ⇒ Exécution de l'algorithme A3A8, grâce à la commande RUN-GSM-ALGORITHM.

COMMAND	INS	P1	P2	P3
SELECT	'A4'	'00'	'00'	'02'
STATUS	'F2'	'00'	'00'	lgth
READ BINARY	'B0'	offset high	offset low	lgth
UPDATE BINARY	'D6'	rec No.	mode	lgth
READ RECORD	'B2'	rec No.	mode	lgth
UPDATE RECORD	'DC'	rec No.	mode	lgth
SEEK	'A2'	'00'	type/mode	lgth
INCREASE	'32'	'00'	'00'	'03'
VERIFY CHV	'20'	'00'	CHV No.	'08'
CHANGE CHV	'24'	'00'	CHV No.	'10'
DISABLE CHV	'26'	'00'	'01'	'08'
ENABLE CHV	'28'	'00'	'01'	'08'
UNBLOCK CHV	'2C'	'00'		'10'
INVALIDATE	'04'	'00'	'00'	'00'
REHABILITATE	'44'	'00'	'00'	'00'
RUN GSM ALGORITHM	'88'	'00'	'00'	'10'
SLEEP	'FA'	'00'	'00'	'00'
GET RESPONSE	'C0'	'00'	'00'	lgth
TERMINAL PROFILE	'10'	'00'	'00'	lgth
ENVELOPE	'C2'	'00'	'00'	lgth
FETCH	'12'	'00'	'00'	lgth
TERMINAL RESPONSE	'14'	'00'	'00'	lgth

4 Liste des commandes APDU d'une carte SIM

⇒ Différentes opérations (10 commandes) utilisées à des fins diverses. En particulier, le modèle SIM Tool Kit permet à un programme stocké et exécuté dans la SIM (un jeu par exemple) d'avoir accès au clavier et à l'écran du mobile, mais également de communiquer avec le monde extérieur via des messages SMS.

Dans cet article, nous présentons seulement les modes d'emploi des commandes les plus usuelles, c'est-à-dire **SELECT**, **READ BINARY**, **UPDATE BINARY**, **READ RECORD**, **UPDATE RECORD**, **VERIFY CHV**, **CHANGE CHV**, **DISABLE CHV**, **ENABLE CHV**, **UNBLOCK CHV**, **RUN GSM ALGORITHM**, **GET RESPONSE**. Le lecteur désirant plus de détails pourra consulter la norme GSM 11.11.

6. Les mots de statut

Les deux derniers octets de la réponse à une requête APDU (SW1, SW2) indiquent le statut de l'opération demandée. Une valeur **90 00** signifie un succès, le mot **9F XX** est une variante du cas précédent, mais précise qu'il est nécessaire de collecter les **XX** octets du résultat. Cette lecture supplémentaire est accomplie à l'aide de la commande **GET RESPONSE (A0 C0 00 00 XX)**.

Par exemple, la requête de sélection du répertoire **GSM SELECT(DF-GSM)** s'exprime sous la forme :

```
>> A0 A4 00 00 02 7F 20
<< 9F 1A
```

La réponse de la carte ne comporte que les deux octets de statut (**9F 1A**) qui indiquent la présence d'un résultat de 26 octets (**1A**).

La commande **GET-RESPONSE** collecte les 26 octets disponibles.

```
>> A0 C0 00 00 1A
```

On obtient une suite d'octets, les deux derniers (**9000**) notifiant le succès de cette opération.

```
<< 00 00 04 7A 7F 20 02 00 00 00 00 00 00 00 11 00 12 08 00
83 8A 83 8A 00 00 83 83 90 00
```

Un octet de statut SW1 autre que **90** ou **9F** est en règle générale la conséquence d'une erreur ou d'un problème particulier.

7. La commande Select

Cette commande est très employée, puisqu'elle permet de naviguer à travers l'arbre des répertoires et fichiers embarqués dans la SIM. Elle s'exprime sous la forme **A0 A4 00 00 02 FH FL**, où **FH FL** représente le nom d'un fichier codé sur deux octets, par exemple **A0 A4 00 00 02 7F 20** sélectionne le répertoire **GSM (7F20** ou **DF-GSM)** ou **A0 A4 00 00 02 3F 00** active le répertoire racine (**3F00** ou **Master File**).

Les identifiants des fichiers sont attribués selon des règles hiérarchisées : le premier octet (**FH**) des répertoires de premier

niveau est fixé à **3F** et **5F** pour le deuxième niveau ; cette valeur est fixée respectivement à **6F** et **4F** pour les fichiers des répertoires de premier et deuxième niveau.

La réponse à une opération de sélection est un en-tête qui comporte des informations diverses relatives aux restrictions d'accès (faut-il présenter un code PIN avant une opération de lecture écriture ?), au nombre de fichiers stockés dans un répertoire **DF** ou à la taille d'un fichier élémentaire **EF**.

8. Sélection d'un répertoire

L'exemple ci-dessous illustre la sélection du répertoire **DF-GSM** :

```
>> A0A4 0000 02 7F20
<< 9F 1A
>> A0C0 0000 1A
<< 00 00 04 7A 7F 20 02 00 00 00 00 00 00 00 11 00 12 08 00 83 8A 83 8A 00 00 83 83 90 00
```

Nous obtenons une réponse dont les deux derniers octets de statut (**90 00**) précisent le succès de cette requête. Les autres octets délivrent une série d'indications, nous informant en particulier que nous devons présenter notre PIN code, et que 3 essais sont possibles. L'analyse détaillée de cet en-tête est la suivante :

00 00, octets sans signification
 04 7A, la taille mémoire non utilisée (1146 octets)
 7F 20, le nom du répertoire
 02, le type du répertoire (MF)
 00 00 00 00 00, sans signification
 0D, 13 octets de données GSM
 11, (00010001) le bit de poids fort codé à zéro implique la présentation d'un code PIN
 00, nombre de sous-répertoires
 12, nombre de fichiers (18)
 08 00, octets non commentés
 83, code CHV1 (PIN utilisateur) initialisé, 3 essais PIN1 possibles
 8A, code initialisé, 10 essais PUK1 possibles
 83, code CHV2 (PIN administrateur) initialisé, 3 essais PIN2 possibles

8A, code initialisé, 10 essais PUK2 possibles
 00 00 83 83, octets non commentés.

Afin d'accéder aux fichiers stockés dans le répertoire, nous présentons notre code PIN (soit une série de 4 zéros), à l'aide de la commande **VERIFY**. Dans cette dernière, le code PIN est codé sous forme de caractères ASCII (30 étant associé au caractère 0) ; bien que le code PIN soit de 8 chiffres au maximum, seulement les quatre premiers digits sont analysés, les autres (codés par FF) ne sont pas pris en compte.

```
>> A0 20 00 01 08 30 30 30 30 FF FF FF FF
<< 90 00
```

En cas d'erreur (par exemple présentation du code 1000), on obtient typiquement deux mots de statut d'erreur 9804 (pour *authentication failed*).

9. Sélection d'un fichier

Lorsqu'un téléphone portable est mis en tension, il sélectionne le répertoire **GSM**, détecte si la présentation d'un code PIN est requise, et, le cas échéant, présente à la carte SIM cette valeur via la commande **VERIFY**. Par la suite, il lit le contenu du fichier **EF-Phase** (6FAE) qui contient le numéro de version fonctionnel de la carte. Cette opération est illustrée ci-dessous :

```
>> A0 A4 00 00 02 6F AE
<< 9F 0F
>> A0 C0 00 00 0F
<< 00 00 00 01 6F AE 04 00 0B 00 BB 01 02 00 00 90 00
```

La réponse se termine par deux octets de statut 90 00, indiquant le succès de la requête.

L'interprétation commentée de l'en-tête du fichier est la suivante :

00 00, octets sans signification

00 01, la taille du fichier, soit un octet
 6F AE, le nom du fichier
 04, le type du fichier (EF)
 00, octet non commenté
 0B, 00, BB, les conditions d'accès
 01, le statut du fichier
 02, la longueur des données résiduelles
 00 00, octets non commentés

Au terme de cette analyse, nous déduisons la taille du fichier (1 octet) et nous lisons son contenu à l'aide de la commande **READ BINARY**.

```
>> A0 B0 00 00 01
<< 03 90 00
```

La valeur 3 précise une carte de type phase2+.

10. Lecture de l'IMSI

Le fichier **EF-IMSI** (6F07), de type transparent, est localisé dans le répertoire **GSM**. On obtient sa taille grâce à l'en-tête obtenu après sa sélection (voir code ci-contre).

Dans notre exemple, l'IMSI possède 9 octets (08 29 80 10 36 60 04 81 80).

```
>> A0 A4 00 00 02 6F 07
<< 9F 0F
>> A0 C0 00 00 0F
>> 00 00 00 09 6F 07 04 00 18 00 18 01 02 00 00 90 00
<< A0 B0 00 00 09
>> 08 29 80 10 36 60 04 81 80 90 00
```

11. Lecture des paramètres TMSI et LAI

Le fichier **EF-LOCI** (6F 7E) contient une liste d'attributs (TMSI 4 octets, LAI 5 octets, TMSI-TIME 1 octet et Location-Update-Status 1 octet), soit une taille globale de onze octets (voir code ci-contre).

Dans notre cas, TMSI a pour valeur 78 08 8F 48, et LAI 02 F8 10 38 00.

```
>> A0 A4 00 00 02 6F 7E
<< 9F 0F
>> A0 C0 00 00 0F
<< 00 00 00 0B 6F 7E 04 00 11 FF 15 01 02 00 00 90 00
>> A0 B0 00 00 0B
<< 78 08 8F 48 02 F8 10 38 00 00 90 00
```

12. Exécution de l'algorithme d'authentification du GSM

La commande **RUN-GSM-ALGO (INS=88)** exécute la fonction **A3/A8** du GSM. Son argument d'entrée est un nombre aléatoire de 16 octets (**RAND**). Elle retourne une liste de deux valeurs : la signature **SRES** (4 octets) et la clé **Kc** (8 octets).

```
>> A0 88 00 00 10 01 23 45 67 89 AB CE DF 01 23 45 67 89 0A BC DE
<< 9F 0C
>> A0 C0 00 00 0C
<< 11 D2 60 F1 67 F0 E8 FE F2 BC 60 00 90 00
```

Dans l'illustration ci-dessus, **RAND=0123456789ABCDEF01234567890ABCDE**, **SRES=11D260F1** et **Kc=67F0E8FEF2BC6000**. On remarque que les douze derniers bits de la clé **Kc** sont égaux à zéro (et donc que ce paramètre ne comporte que 52 bits utiles).

13. Mise à jour du fichier EF-Kc

La carte SIM ne met pas à jour automatiquement le fichier **EF-Kc (6F20)** au terme de l'exécution de la procédure **RUN-GSM-ALGO**. Cette opération est réalisée par le mobile grâce à la commande **UPDATE-BINARY (INS=D6)**. La valeur stockée dans le fichier **EF-Kc** est une liste de deux valeurs, la clé **Kc** et un octet de validation (**00** lorsque le contenu est valide et **07** dans le cas contraire).

```
>> A0 A4 00 00 02 6F 20
<< 9F 0F
>> A0 C0 00 00 0F
<< 00 00 00 09 6F 20 04 00 11 00 BB 01 02 00 00 90 00
<< A0 B0 00 00 09
>> FF FF FF FF FF FF FF FF 07 90 00
>> A0 D6 00 00 09 01 23 45 67 89 AB CD EF 00
<< 90 00
>> A0 B0 00 00 09
<< 01 23 45 67 89 AB CD EF 00 90 00
```

14. Lecture de la table des services SIM (EF-SIM-Service-Table)

Le fichier **EF-SIM-Service-Table (6F 38)** est la liste des services offerts par la carte SIM. Chaque service, identifié par son ordre d'apparition (n°1, n°2,...) est associé à deux bits : le premier (poids fort) renseigne l'existence du service (1 pour présent), le deuxième (poids faible) indique son activation (1 pour actif).

```
>> A0 A4 00 00 02 6F 38
>> A0 C0 00 00 0F
<< 00 00 00 0D 6F 38 04 00 1B 00 BB 01 02 00 00 90 00
>> A0 B0 00 00 0D
<< FF 3C CF 3F 03 00 3C 03 00 0C 00 00 00 90 00
```

Pour notre SIM, les services 1, 2, 3, et 4 sont présents et actifs (**FF = 11.11.11.11**). Les services 5 et 8 sont absents et inactifs, puisque le deuxième octet a pour valeur **3C**, soit **00.11.11.00**.

Le service n°1 permet la désactivation du code PIN utilisateur (**CHV1**). Le deuxième notifie la présence d'un annuaire de numéros abrégés (**ADN**). Le troisième est un annuaire de numéros non abrégés (**FDN**). Enfin, le quatrième indique l'existence du fichier **EF-SMS** réalisant le stockage des SMS.

Les fichiers **EF-ADN**, **EF-FDN** et **EF-SMS** occupent un espace mémoire de l'ordre de plusieurs Ko. Ils sont localisés dans le répertoire **DF-TELECOM (7F10)**, sélectionné à l'aide de la commande qui suit :

```
>> A0 A4 00 00 02 7F 10
<< 9F 1A
>> A0 C0 00 00 1A
<< 00 00 04 7A 7F 10 02 00 00 00 00 00 00 11 00 09 08 00 83 8A 83 8A 00
00 83 83 90 00
```

15. Lecture et écriture des SMS dans la SIM

Le fichier **EF-SMS (6F3C)** est un fichier cyclique, c'est-à-dire qu'il se comporte comme une liste d'enregistrements de taille fixe, identifiés par un index.

```
>> A0 A4 00 00 02 6F 3C
<< 9F 0F
<< A0 C0 00 00 0F
>> 00 00 14 A0 6F 3C 04 00 11 00 BB 01 02 01 B0 90 00
```

En analysant l'en-tête du fichier, on obtient sa longueur totale globale (**14A0** soit 5280 octets), ainsi que la taille d'un enregistrement (**B0** soit 176 octets). On en déduit qu'il se divise en 30 enregistrements (puisque $5280/176 = 30$).

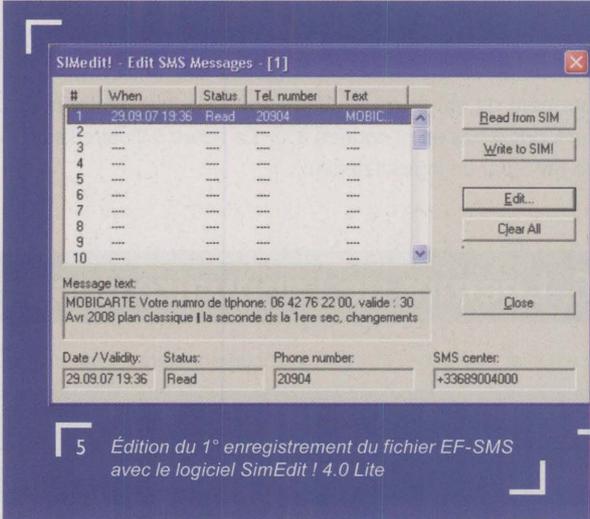
Les conditions d'accès (octets 9,10, 11) sont les suivantes :

- ⇒ Les commandes **READ**, **SEEK** et **UPDATE** sont contrôlées par **CHV1**.
- ⇒ Les commandes **REHABILITATE** et **INVALIDATE** sont contrôlées par **ADM**.

La commande **READ-RECORD**, soit **A0 B2 01 04 B0** lit un enregistrement dont l'index est **P1** (la valeur **P1=01** doit être utilisée pour lire le premier enregistrement). Dans notre exemple, **P1** sera compris entre **01** et **1E** (**30** en notation décimale). La commande décrite ci-dessous réalise la lecture du premier (**01**) enregistrement :

```
>> A0 B2 01 04 B0
>> 01 07 91 33 86 09 40 00 F0 04 05 85 02 09 F4 00 F1 70 90 92 91 63 90
80 96 CD A7 30 39 0C 4A A9 45 90 F5 4D 97 97 41 EE 7A BB 20 7F 83 C8 65
10 BD C0 2E C0 D1 6F 77 59 07 82 D9 40 34 19 E8 66 03 C9 64 20 18 8C 05
B2 87 D9 69 72 19 A4 03 CD 60 A0 A0 5D 0E 92 C1 60 38 10 9C 1D 76 83 C6
EC F0 7C 9E 8E D7 CB A0 3F 88 1D 06 CD CB E3 B7 9B 5C 06 91 09 73 10 3B
8C 0A 95 E5 65 D0 BC 3C 66 81 C6 E8 B0 FB 5C 6E 97 DD F4 39 E8 2C 0F D3
EB 69 FA 1C 44 2E 83 E0 EC B0 5B 07 92 01 FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF 90 00
```

- 01** indication de présence d'un SMS
- 07** la longueur de l'attribut **SMSC information** (7 octets)
 - 91 33 86 09 40 00 F0**, **SMSC information**, le numéro du centre SMS
- 04** le premier octet d'un message SMS deliver
- 05** la longueur du numéro de l'émetteur du SMS (5 digits)
- 85** type de numéro de l'émetteur **1000 0101**
 - 02 09 F4**, numéro de l'émetteur
- 00** l'octet **TP-PID (Protocol Identifier)**.
- F1** l'octet **TP-DCS (Data Coding Scheme)**



5 Édition du 1^{er} enregistrement du fichier EF-SMS avec le logiciel SimEdit ! 4.0 Lite

70 90 92 91 63 90 80 TP-SCTS, (*TimeStamp*) l'heure d'émission

96 TP-UDL (*User Data Length*), la longueur du message (150 octets).

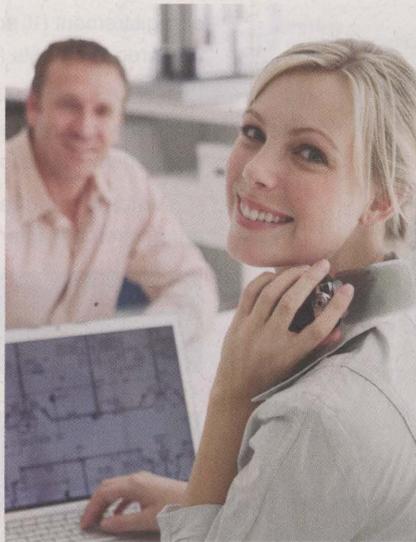
On trouvera à l'adresse <http://www.dreamfabric.com/sms/> une description plus précise du format des SMS. Le logiciel gratuit SIMedit ! (daté des années 2001) réalise l'édition des fichiers EF-SMS, EF-ADN et EF-FDN.



En collaboration avec les Editions DIAMOND, Cardelya, partenaire de Gemalto, offre la possibilité d'acquérir des KITS de cartes à puce permettant à son lectorat la mise en oeuvre des outils de programmation présentés.



Cardelya™ propose des solutions globales autour de la carte à puce allant de la fabrication des cartes en petites séries jusqu'à leur déploiement aux utilisateurs finaux en passant par la personnalisation électrique et graphique. D'une expérience de plus de dix années dans les applications à base de cartes à puce, Cardelya™ vous aide à personnaliser, utiliser et déployer des cartes à puce pour tous les usages de celles-ci dans une organisation (contrôle d'accès physique, contrôle d'accès logique, fidélité, paiement, télé-déclaration, etc., ...). Grâce à son positionnement original (industriel et intellectuel), Cardelya™ peut fournir tous les produits et services rattachés à une application carte (lecteurs avec et sans contact, cartes à puce avec et sans contact, cartes multi-technologies, ateliers de personnalisation électrique et graphique, service bureau de personnalisation, ...).



www.gemalto.com

Leader de la sécurité numérique

Des solutions d'authentification forte pour sécuriser votre entreprise

Gemalto propose des solutions de sécurité numérique intégrées. Nos produits et services sont utilisés par plus d'un milliard de personnes à travers le monde pour diverses applications, notamment dans les télécommunications, les services financiers, les administrations, la gestion des identités, le contenu multimédia, la gestion des droits numériques, la sécurité informatique et les transports en commun.

Nos solutions d'authentification basées sur les cartes à puce présentent de nombreux avantages, notamment des capacités de stockage de données, des performances de traitement, une portabilité et une grande simplicité d'utilisation.



Cette offre spéciale est disponible sur

<http://www.cardelya.fr/hscarte/>

Code d'accès **VGOCPB7491LPJ**

Le premier octet d'un enregistrement précise la validité des données, **00** indique typiquement l'absence de SMS (plus précisément, le bit de poids faible de cet octet). Le SMS proprement dit est contenu dans les 175 octets restants, les valeurs inutilisées étant codées à **FF**. La figure 5 illustre l'édition de ce SMS par un logiciel dédié.

La commande **UPDATE-RECORD** réalise l'écriture d'un enregistrement entier dans le fichier **EF-SMS**. Elle est identifiée par l'attribut **INS=DC** et le champ **P1** spécifie le numéro d'enregistrement, par exemple :

```
>> A0 DC 01 04 00 [176 octets]
<< 90 00
```

16. Lecture de l'annuaire des numéros, ADN

Le fichier cyclique **EF-ADN (6F 3A)** est un annuaire téléphonique.

```
>> A0 A4 00 00 02 6F 3A
<< 9F 0F
>> A0 C0 00 00 0F
<< 00 00 1B 58 6F 3A 04 00 11 00 22 01 02 01 1C 90 00
```

En analysant l'en-tête du fichier, on obtient sa longueur totale globale (**1B58** soit 7000 octets), ainsi que la taille d'un enregistrement (**1C** soit 28 octets). On en déduit qu'il se divise en 250 enregistrements (puisque $250 = 7000/28$).

Chaque numéro de téléphone est associé à une étiquette (*Apha Tagging*) dont la longueur est obtenue en soustrayant 14 à la taille d'un enregistrement (soit $14 = 28 - 14$). Par défaut, cette étiquette est codée selon l'alphabet SMS, c'est-à-dire avec un bit de poids fort à zéro.

Les commandes **READ-RECORD** et **UPDATE-RECORD** réalisent respectivement les opérations d'écriture et de lecture des enregistrements. L'exemple ci-dessous illustre une lecture du premier enregistrement.

```
>> A0 B2 01 04 1C
<< 11 53 65 72 76 2E 20 43 6C 69 65 6E 74 FF 03 81 23 53 FF FF FF FF FF
FF FF FF FF FF 90 00
```

Le contenu de l'enregistrement s'interprète de la manière suivante :

11 53 65 72 76 2E 20 43 6C 69 65 6E 74 FF, « Serv. Client », une étiquette de 14 octets associée (cadrée à gauche) au numéro de téléphone ;

03, longueur (en octets) des attributs **TON/NPI** et numéro de téléphone ($3 = 1 + 2$) ;

81, type de numéro de téléphone (**TON/NPI**), **81** est une valeur typique pour le GSM ;

23 53, numéro de téléphone ;

FF (*Capability/Configuration Identifier*) un attribut non utilisé ;

FF (*Extension1 Record Identifier*) un attribut non utilisé.

Les octets restants sont sans signification.

Un numéro de téléphone comporte typiquement 10 chiffres (soit 5 octets), ainsi **06 82 12 34 56** sera inscrit dans l'annuaire sous la forme **06 81 60 28 21 43 65**.

L'analyse de cet enregistrement est illustré par la figure 6.

Lecture de l'annuaire de service, **FDN**.

Le fichier cyclique **EF-FDN (6F 3B)** est un annuaire des services.

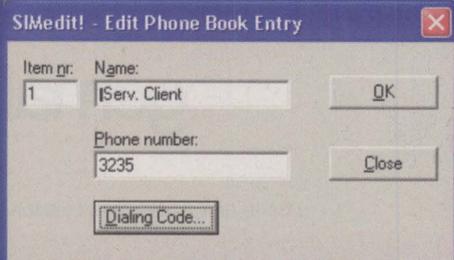
```
>> A0 A4 00 00 02 6F 3B
<< 9F 0F
>> A0 C0 00 00 0F
<< 00 00 03 48 6F 3B 04 00 12 00 BB 01 02 01 1C 90 00
```

En analysant l'en-tête du fichier, on obtient sa longueur totale globale (**0348** soit 840 octets), ainsi que la taille d'un enregistrement (**1C** soit 28 octets). On en déduit qu'il se divise en 30 enregistrements (puisque $30 = 840/28$).

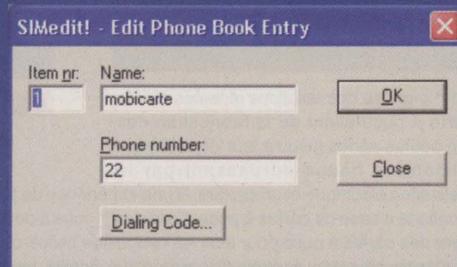
Les commandes **READ-RECORD** et **UPDATE-RECORD** réalisent respectivement les opérations d'écriture et de lecture des enregistrements. Le format de ce fichier est identique à celui de **EF-ADN**.

Voici le résultat obtenu lors de la lecture du premier enregistrement, interprété par la figure 7.

```
>> A0 B2 01 04 1C
<< 6D 6F 62 69 63 61 72 74 65 FF FF FF FF FF 02 81 22 FF FF FF FF FF FF
FF FF FF FF 90 00
```



6 Édition du 1^{er} enregistrement du fichier **EF-ADN** avec le logiciel **SimEdit ! 4.0 Lite**



7 Édition du 1^{er} enregistrement du fichier **EF-ADN** avec le logiciel **SimEdit ! 4.0 Lite**

17. Opérations liées aux codes PIN

Un code PIN comporte huit octets, représentant des chiffres au format ASCII (0->30, 1->31, ..., 9->39). Les octets sans signification sont codés par FF.

Nous avons préalablement décrit la commande **VERIFY CHV** qui présente de code PIN (**CHV1**) du propriétaire du mobile à la carte SIM. Elle est codée sous la forme **A0 20 00 P2 08 [PIN]**, P2 prenant la valeur **01** dans le cas de **CHV1** (le PIN de l'utilisateur) et **02** pour **CHV2**.

```
>> A0 20 00 01 08 30 30 30 30 FF FF FF FF
<< 90 00
```

L'usage du PIN utilisateur est annulé grâce à la commande **DISABLE PIN** (**A0 26 00 01 08 [PIN]**). La commande **ENABLE PIN** (**A0 28 00 01 08 [PIN]**) réalise l'opération inverse. Par exemple :

Pour une activation du code PIN :

```
>> A0 28 00 01 08 30 30 30 30 FF FF FF FF
<< 90 00
```

Pour une annulation du code PIN :

```
>> A0 26 00 01 08 30 30 30 30 FF FF FF FF
<< 90 00
```

La modification du PIN est possible grâce à **CHANGE CHV** (**A0 24 00 01 10 [Ancien PIN] [Nouveau PIN]**).

```
>> A0 24 00 01 10 30 30 30 30 FF FF FF FF 31 32 33 34 FF FF FF FF
<< 90 00
```

Au terme de trois essais infructueux de présentation du code PIN utilisateur (**CHV1**) la carte est bloquée. La commande **UNBLOCK CHV** (**A0 2C 00 01 10 [PUK] [PIN]**) annule cet état et permet au module d'être à nouveau opérationnel. Le PUK est un code unique de huit chiffres associé à la SIM.

```
>> A0 2C 00 01 10 31 32 33 34 35 36 37 38 30 30 30 30 FF FF FF FF
<< 90 00
```

En guise de conclusion

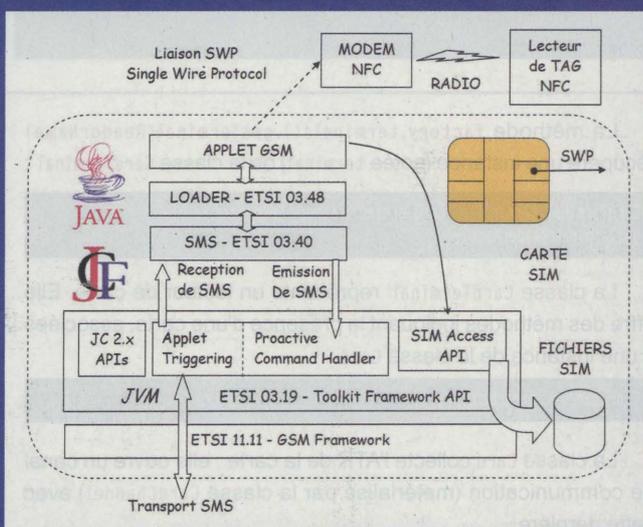
Dans cet article nous avons décrit les principaux usages d'une carte SIM en suivant une démarche pragmatique. D'autres fonctionnalités, parfois qualifiées d'écosystème de la SIM, sont disponibles ; elles sont cependant contrôlées par les opérateurs de téléphonie mobile, et, par conséquent, ne permettent pas une mise en œuvre pratique.

De nombreuses cartes intègrent une machine virtuelle JAVA (JVM) et donc exécutent des applications écrites en JAVA, plus précisément le JavaCard qui est un sous-ensemble de ce langage de programmation.

L'ETSI a enrichi cet environnement grâce à la norme 03.19 qui permet aux applets GSM (logés dans la SIM), d'accéder aux fichiers de la carte et d'interagir avec le clavier et l'écran du mobile. Un apport fondamental est la capacité offerte aux applications embarquées d'envoyer et de recevoir des messages SMS (dont le format est plus précisément décrit par le standard ETSI 03.40). Ces derniers sont généralement chiffrés selon des mécanismes décrits par la norme ETSI 03.48. Ils transportent typiquement des commandes APDU, interprétées par l'applet GSM et donc capables d'interagir avec le système de fichiers de la SIM.

L'écosystème actuel des SIM (cf. Figure 8) permet donc de modifier à distance leur contenu, voire de télécharger de nouvelles applications. Cette technologie est usuellement qualifiée de gestion OTA (*Over The Air*). Elle s'appuie sur des serveurs dédiés gérant une flotte de SIM contrôlées par des opérateurs.

Cet écosystème va s'enrichir avec une nouvelle famille d'applets GSM, les applications NFC. Le NFC (*Near Field Communication*) est un modem radio de courte portée (10 centimètres au plus) qui va progressivement être disponible dans les téléphones mobiles. Grâce à un contact spécifique la carte SIM contrôle ce modem selon le protocole SWP (*Single Wire Protocol*) en cours de normalisation à l'ETSI. Un service typique de cet écosystème est le téléchargement d'une application carte Navigo dans une carte SIM, qui pourra communiquer avec les portiques du métro parisien via la puce NFC.



8 Écosystème de la carte SIM

On peut se poser la question de la survie de la carte SIM dans un monde dit de convergence, où le réseau Internet s'enrichit chaque jour de nouvelles applications. Il existe des axes de recherches relatifs à cette thématique, par exemple une *spin-off*

(www.ethertrust.com) issue de deux acteurs académiques majeurs (Telecom ParisTech et le LIP6 de l'université Pierre et Marie Curie) développe des technologies de cartes à puce dédiées aux environnements émergents tout IP. ■



Références

- ⇒ <http://www.gsm-security.net>
- ⇒ VEDDER (Klaus), « SM: Security, Services, and the SIM », *State of the Art in Applied Cryptography* 1997, pp. 224-240.
- ⇒ VEDDER (Klaus), WEIKMANN (Franz), « Smart Cards – Requirements, Properties, and Applications », *State of the Art in Applied Cryptography* 1997, pp. 307-331.
- ⇒ BRICENO (Marc), GOLDBERG (Ian), WAGNER (David), « GSM Cloning », <http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html>
- ⇒ BIRYUKOV (Alex), SHAMIR (Adi), « Real Time Cryptanalysis of the Alleged A5/1 on a PC », décembre 1999.
- ⇒ RAO (Josyula R.), ROHATGI (Pankaj), SCHERZER (Helmut), TINGUELY (Stephane), « Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards », *Proceedings of the 2002 IEEE Symposium on Security and Privacy*.
- ⇒ BARKAN (Elad), BIHAM (Eli), KELLER (Nathan), « Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication », *Proceedings of Crypto 2003*.
- ⇒ VEDDER (Klaus), « Smart Card Platform 2006 », http://www.etsi.org/WebSite/document/Workshop/Security2006/Security2006S1_3_Klaus_Vedder.pdf
- ⇒ VEDDER (Klaus), « Smart Card Platform 2007 », http://www.etsi.org/WebSite/document/Workshop/Security2007/Security2007S5_4_Klaus_Vedder.pdf
- ⇒ GSM 02.09, « Digital cellular telecommunications system (Phase 2); Security aspects », www.etsi.org
- ⇒ GSM 02.17, « Digital cellular telecommunications system (Phase 2); Subscriber Identity Modules (SIM), Functional characteristics », www.etsi.org
- ⇒ GSM 03.20, « Digital cellular telecommunications system (Phase 2); Security related network functions », www.etsi.org
- ⇒ GSM 11.11, « Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module-Mobile Equipment (SIMME) interface », www.etsi.org
- ⇒ GSM 11.14, « Digital cellular telecommunications system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module-Mobile Equipment (SIM-ME) interface », www.etsi.org
- ⇒ SUN, « Package javax.smartcardio, Java™ Smart Card I/O API », <http://java.sun.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html>
- ⇒ COMPELSON Laboratories, *SIM Edit! 4.0 lite*, 2001, www.compelson.com



Annexe

Un lecteur de carte SIM écrit en JAVA 1.6

Disponible à partir de JAVA 1.6, le paquetage `javax.smartcardio` permet d'accéder à des lecteurs de cartes à puce.

La classe `TerminalFactory` sélectionne une implémentation logicielle (typiquement PC/SC, la norme définie par Microsoft pour installer des lecteurs sur des machines Windows). La méthode `TerminalFactory.getDefault()` obtient une instance d'un objet `TerminalFactory`, notée `factory` :

```
TerminalFactory factory = TerminalFactory.getDefault();
```

Il est possible de collecter la liste des lecteurs de carte présents sur le PC, mais cette option ne fonctionne pas avec toutes les versions de JAVA.

La méthode `factory.terminals().getTerminal(ReaderName)` récupère une instance (notée `terminal`) de la classe `CardTerminal` :

```
CardTerminal terminal = (CardTerminal)factory.terminals().getTerminal(ReaderName);
```

La classe `CardTerminal` représente un lecteur de carte. Elle offre des méthodes indiquant la présence d'une carte, associées à une instance de la classe `Card` :

```
Card card = terminal.connect("*");
```

La classe `Card` collecte l'ATR de la carte ; elle ouvre un canal de communication (matérialisé par la classe `CardChannel`) avec cette dernière,

```
CardChannel channel = card.getBasicChannel();
```

La classe `CommandAPDU` construit une commande APDU à partir des informations que l'on désire transmettre à la carte, usuellement stockées dans un tableau d'octets (`byte [] req`). Plusieurs constructeurs sont disponibles en fonction du type de commande :

```
CommandAPDU request = new CommandAPDU (req) ;
```

La méthode `transmit()` s'applique à une requête `CommandAPDU` et retourne une réponse `ResponseAPDU`.

```
ResponseAPDU response = channel.transmit(request)
```

La méthode `getBytes()` transforme la réponse en une suite d'octets.

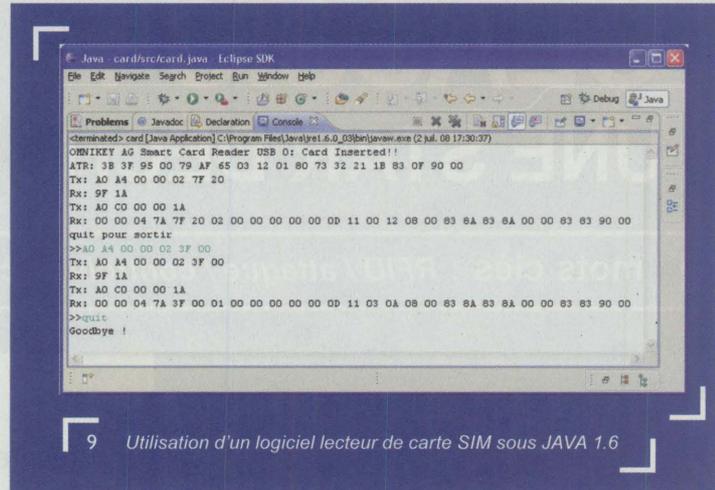
```
byte resp[] = response.getBytes()
```

Dans l'exemple de code source que nous produisons en annexe, la commande envoyée vers la carte SIM est simplement exprimée sous forme d'une chaîne de caractère représentant une suite d'octets en notation hexadécimale. La réponse est un tableau d'octets, incluant les deux derniers octets de statut (`SW1` et `SW2`). Par exemple, la chaîne `A0A4 0000 02 7F20` représente la sélection du répertoire GSM (`SELECT(DF_GSM)`). L'invocation de la méthode

```
byte [] resp = EchoAPDU("A0A4 0000 02 7F20");
```

provoque l'affichage des éléments suivants :

```
Tx: A0 A4 00 00 02 7F 20
Rx: 9F 1A
Tx: A0 C0 00 00 1A
Rx: 00 00 04 7A 7F 20 02 00 00 00 00 00 11 00 12 08 00 83 8A 83 8A 00 00 83 83 90 00
```



Remarque: les fichiers sont disponibles sur : www.miscmag.com/hscarte/

MASTÈRE SPÉCIALISÉ

SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

www.esiea.fr/ms-sis

DU CODE AU RESEAU

-  Réseaux
-  Modèles et Politiques de sécurité
-  Cryptologie pour la sécurité
-  Sécurité des réseaux, des systèmes et des applications

DEVENEZ LES **SPECIALISTES DE LA SECURITE** QUE LES ENTREPRISES ATTENDENT

- Un groupe d'enseignants composé d'une cinquantaine d'**experts en sécurité**
- Des étudiants **acteurs de leur formation**
- Une formation **intensive** : 510 heures de cours et plus de 250 heures de projets
- Un fort soutien de l'**environnement industriel**



Accrédité par la Conférence des Grandes Ecoles

RENTREE **OCTOBRE 2009**



NXP MIFARE CLASSIC : UNE STAR DÉCHUE

■ mots clés : RFID / attaque / contrôle d'accès



Frénésie, le mot n'est pas trop fort pour décrire l'engouement actuel envers le sans contact. Plus question désormais de se torturer l'esprit pour comprendre le pictogramme décrivant la façon d'insérer une carte à puce dans un lecteur ; il suffit de passer sa carte à proximité de celui-ci et le tour est joué. La star de toutes ces cartes sans contact, la figure de proue du domaine, c'est la NXP Mifare Classic (une icône vendue à plus d'un milliard d'exemplaires¹). Dans cet article, nous présentons comment elle fut lapidée au printemps 2008, lorsque plusieurs équipes de recherche ont mis au grand jour des failles de sécurité aussi sérieuses qu'impardonnables. La star déchue entraîne dans sa débâcle des millions de systèmes de contrôle d'accès, de billetterie et de porte-monnaie électronique.

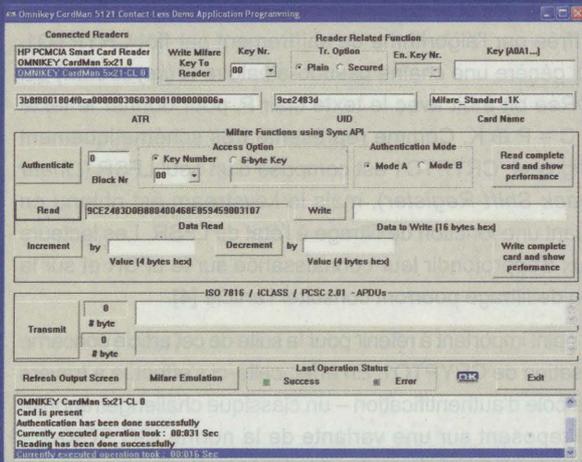
C'est en 1994 que Philips Semiconducteurs, devenu aujourd'hui NXP Semiconducteurs, propose « Mifare », une dénomination commerciale qui allait regrouper une large gamme de cartes à puce sans contact. Répondant tous au standard ISO14443, les produits de la gamme Mifare sont généralement considérés comme une technologie RFID (*Radio Frequency Identification* [5]), à laquelle *MISC* a consacré un dossier spécial dans son numéro 33.

La « Mifare Classic » est le produit phare de la gamme et présente la particularité d'utiliser un algorithme de chiffrement par flot, propriétaire et tenu secret, CRYPTO1, qui peut être implémenté à moindre coût en logique câblée. Le principe général de la carte Mifare Classic est simple : il s'agit d'une mémoire EEPROM dont l'accès en lecture et en écriture est protégé par un protocole d'authentification de type challenge/réponse reposant sur CRYPTO1. Cet algorithme n'étant pas public, le lecteur doit posséder un ASIC (*Application-Specific Integrated Circuit*) spécifique, par exemple le RC632, pour communiquer avec une Mifare Classic ; il doit également connaître la ou les clés qui lui autorisent l'accès à la carte. Outre l'authentification, CRYPTO1 est utilisé pour chiffrer les informations échangées de la carte vers le lecteur, et vice versa.

En se procurant une carte Mifare Classic vierge (quelques dizaines de centimes d'euros²), un lecteur compatible (environ 100 euros) et un logiciel (gratuit) pour piloter le lecteur, chacun peut gérer très simplement sa propre carte à puce sur laquelle il placera ses données confidentielles. La figure 1 représente des cartes et lecteurs Mifare et la figure 2 est une copie d'écran de l'outil Omnikey Cardman 5121 interrogeant une carte Mifare Classic.



Cartes et lecteurs Mifare



2 Copie d'écran de l'outil Omnikey Cardman 5121

Facile d'utilisation, la Mifare Classic n'en possède pas moins un talon d'Achille connu de longue date : la taille de ses clefs, 48 bits seulement. Elle ne doit donc son salut qu'au fait que l'algorithme CRYPTO1 soit tenu secret, empêchant ainsi toute attaque *offline*. En effet, seule une telle attaque peut venir à bout d'une recherche exhaustive sur 48 bits en raison des temps de réponse de la carte. Sauf que... l'année 2008 a été particulièrement difficile pour la Mifare Classic, avec quatre attaques effectives proposées par diverses équipes universitaires. Au final, ces attaques ont mené à la révélation de CRYPTO1 et ont exhibé des faiblesses qui permettent à un attaquant de retrouver les clefs d'une Mifare Classic en moins d'une seconde.

Nous décrivons dans cet article les principes sur lesquels reposent les quatre attaques et montrons leur facilité de mise en œuvre. Nous illustrons leurs conséquences sur un système de contrôle d'accès et présentons les alternatives possibles à la Mifare Classic.

1. Le cœur de la Mifare Classic

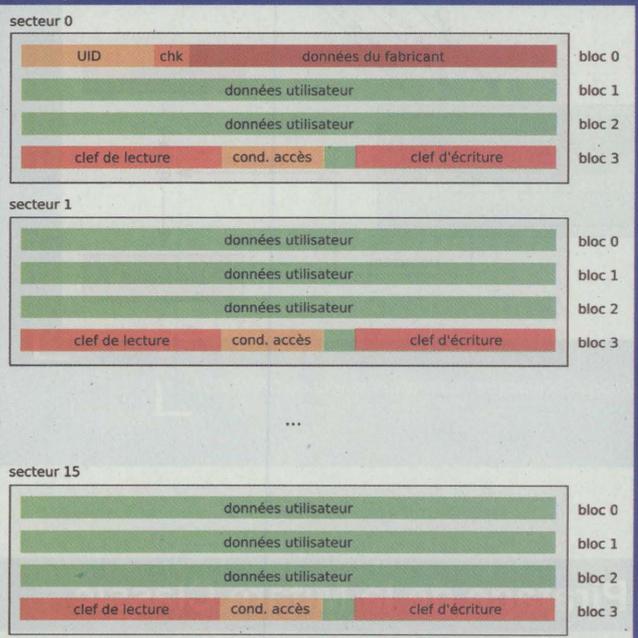
1.1 Organisation de la mémoire

La Mifare Classic se décline en trois versions, dénommées « mini », « 1K » et « 4K », selon la taille de la mémoire EEPROM disponible (respectivement 320, 1024 et 4096 octets). Cette mémoire est organisée en secteurs, eux-mêmes subdivisés en blocs, les plus petites unités mémoire adressables. Pour la Mifare Classic 1K, dont la structure de la mémoire est représentée sur la figure 3, chacun des 16 secteurs est subdivisé en 4 blocs de 16 octets.

Le quatrième bloc de chaque secteur contient deux clefs de 6 octets chacune pour contrôler l'accès en lecture et en écriture au secteur. Il contient également 3 octets qui définissent les conditions d'accès aux blocs (par exemple, l'accès en écriture peut être interdit), et 1 octet dont l'objectif n'est pas défini.

Les trois premiers blocs de chaque secteur (sauf pour le premier secteur) sont disponibles pour stocker des données principalement manipulables à partir des commandes *Read*, *Write*, *Increment* et *Decrement*. L'utilisation de ces commandes nécessite de fournir la clef de lecture ou d'écriture correspondante au secteur.

Le bloc 0 du secteur 0 est particulier, car il contient le numéro de série (UID) de la Mifare (4 octets), une somme de contrôle sur cet UID (1 octet) et des données propres au fabricant de la carte dont la description n'est pas publique. Ce bloc n'est pas modifiable par l'utilisateur.

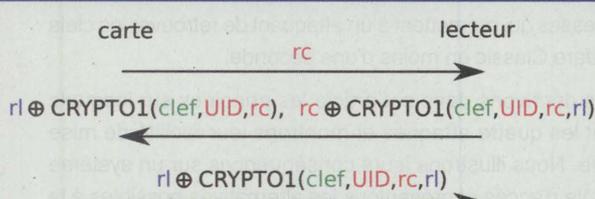


3 Structure de la mémoire d'une Mifare Classic 1K

1.2 Accès à la mémoire : authentification et chiffrement

Le protocole de communication entre un lecteur et la Mifare Classic repose sur les parties 1 à 3 de la norme ISO 14443-A

(la partie 4 de cette norme, optionnelle, est remplacée sur la Mifare Classic par une couche propriétaire). Après avoir exécuté le protocole d'évitement de collisions comme défini dans l'ISO 14443-A, le lecteur connaît les UID de toutes les cartes présentes dans son environnement et peut alors communiquer avec chaque carte de manière individuelle. Notons que cet UID n'est pas chiffré et que tout un chacun peut l'obtenir simplement en interrogeant la carte.



5 Protocole d'authentification

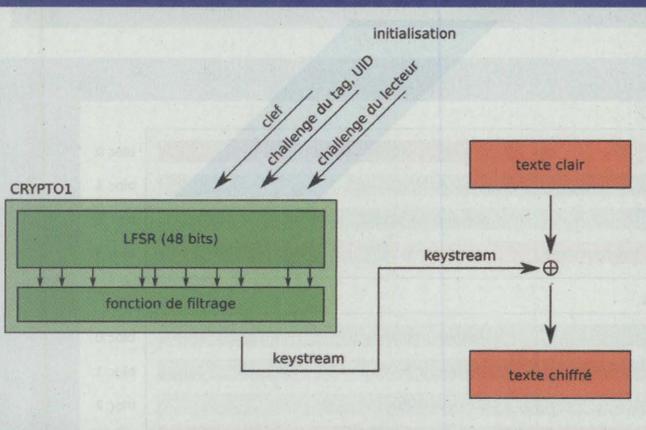
La communication bidirectionnelle entre la carte et le lecteur est chiffrée par l'algorithme de chiffrement par flot CRYPTO1. Celui-ci génère une chaîne pseudo-aléatoire K (*keystream*) qui est XORée bit à bit avec le texte clair P , pour donner le texte chiffré $C = P \oplus K$. Comme représenté très schématiquement sur la figure 4, CRYPTO1 est composé d'un seul LFSR (*Linear Feedback Shift Register*), mais le keystream est obtenu en appliquant une fonction de filtrage à l'état du LFSR. Les lecteurs désireux d'approfondir leur connaissance sur le LFSR et sur la fonction de filtrage pourront consulter l'article [4].

Un point important à retenir pour la suite de cet article concerne l'initialisation de CRYPTO1. En effet, celle-ci s'effectue à travers un protocole d'authentification – un classique challenge/réponse mutuel reposant sur une variante de la norme ISO 9798-2 – qui est exécuté entre la carte et le lecteur dès que ce dernier souhaite accéder (en lecture ou écriture) à un secteur de la carte.

Ainsi, dans un premier temps et avant que le protocole d'authentification ne démarre, CRYPTO1 est seulement initialisé à partir de la clef secrète du secteur (nous parlerons de l'état E_0 du LFSR). Le protocole d'authentification, dont une version simplifiée est donnée sur la figure 5, peut alors se dérouler comme suit :

- 1► La carte envoie au lecteur un challenge pseudo-aléatoire r_c .
- 2► Le lecteur réinitialise CRYPTO1 avec l'UID de la carte et r_c , puis chiffre un challenge r_l . Le LFSR passe alors de l'état E_0 à l'état E_1 . Puis, il réinitialise de nouveau CRYPTO1 avec r_l et chiffre maintenant r_c . Le LFSR passe ainsi de l'état E_1 à l'état E_2 .
- 3► Connaissant l'UID, r_c et r_l , la carte peut à son tour synchroniser son LFSR dans l'état E_2 , et terminer l'authentification mutuelle en renvoyant r_l chiffré au lecteur.

Tous les messages qui seront ensuite échangés entre le lecteur et la carte (pour un secteur donné) seront chiffrés à partir de cette dernière initialisation.



4 Description schématique du fonctionnement de CRYPTO1

2. Piratage de la Mifare Classic

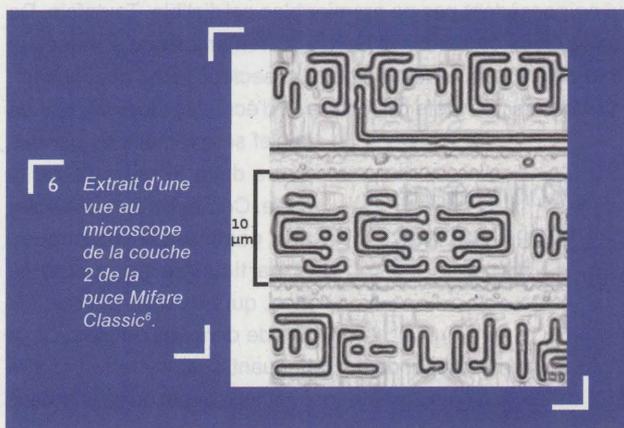
2.1 Attaque de Nohl, Plötz, Starbug et Evans (présentation en décembre 2007, article publié en juillet 2008).

La première attaque connue sur la Mifare Classic remonte à la fin du mois de décembre 2007, quand Nohl (*University of Virginia*, États-Unis) et Plötz (*Humboldt Universität*, Allemagne) font une présentation^{3,4,5} très remarquée lors d'une rencontre

du CCC (*Chaos Computer Club*) à Berlin au cours de laquelle ils montrent qu'il est possible de retrouver les clefs d'une Mifare Classic avec des moyens technologiques extrêmement limités. Ce travail a abouti à une publication [1] par Nohl, Plötz, Starbug et Evans, qui a été présentée à la conférence *USENIX Security Symposium* en juillet 2008.

La première approche de ces chercheurs a été de faire du *reverse engineering hardware* sur la puce afin de reconstruire l'algorithme CRYPTO1. Pour cela, ils ont extrait la puce d'une

carte Mifare Classic et ont ensuite séparé minutieusement les six couches de silicium. La plus basse est exclusivement constituée de transistors. Grâce à une importante série de photos prises de cette couche à l'aide d'un microscope de rapport optique 500x, ils ont été capables de reconstituer par traitement automatique les images de toutes les couches supérieures. Ils se sont alors rendu compte que les transistors étaient groupés de telle sorte que chaque amas représente une porte logique (AND, XOR, flip-flop, etc.) au niveau de la deuxième couche de silicium, comme le montre la figure 6. Sur quelques milliers de portes détectées sur la puce, il n'y en a que 70 de différentes. En utilisant des outils de reconnaissance de formes sur cette deuxième couche, ils ont retrouvé les itérations de chaque porte logique et ont ainsi reconstruit les briques de base de CRYPTO1.



6 Extrait d'une vue au microscope de la couche 2 de la puce Mifare Classic⁶.

Leur seconde approche s'est concentrée sur l'étude du protocole d'authentification entre le lecteur et la carte afin d'affiner les résultats obtenus par l'analyse hardware. Après une étude statistique des échanges carte/lecteur, l'équipe de recherche est arrivée à ses fins et a établi que CRYPTO1 repose sur un Fibonacci LFSR (*Linear Feedback Shift Register*) dont le polynôme primitif est :

$$x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1.$$

Enfin, une observation très intéressante concerne le générateur pseudo-aléatoire. En effet, les chercheurs ont montré que celui-ci repose sur un LFSR de 16 bits de la forme $x^{16} + x^{14} + x^{13} + x^{11} + 1$. Il est initialisé avec une valeur constante au moment où la puce est alimentée par le lecteur. Chaque valeur générée dépend donc du nombre de cycles d'horloge écoulés entre l'entrée de la puce dans le champ du lecteur et la génération de cette valeur. Avec une fréquence de 106 kHz, le générateur boucle toutes les 0,6s. Une observation assez similaire a été faite au niveau du générateur du lecteur, ce qui permet aux chercheurs de parfaitement maîtriser les nombres aléatoires générés durant les échanges entre la carte et le lecteur.

Ayant découvert CRYPTO1 et la manière dont il est utilisé par la Mifare Classic, il est possible de retrouver la clef de la carte par une recherche exhaustive offline. Il faut pour cela

écouter deux échanges challenge/réponse entre une carte et un lecteur légitimes et ensuite essayer de retrouver la clef qui a permis d'obtenir ce résultat. Moins de cinquante minutes sont nécessaires pour tester l'espace complet des clefs avec soixante-quatre FPGA de type Xilinx Virtex-5 LX50 et ainsi retrouver la clef d'une Mifare Classic. Si l'attaquant connaît l'UID de la carte qu'il souhaite attaquer, alors il peut interroger un lecteur avec cet UID sans que la carte légitime ne soit nécessairement présente. La description du protocole d'authentification donnée au début de cet article montre en effet que le lecteur s'authentifie en premier et fournit ainsi les données suffisantes pour effectuer une attaque offline avant même que la carte ne s'authentifie.

Sachant que l'attaquant peut maîtriser les générateurs pseudo-aléatoires, il est même possible de précalculer toutes les réponses potentielles d'une carte ou d'un lecteur pour un UID donné. Après une étape de précalculs équivalente à une recherche exhaustive, une simple recherche dans une table permet alors de retrouver la clef de la carte. Les chercheurs ont également constaté, sans pouvoir le justifier, que si quelques bits sont modifiés dans l'UID, alors il est possible de régénérer le même keystream en modifiant quelques bits de la clef secrète (clef et UID sont XORés durant l'initialisation). L'attaque par précalculs prend alors toute son ampleur, car réaliser les calculs pour un UID donné suffit pour attaquer ensuite n'importe quel UID. Enfin, au lieu de stocker toutes les valeurs précalculées, il est possible d'utiliser une technique de compromis temps-mémoire, présentée en détail dans le numéro 10 de *MISC*.

Notons que les chercheurs n'ont pas prouvé par la pratique qu'ils étaient capables de mettre l'attaque en œuvre. L'annonce de celle-ci a toutefois eu un énorme retentissement aux Pays-Bas, car un déploiement à grande échelle d'un système de billetterie pour les transports en commun basé sur la Mifare Classic y était engagé. Diverses expertises ont été demandées par les autorités néerlandaises à l'institut *Netherlands Organisation for Applied Scientific Research* (TNO) et la question a été débattue à la Chambre des Représentants⁷. Le rapport⁸ du TNO – aujourd'hui déclassifié – confirme la majeure partie des résultats présentés par Nohl et Plötz et plusieurs rapports^{9,10} qui ont fait suite à cette première expertise confirment que retrouver les clefs d'une carte Mifare Classic peut être réalisé en quelques minutes.

⇒ 2.2 Attaque de De Koning Gans, Hoepman et Garcia (attaque rendue publique en mars 2008, article publié en septembre 2008).

En même temps que Nohl et Plötz, et de manière indépendante, De Koning Gans, Hoepman et Garcia [2] – un groupe de chercheurs de l'Université Radboud de Nijmegen aux Pays-Bas – ont également travaillé sur la Mifare Classic, mais avec une toute autre approche : au lieu de chercher à reconstruire CRYPTO1, ils montrent qu'il est possible de découvrir (partiellement) le keystream, ce qui est suffisant pour envoyer des

commandes à la carte et déchiffrer ses réponses. Outre cette attaque, ils ont découvert les mêmes faiblesses du générateur pseudo-aléatoire que celles mises en avant par Nohl et Plötz.

L'attaque des chercheurs néerlandais consiste à tirer profit des propriétés des chiffrements par flot. En effet, comme nous l'avons vu, dans un tel chiffrement le texte clair P est chiffré en l'XORant avec une chaîne K de bits pseudo-aléatoires. Ainsi, le texte chiffré est $C = P \oplus K$. Ce type de chiffrement est sûr pour autant que personne ne puisse prédire K sans connaître l'initialisation de l'algorithme, autrement dit sans connaître la clef secrète. Nous avons vu que les générateurs pseudo-aléatoires de la carte et du lecteur peuvent être influencés par l'attaquant ; cela permet à ce dernier de forcer la carte à utiliser plusieurs fois le même keystream K . Si un attaquant observe deux messages C_0 et C_1 échangés entre une carte et un lecteur légitimes, utilisant le même keystream, alors nous avons : $C_0 = P_0 \oplus K$ et $C_1 = P_1 \oplus K$. Si maintenant l'attaquant connaît l'un de ces messages clairs, par exemple P_1 , alors le second message clair est révélé : $P_0 = C_0 \oplus C_1 \oplus P_1$.

Nous montrons maintenant plus précisément comment tirer profit de cette propriété, en décrivant les étapes à suivre pour mener à bien l'attaque décrite dans [2].

- 1▶ L'attaquant enregistre une communication entre un lecteur et une carte tous deux légitimes. Le but de l'attaquant est de décrypter cette communication.
- 2▶ L'attaquant joue maintenant le rôle du lecteur et communique directement avec la carte. Il rejoue la phase d'authentification enregistrée en forçant la carte légitime à réutiliser le même challenge que celui écouté.
- 3▶ Lorsque l'authentification est terminée, l'attaquant renvoie une commande enregistrée pendant l'échange légitime, en la modifiant de telle sorte que les informations renvoyées par la carte soient connues de l'attaquant. Nous verrons plus loin comment forger une telle commande.
- 4▶ Connaissant les informations renvoyées par la carte en clair et en chiffré, l'attaquant peut en déduire le keystream correspondant.
- 5▶ L'attaquant peut enfin utiliser ce keystream pour déchiffrer l'échange enregistré en 1.

Le point critique dans cette procédure est de réussir à forger des commandes, pour forcer la carte à renvoyer des réponses chiffrées dont l'attaquant connaît le clair. Ceci peut être réalisé simplement en utilisant la propriété de malléabilité des chiffrements par flot : si un attaquant XORe un message chiffré C avec S , alors le message clair obtenu après déchiffrement aura été biaisé de S également, par rapport au message clair original. Autrement dit : $C \oplus S = (P \oplus K) \oplus S = (P \oplus S) \oplus K$. En conséquence, si un lecteur légitime envoie une commande chiffrée pour lire un bloc de la carte, l'attaquant peut rejouer cette commande en la

modifiant (sans même la déchiffrer) de telle sorte à changer le numéro du bloc à lire.

Nous savons que le premier bloc du premier secteur contient des données dont certaines sont connues ou prédictibles (voir Figure 3). Il y a l'UID, un test d'intégrité sur cet UID, ainsi que des informations sur le fabricant, qui ne sont pas publiquement documentées, mais dont certaines sont constantes. Si l'attaquant a judicieusement modifié la commande du lecteur, la carte renverra de manière chiffrée à l'attaquant le contenu du premier bloc du premier secteur. L'affaire est conclue, l'attaquant connaît une réponse en clair, son chiffré, donc le keystream correspondant. Il peut donc maintenant déchiffrer les autres blocs du premier secteur.

L'affaire est un peu plus compliquée pour les autres secteurs, qui ne possèdent pas ce premier bloc prédictible. Toutefois, De Koning Gans, Hoepman et Garcia ont remarqué que si un lecteur tente de lire le quatrième bloc d'un secteur (celui qui contient, entre autres, les clefs de lecture et d'écriture) alors les bits de

la clef sont remplacés par des zéros dans la réponse de la carte. Ceci évite d'implémenter une commande compliquée et particulière au quatrième bloc, qui ne renverrait que les bits de données du bloc. Cette

approche permet cependant à l'attaquant d'obtenir le chiffré d'un texte clair connu (des zéros) et donc le keystream correspondant. Six octets de keystream sont ainsi révélés grâce à la clef de lecture, et six autres grâce à la clef d'écriture lorsque celle-ci est présente (optionnelle, cette dernière peut laisser sa place à des données). Le quatrième bloc fournit donc 6 ou 12 octets de keystream à l'attaquant, ce qui lui permet de déchiffrer 6 ou 12 octets de chacun des autres blocs de ce secteur.

En résumé, si l'attaquant peut avoir accès à la carte qu'il souhaite attaquer, et s'il est en mesure d'écouter une communication (par secteur) entre cette carte et un lecteur légitime, alors il peut obtenir toutes les données du premier secteur, ainsi que les 6 premiers octets et fréquemment les 6 derniers octets de chacun des blocs de données des autres secteurs.

L'attaque proposée par De Koning Gans, Hoepman et Garcia a été validée par des experts de l'institut scientifique TNO et a même fait l'objet d'une communication parlementaire par le ministre de l'Intérieur. Vu les enjeux économiques qu'implique une migration de la Mifare Classic vers une solution mieux sécurisée, les autorités néerlandaises ont demandé une contre-expertise au « Information Security Group – Smart Card Centre – Royal Holloway, University of London ». Ce rapport^{11 12} confirme les indications déjà fournies dans les rapports du TNO, et incite même de presser le remplacement des Mifare Classic par une solution plus adéquate dans le cadre du nouveau système de billetterie des transports en commun aux Pays-Bas. Suite à ce rapport, un deuxième avis a été soumis à la Chambre des Représentants¹³.

...L'attaque proposée par De Koning Gans, Hoepman et Garcia a été validée par des experts de l'institut scientifique TNO...

⇒ 2.3 Attaque de Courtois, Nohl et O'Neil (annonce publiée en avril 2008).

La troisième attaque – une « *algebraic attack* » – a été annoncée à la mi-avril 2008 par Courtois (*University College of London*, Royaume-Uni), Nohl (*University of Virginia*, États-Unis) et O'Neil (*VEST Corporation*, France) [3] et présentée durant la *rump session* d'Eurocrypt 2008. Cette annonce de trois pages, rendue publique sur les *eprints* de l'IACR, ne précise pas en détail la procédure de l'attaque qui consiste à résoudre un large système d'équations polynomiales. Les auteurs précisent qu'avec 50 bits du keystream (qui peuvent être obtenus grâce à l'étude faite par l'équipe des Pays-Bas), il suffit d'écouter un seul échange entre un lecteur et une Mifare Classic pour découvrir en 3 minutes de calculs la clé secrète de la carte. Selon les auteurs, avec 4 écoutes judicieusement choisies, il est même possible de retrouver la clé en seulement 12 secondes. Le contenu de l'attaque n'ayant pas encore été publié, il n'est pas possible aujourd'hui de vérifier qu'elle fonctionne réellement.

⇒ 2.4 Attaque de Garcia, De Koning Gans, Muijers, van Rossum, Verdult et Wichers Schreur (vidéo de l'attaque diffusée en mars 2008, article publié en octobre 2008).

Enfin, la dernière attaque connue sur la Mifare Classic – la plus dévastatrice – est encore due à Flavio Garcia et Gerhard De Koning Gans, de l'université Radboud de Nijmegen aux Pays-Bas, accompagnés de leurs collègues Muijers, van Rossum, Verdult et Wichers Schreur. Comme l'équipe germano-américaine, l'équipe néerlandaise a réalisé du reverse engineering dans le but de reconstruire CRYPTO1. L'approche est toutefois différente puisqu'il ne s'agissait pas ici d'analyser le hardware avec un microscope, mais de reconstruire l'algorithme en analysant des communications carte/lecteur. Les chercheurs ont pour cela utilisé l'algorithme de chiffrement par flot du Hitag2 – un produit NXP dont les spécifications peuvent être trouvées sur Internet – en supposant que CRYPTO1 était assez similaire à celui-ci. Nous décrivons ci-dessous les deux attaques proposées dans [4], et nous renvoyons le lecteur à ce même article s'il souhaite obtenir des détails sur l'approche utilisée pour retrouver CRYPTO1. Toutes les deux reposent sur le fait qu'il est possible de retrouver des séquences du keystream lorsque l'attaquant connaît un couple (message clair, message chiffré).

La première attaque consiste en trois étapes : (a) une phase de précalculs indépendante de la carte visée ; (b) une phase dépendante de la carte visée qui permet de retrouver l'état E2 du LFSR de CRYPTO1, sans que la présence de la carte ne soit

pour autant requise ; (c) une phase qui permet de découvrir l'état E0 (c'est-à-dire la clé secrète) à partir de l'état E2 du LFSR.

La première phase nécessite un calcul exhaustif sur un sous-ensemble des états possibles du LFSR. Ainsi, pour chaque état de la forme $0x000*****$ où * représente une valeur hexadécimale quelconque, les 64 premiers bits du keystream sont générés et les 2³⁶ couples (état, keystream) obtenus sont enregistrés dans une table. Si cette étape est coûteuse en espace (environ un téraoctet de mémoire est nécessaire pour son stockage), elle peut néanmoins être réalisée sur un simple PC en un temps raisonnable.

La seconde phase consiste à exécuter le protocole d'authentification face à un lecteur légitime, en envoyant à ce dernier un challenge de 32 bits de la forme $0x0000***0$. Il faut pour cela disposer d'une carte dont l'UID peut être choisi (pour lui donner la valeur de l'UID de la carte visée par l'attaque). Une telle carte Mifare Classic n'existe pas, alors les auteurs de [4] ont fabriqué eux-mêmes un dispositif – appelé *ghost*¹⁴ – qui implémente une Mifare Classic, tout en permettant de choisir l'UID ainsi que le challenge utilisé pour l'authentification. Le lecteur répond donc au ghost en XORant le challenge reçu avec le keystream. Challenge et réponse du lecteur permettent alors à l'attaquant de découvrir 32 bits de keystream. En restant silencieux à cette réponse, le ghost incite le lecteur à interrompre l'authentification par une commande « halt » chiffrée. Le format de cette commande est connu, ce qui permet à l'attaquant d'obtenir 32 bits supplémentaires de keystream. En répétant 4096 fois cette tentative d'authentification (testant tous les challenges de la forme $0x0000***0$), l'attaquant est certain d'aboutir à un keystream de la table précalculée et donc de déterminer E2.

La troisième phase consiste à retrouver E0 – c'est-à-dire la clé secrète – à partir de E2. Connaissant l'état du LFSR à un instant k , il est a priori possible de calculer l'état du LFSR à l'instant $(k-1)$ puisque le polynôme primitif $g(x)$ est connu depuis les travaux de Nohl et Plötz. Toutefois, si le passage de E0 à E1 ne dépend que du challenge du ghost et de son UID (tous deux connus), le passage de E1 à E2 dépend du challenge du lecteur qui, lui, n'est pas connu de l'attaquant. Malheureusement, la fonction de filtrage de CRYPTO1 ne porte pas sur tous les bits de l'état et l'équipe néerlandaise montre dans [4] qu'il est alors possible de retrouver E1 à partir de E2 en seulement 32 essais. À partir de E1, il n'y a pas de difficulté à retrouver E0, c'est-à-dire la clé secrète de la carte.

La seconde attaque repose sur l'inversibilité de la fonction de filtrage. La faiblesse exploite le fait que l'entrée de cette fonction ne repose que sur les bits de position impairs du LFSR (à l'état E2, à partir du bit numéro 9). En ayant une partie du keystream, on peut exploiter cette particularité pour retrouver séparément les bits appropriés du LFSR qui donnent les bits pairs (respectivement impairs) du keystream. Les résultats peuvent être stockés dans deux tableaux de 2¹⁹ lignes (environ 8 mégaoctets de mémoire). Un subtil jonglage avec le LFSR (voir [4]) montre qu'au final, pour un keystream de longueur n bits, la fusion de deux lignes correctement choisies forme une séquence $r = r0 r1 \dots r(47+n)$, où

chaque bloc de 48 bits consécutifs représente les différents états du LFSR qui produisent le keystream en question. Alors le bloc $r_0 \dots r_{47}$ est l'état initial E_0 du LFSR, c'est-à-dire la clef secrète.

Les deux attaques décrites dans [4] ont été mises en pratique par l'équipe de l'université de Radboud. Dans le premier cas, l'attaquant enregistre 2^{12} authentifications entre un lecteur légitime et le ghost, ce qui prend entre 2 et 14 minutes selon que le lecteur est en ligne ou pas, puis retrouve la clef secrète en quelques secondes seulement en s'aidant de la table précalculée. Cette

attaque a fait l'objet de la diffusion d'une vidéo sur YouTube¹⁵ (Figure 7), qui montre très clairement la facilité de sa mise en œuvre. Notons qu'il est possible de réduire la taille de la table précalculée en augmentant le nombre de requêtes envoyées au lecteur ou inversement. L'autre attaque qui utilise la faiblesse sur la fonction de filtrage ne nécessite qu'une seule interrogation d'un lecteur légitime. Cette dernière attaque, qui n'utilise pas la table précalculée, peut être réalisée en moins d'une seconde. Il s'agit de l'attaque la plus dévastatrice de toutes.

3. Un exemple concret : contrôle d'accès à un bâtiment



7 Vidéo de l'attaque de l'université de Radboud sur YouTube

Afin d'illustrer sur un exemple concret la Mifare Classic et les conséquences des attaques, nous présentons une architecture usuelle de contrôle d'accès à un bâtiment. Dans une telle application, des lecteurs sont placés aux abords des accès au bâtiment et sont reliés à un contrôleur situé dans un

local technique du bâtiment. La communication, qui repose sur le protocole Wiegand, est unidirectionnelle, du lecteur vers le contrôleur. Le contrôleur – une unité autonome qui possède une batterie de secours – ainsi que les potentiels contrôleurs d'autres bâtiments sont connectés par TCP/IP au serveur qui gère le contrôle d'accès.

Les lecteurs se chargent de l'authentification et possèdent donc les clefs des cartes : lorsqu'une carte se présente, le lecteur utilise la clef appropriée pour exécuter le protocole d'authentification mutuelle et accède ensuite à un bloc donné qui contient une valeur d'authentification, en fait un simple identificateur unique à chaque carte. Comme nous l'avons vu, cet identificateur est chiffré lorsqu'il transite de la carte au lecteur. Il est ensuite déchiffré par le lecteur, puis envoyé au contrôleur. Celui-ci vérifie dans sa table que l'identificateur qu'il a reçu est autorisé à accéder au bâtiment¹⁶ ; le cas échéant, le contrôleur déclenche l'ouverture de la gâche électrique de la porte. Le contrôleur envoie alors un log au serveur ou le stocke en mémoire

si la connexion avec le serveur ne peut être établie. Le serveur peut interagir avec le contrôleur afin d'introduire de nouvelles règles dans sa table d'autorisation, mais n'intervient directement ni pour l'authentification, ni pour l'autorisation.

En général, un secteur de la carte est utilisé pour stocker l'identificateur et les autres secteurs sont disponibles pour d'autres services, éventuellement gérés par d'autres sociétés. Ces autres services peuvent être encore du contrôle d'accès, par exemple pour les parkings, du paiement pour les distributeurs de boissons, etc. Les clefs de lecture et d'écriture peuvent être différentes pour chaque secteur, ce qui permet de gérer séparément chaque service. Les clefs devraient également être différentes pour chaque carte. Il est toutefois fréquent de voir des systèmes où les mêmes clefs sont utilisées pour toutes les cartes. Cette approche, contraire aux principes de base de la sécurité, est utilisée avec certains lecteurs qui ne peuvent stocker que quelques clefs ou qui ne permettent pas la sélection de la clef à utiliser en fonction de l'UID reçu.

L'attaque proposée par l'équipe de l'université de Radboud permet alors à un attaquant de lire la carte d'un utilisateur, d'en faire une copie (voir Figure 7), et éventuellement d'en modifier le contenu si l'accès en écriture n'est pas bloqué. Si les mêmes clefs sont utilisées pour toutes les cartes comme nous l'avons observé dans certains contrôles d'accès, alors l'attaque d'une seule carte compromet tout le système.

Notons que des cartes Mifare Classic vierges peuvent être achetées librement sur Internet pour moins d'un euro et qu'il existe des logiciels prêts à l'emploi pour lire et écrire sur de telles cartes ; il n'est donc pas difficile du tout de modifier ou dupliquer une carte lorsque l'on connaît les clefs. Cette approche ne permet toutefois pas de créer un clone parfait, puisque les UID des cartes Mifare Classic (même vierges) ne peuvent pas être changés. Toutefois, dans les systèmes que nous avons observés, l'UID n'est utilisé que pour l'évitement de collision et n'est pas remonté jusqu'à la couche application. Autrement dit, le lecteur ne possède pas de table de correspondance entre les UID et les identificateurs des cartes. Un clone imparfait est donc souvent tout à fait suffisant pour passer le contrôle.

4. Alternatives à la Mifare Classic

La facilité de mettre en œuvre les attaques décrites précédemment et les conséquences que l'on peut attendre de ces attaques montrent la nécessité de faire évoluer les systèmes reposant actuellement sur la technologie Mifare Classic. Nous décrivons ci-dessous quelques alternatives à la Mifare Classic, en décrivant avant tout les autres cartes appartenant à la famille Mifare.

La « Mifare UltraLight » est un produit d'entrée de gamme qui ne propose pas de fonctionnalité cryptographique. Inadapté au contrôle d'accès, il est le plus souvent utilisé comme ticket jetable. Une évolution très récente de ce produit est la « Mifare Ultralight C » qui dispose de 3DES. Difficile toutefois d'être plus précis sur les spécifications de ce produit annoncé par NXP en août 2008.

La « DESFire » est plus puissante que la Classic et possède notamment un coprocesseur cryptographique. Les algorithmes de chiffrement DES, 3DES, voire AES pour la version DESFire EV1 (anciennement connue sous le nom DESFire8), peuvent être utilisés sur cette carte en lieu et place de CRYPTO1. Cette évolution constitue une avancée notable, puisque la sécurité ne repose plus sur le secret de l'algorithme. La DESFire offre également plus de souplesse que la Classic dans le stockage des informations, car la structure de la mémoire n'est pas figée : lors de la personnalisation de la carte, chaque application définit la taille des fichiers dont elle a besoin. L'accès à la mémoire est également plus souple, puisqu'il répond à la norme ISO14443 Part-4, contrairement à la Mifare Classic qui nécessite une API spécifique. Enfin, la DESFire permet d'atteindre un taux de transfert de données de 848 kbit/s dans les versions les plus récentes contre seulement 106 kbit/s pour la Classic. Le coût de la DESFire¹⁷ est cependant environ trois fois plus élevé que celui de la Classic.

La « SmartMX » est la carte la plus puissante dans la gamme Mifare, parfois même considérée comme n'étant pas une Mifare. Elle dispose d'une large mémoire de 12 à 144 kilo-octets d'EEPROM, de 200 kilo-octets de ROM et 6 kilo-octets de RAM, et permet l'utilisation aussi bien de cryptographie symétrique (3DES et AES) qu'asymétrique (RSA, ECC). Son usage est principalement destiné aux applications bancaires et aux documents d'identité.

Dans le but de favoriser la transition de la Classic vers une solution mieux sécurisée, NXP a annoncé la mise sur le marché de la « Mifare Plus » au cours de l'année 2009. Cette carte utilisera une structure fixe pour sa mémoire et intégrera deux modules cryptographiques : l'un permettant l'utilisation de CRYPTO1, l'autre de AES. Il sera ainsi possible de gérer un système où cohabiteront des Mifare Classic et des Mifare Plus, autorisant ainsi une période de transition pour le renouvellement des cartes et des lecteurs dans le système. Lorsque la migration vers la Mifare Plus sera achevée, le module CRYPTO1 de chaque carte pourra être irrémédiablement désactivé en envoyant à la carte une commande ad hoc authentifiée par AES.

Si le remplacement brutal de toutes les cartes et lecteurs du système est possible, il est alors envisageable d'avoir recours à un autre fabricant. Sony, par exemple, a développé le système « FeliCa »¹⁸ dans le but d'avoir une plateforme multi-application. Cette carte est largement déployée dans le monde asiatique, mais reste plus discrète en Europe. Elle est compatible avec la norme ISO 18092 relative à la NFC (*Near Field Communication*). Comme la DESFire, la FeliCa utilise l'algorithme 3DES pour l'authentification mutuelle entre la carte et le lecteur. Sa vitesse de communication, 212 kbit/s pour les versions les plus rapides, reste inférieure à celle de la DESFire, mais suffit amplement au contrôle d'accès.

« Calypso »¹⁹ est un système développé par un groupe de partenaires européens actifs dans le domaine des transports publics. La carte Calypso n'est qu'un des éléments d'une solution intégrée visant à faciliter la gestion de la billetterie dans les transports, tout en offrant une sécurité renforcée. Calypso offre la possibilité d'utiliser DES, DESX ou 3DES, et la communication repose sur le standard ISO 14443-B. Ce système ouvert offre la possibilité d'ajouter de nouvelles applications sur la carte, telle qu'un porte-monnaie électronique. Contrairement aux cartes Mifare ou FeliCa, les spécifications des cartes Calypso ne sont pas tenues secrètes – bien que distribuées avec parcimonie – et toute société peut produire des cartes répondant aux critères Calypso, moyennant l'obtention d'une licence auprès de la société Innovatron.

La firme allemande Infineon propose quant à elle trois types de cartes sous la dénomination « my-d » (*proximity, light ou vicinity*) qui reposent sur les standards ISO14443-A (*proximity*) ou ISO15693 (*light, vicinity*). D'après les documentations disponibles sur le site de la firme, les protocoles cryptographiques utilisés ne semblent pas standards. En outre, la taille des clefs est limitée à 64 bits.

Citons également que la firme suisse Legic propose également plusieurs puces²⁰ répondant aussi bien à la norme ISO14443-A qu'à la norme ISO15693, avec différentes tailles de mémoire. Nous citerons enfin que la société HID, bien connue pour offrir une large palette de produits RFID, possède dans son catalogue des cartes « iClass »²¹ qui existent en plusieurs versions de taille de mémoire.

Le passage à l'une de ces alternatives à partir d'un système reposant sur une Mifare Classic n'est pas une chose aisée et doit être préparé avec minutie. Outre les problèmes de coût (changer un système de contrôle d'accès peut coûter plusieurs centaines de milliers d'euros dans les grandes entreprises, montant difficilement justifiable auprès de la direction), il y a des impératifs bien réels : la transition ne peut se faire en une nuit et on imagine mal un système de contrôle d'accès restant non opérationnel pendant toute une semaine.

...NXP a annoncé la mise sur le marché de la « Mifare Plus » au cours de l'année 2009...

Conclusion

La carte Mifare Classic, qui occupe la tête des ventes depuis plusieurs années dans le domaine des cartes à puce sans contact, est aujourd'hui totalement cassée. Un attaquant peut alors retrouver les clefs d'une carte en interrogeant simplement un lecteur pendant une fraction de seconde. Cette attaque est possible aujourd'hui, elle sera demain à la

portée de tous, puisque l'article [4] délivre tous les éléments essentiels à sa mise en œuvre. Le destin des cartes Mifare Classic pourrait alors s'apparenter à celui des cartes à pistes magnétiques, qui ne présentent aucune sécurité intrinsèque, mais qui restent largement déployées, en particulier dans le domaine du contrôle d'accès. ■

Notes

- ¹ <http://www.nxp.com/products/identification/mifare/classic/>
- ² Entre \$0,46 et \$0,84 pièce pour le modèle 1K, selon la quantité commandée. Source : <http://www.smartcardworld.com>
- ³ http://www.icp.ucl.ac.be/mifare/mifare_security.mp4 (vidéo de la présentation à CCC).
- ⁴ http://www.icp.ucl.ac.be/mifare/1049_CCC-07-Mifare-v2.pdf (transparents de Karsten Nohl).
- ⁵ http://www.icp.ucl.ac.be/mifare/1051_24c3-mifare-henryk-ooo.pdf (transparents de Henryk Plötz).
- ⁶ Photo extraite de l'article [1]. Reproduction avec l'aimable autorisation de Karsten Nohl.
- ⁷ <http://www.cs.vu.nl/~ast/ov-chip-card/>
- ⁸ http://www.tno.nl/downloads%5Ctno_ict_reaction_ccc_presentation_170108.pdf
- ⁹ http://www.translink.nl/media/bijlagen/nieuws/TNO ICT - Security Analysis OV-Chipkaart - public_report.pdf
- ¹⁰ http://www.translink.nl/media/bijlagen/nieuws/Conclusies_aanvullend_onderzoek_TNO_12_maart.pdf
- ¹¹ http://www.verkeerenwaterstaat.nl/Images/20083843%20bijlage%201_tcm195-218926.pdf
- ¹² http://www.verkeerenwaterstaat.nl/Images/20083843%20bijlage%202_tcm195-218928.pdf
- ¹³ http://www.verkeerenwaterstaat.nl/Images/20083843_tcm195-218925.pdf (en néerlandais)
- ¹⁴ Le matériel nécessaire pour la fabrication de ce ghost ne dépasse pas 40 euros.
- ¹⁵ <http://www.youtube.com/watch?v=NW3RGbQTLhE>
- ¹⁶ La table d'autorisation contient les identificateurs autorisés à accéder au bâtiment, mais peut également contenir des conditions supplémentaires pour chacun de ses identificateurs, par exemple les plages horaires durant lesquelles l'accès doit être autorisé.
- ¹⁷ Environ \$2,50 pour l'achat d'une DESFire à l'unité. Source : <http://www.smartcardworld.com>
- ¹⁸ <http://www.sony.net/Products/felica/pdt/crd.html>
- ¹⁹ <http://calypso.spiritech.net/>
- ²⁰ http://www.legic.com/fr/legic_advant/puces_transpondeur.html
- ²¹ http://www.hidcorp.com/francais/prod_detail.php?prod_id=35

Références

- ⇒ [1] NOHL (Karsten), EVANS (David), STARBUG, PLÖTZ (Henryk), « *Reverse-Engineering a Cryptographic RFID Tag* », Conférence USENIX Security Symposium, juillet 2008.
- ⇒ [2] DE KONING GANS (Gerhard), HOEPMAN (Jaap-Henk), GARCIA (Flavio D.), « *A Practical Attack on the MIFARE Classic* », Conférence CARDIS, septembre 2008.
- ⇒ [3] COURTOIS (Nicolas), NOHL (Karsten), O'NEIL (Sean), « *Algebraic Attacks on the Crypto1 Stream Cipher in MiFare Classic and Oyster Cards* », *IACR ePrint*, numéro 2008/166, avril 2008.
- ⇒ [4] GARCIA (Flavio D.), DE KONING GANS (Gerhard), MUIJRS (Ruben), VAN ROSSUM (Peter), VERDULT (Roel), WICHERS SCHREUR (Ronny), « *Dismantling MIFARE Classic* », Conférence ESORICS, octobre 2008.
- ⇒ [5] FINKENZELLER (Klaus), *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, John Wiley & Sons, New York, NY, USA, *second edition*, 2003.

Ces références sont accessibles à partir de la page www.avoine.net/rfid/.

L'attaque proposée par Garcia, De Koning Gans, Muijrs, van Rossum, Verdult et Wichers Schreur a été acceptée pour publication à la conférence ESORICS où elle sera présentée en octobre 2008. Quelques mois auparavant, en juillet, NXP avait tenté d'interdire la publication de l'article en portant l'affaire devant le tribunal d'Arnhem aux Pays-Bas. Traités d'« irresponsables » par NXP, les chercheurs néerlandais ont mis en avant le fait que NXP n'a pas été pris au dépourvu puisqu'ils ont annoncé publiquement les résultats de leurs travaux en mars 2008 et que des faiblesses sur la Mifare Classic étaient déjà connues depuis décembre 2007. Au nom de la liberté d'expression, la justice néerlandaise a décidé d'autoriser la publication de l'article. Plus récemment, la justice américaine a également relaxé des étudiants du MIT qui avaient exhibé des faiblesses dans le système de contrôle d'accès des transports publics du Massachusetts.

Offre Collectionneur !

Vous êtes un fidèle lecteur mais vous ne vous rappelez plus dans quel magazine vous avez lu un article sur ... ?

Un sujet vous passionne et vous recherchez des magazines traitant de ce sujet ?



Allez sur www.ed-diamond.com et utilisez le moteur de recherche sur tous les sommaires des magazines édités par Diamond Editions (Misc, GNU/Linux Magazine et hors-série, Linux Pratique, Linux Pratique Essentiel). Vous pourrez également compléter votre collection !

Bon de commande à remplir et à retourner à : Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

DÉSIGNATION	PRIX	QTÉ	TOTAL
MISC N°1 Les vulnérabilités du Web !	5,95 €		
MISC N°2 Windows et la sécurité	7,45 €		
MISC N°4 Internet, un château construit sur du sable	7,45 €		
MISC N°6 Insécurité du wireless ?	7,45 €		
MISC N°7 La guerre de l'information	7,45 €		
MISC N°8 Honeybots : le piège à pirates	7,45 €		
MISC N°9 Que faire après une intrusion ?	7,45 €		
MISC N°10 VPN (Virtual Private Network)	7,45 €		
MISC N°11 Tests d'intrusion	7,45 €		
MISC N°12 La faille venait du logiciel !	7,45 €		
MISC N°13 PKI - Public Key Infrastructure	7,45 €		
MISC N°14 Reverse Engineering	7,45 €		
MISC N°16 Télécoms, les risques des infrastructures	7,45 €		
MISC N°17 Comment lutter contre le spam, les malwares, les spywares	7,45 €		
MISC N°18 Dissimulation d'informations	7,45 €		
MISC N°19 Les dénis de service	7,45 €		
MISC N°20 Cryptographie malicieuse	7,45 €		
MISC N°21 Limites de la sécurité	7,45 €		
MISC N°22 Superviser sa sécurité	7,45 €		
MISC N°23 De la recherche de faille à l'exploit	7,45 €		
MISC N°24 Attaques sur le Web	7,45 €		
MISC N°25 Bluetooth, P2P, AIM, les nouvelles cibles	7,45 €		
MISC N°26 Matériel mémoire, humain, multimédia	8,00 €		
MISC N°27 IPv6 : sécurité, mobilité et VPN, les nouveaux enjeux	8,00 €		
MISC N°28 Exploits et correctifs : les nouvelles protections à l'épreuve du feu	8,00 €		
MISC N°29 Sécurité du cœur de réseau IP	8,00 €		
MISC N°30 Les protections logicielles	8,00 €		
MISC N°31 Le risque VoIP : le PABX est-il votre faiblesse ?	8,00 €		
MISC N°32 Que penser de la sécurité selon Microsoft ?	8,00 €		
MISC N°33 RFID, instrument de sécurité ou de surveillance ?	8,00 €		
MISC N°34 Noyau et Rootkit : attaque, exploitation, corruption et dissimulation au cœur du système	8,00 €		
MISC N°35 Autopsie & Forensic : comment réagir après un incident ?	8,00 €		
MISC N°36 Lutte informatique offensive : les attaques ciblées	8,00 €		
MISC N°37 Déni de service : vos serveurs en ligne de mire	8,00 €		
MISC N°38 Code malicieux : quoi de neuf ?	8,00 €		
MISC N°39 Fuzzing : injectez des données et trouvez les failles cachées	8,00 €		

TOTAL
Frais de port France Metro : + 3,81 €
Frais de port Etranger : + 5,34 €
TOTAL

Oui je souhaite compléter ma collection

1 Voici mes coordonnées postales

Nom : _____

Prénom : _____

Adresse : _____

Code Postal : _____

Ville : _____

2 Je joins mon règlement :

Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

Paiement par carte bancaire :

N° Carte : _____

Expire le : _____ Cryptogramme Visuel : _____ Voir image ci-dessous

Date et signature obligatoire : _____ 200 _____

Votre cryptogramme visuel

4 façons de commander :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-17h au 03 88 58 02 08
- par fax au 03 88 58 02 09 (CB)

MISE EN PLACE DE PKI ET CARTE À PUCE

■ mots clés : PKI / carte à puce / personnalisation / déblocage / middleware

Face aux nouvelles menaces liées à l'interconnexion croissante des SI (commerce en ligne, dématérialisation, mobilité des salariés,...) et des menaces internes, l'utilisation de certificats électroniques s'impose comme étant l'une des solutions les plus sécurisées pour garantir le contrôle d'accès, la traçabilité et l'imputabilité, en d'autres termes, garantir l'identité de la personne ayant agi. Des infrastructures de gestion de clés (PKI, Public Key Infrastructure) ont d'ailleurs été déployées dans de nombreuses grandes entreprises. Il apparaît même aujourd'hui des cartes d'identité nationales intégrant des certificats numériques (Portugal, Belgique,...).

La sécurité d'un système bâti sur des certificats repose sur la protection des clés privées des autorités de certification, des serveurs et des utilisateurs. C'est au confinement de la clé privée de ces derniers que nous nous intéresserons : la carte à puce.

➔ 1. Les enjeux

Et voilà donc avec l'intégration des cartes dans le système, l'apparition de toutes nouvelles problématiques.

En effet, la mise en œuvre des cartes à puce dans le cadre d'une PKI ne revient pas simplement à associer à des utilisateurs un objet doté d'une certaine intelligence. Elle implique de réfléchir

aux usages, donc aux applications, à l'intégration de la carte sur le poste de travail, donc au déploiement des matériels et logiciels, et de définir toutes les procédures qui rendront la carte utilisable par des humains capables d'oublier leur code PIN¹ ou de perdre leur carte.

➔ 2. La carte PKI

Les cartes à puce utilisées dans les projets de PKI sont très majoritairement des cartes Java. Pour fonctionner sur le poste utilisateur, les cartes PKI utilisent :

- ⇒ un lecteur de carte à puce ;
- ⇒ une interface applicative de haut niveau ou *middleware* cryptographique.

Peu ou prou, tous les fabricants de cartes fournissent des cartes pour la PKI. Ils se différencient néanmoins quant à la fourniture des cartes.

La personnalisation des cartes intervient tout d'abord à deux niveaux :

- ⇒ la personnalisation graphique ;

⇒ la personnalisation électrique (pré-personnalisation et personnalisation seront confondues).

Puis, viennent les problématiques de :

- ⇒ déblocage de carte ;
- ⇒ recyclage.

⇒ 2.1 Architecture du poste utilisateur

⇒ 2.1.1 Clé ou carte pleine

Le choix du facteur de forme dépend de plusieurs critères :

- ⇒ usage pratique de la carte à puce : accès uniquement logique ou également accès physique ;
- ⇒ impression d'informations personnelles : numéro de matricule RH, photo, nom, prénom, etc. ;
- ⇒ coût ;
- ⇒ facilité de déploiement.

Par exemple, lorsque l'utilisation de la carte se limite à la PKI, l'utilisation d'une carte pleine au format carte de crédit n'est pas nécessaire. Malgré tout, la tendance, pour des raisons pratiques, est de mutualiser l'utilisation de la carte à puce pour des accès physiques et logiques.

Le facteur économique est également essentiel : en privilégiant la carte pleine au détriment d'une carte au format clé, il faut ajouter le coût du lecteur à la solution globale.

De plus, dans un parc informatique souvent hétérogène, il faudra prendre en compte les coûts d'intégration des lecteurs (PCMCIA pour les ordinateurs portables, USB pour les autres).

Ce coût peut tout à fait se justifier dans le cas d'une carte multi-usage où le coût de la carte sera finalement mutualisé sur l'ensemble des applications.

⇒ 2.1.2 Le middleware PC/SC²

Pour utiliser une carte à puce dans un environnement de type PC, il est nécessaire de connecter un lecteur et d'avoir un support logiciel sur le PC. Les lecteurs de carte sont attachés à une des interfaces standards du PC : USB ou PCMCIA, plus rarement série et PS/2.

En créant une couche d'abstraction du lecteur, PC/SC garantit l'interopérabilité entre les applications et les cartes à puce.

PC/SC, nativement intégré sous Windows (Win32 API), a également été porté sous Linux [1] et un wrapper est aussi proposé pour le développement d'applications en Java au-dessus de PC/SC.

Les principaux fournisseurs de lecteurs de carte à puce (SCM, Omnikey, Gemalto,...) fournissent également les pilotes PC/SC pour Linux et Mac.

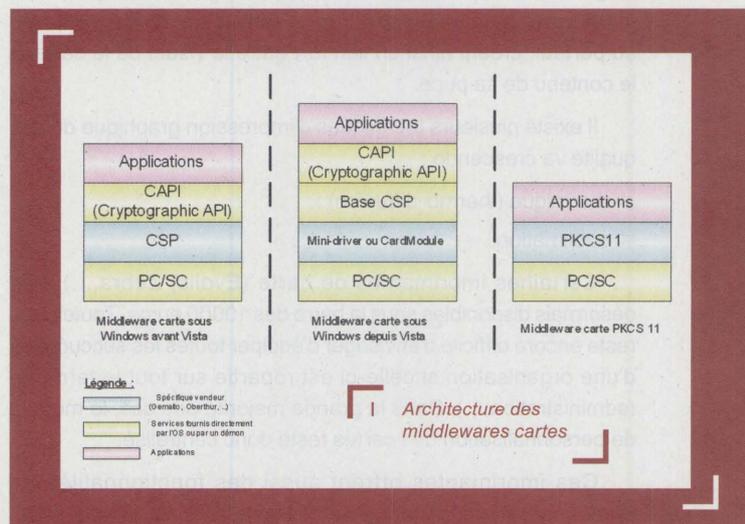
Sauf pour des fonctionnalités spécifiques qui ne sont pas offertes autrement, il y a un fort intérêt à passer par un middleware carte qui masque les spécificités des cartes aux applications et qui garantisse plus facilement le support des cartes de plusieurs industriels.

⇒ 2.1.3 Middleware cryptographique

Le middleware cryptographique permet aux applications d'utiliser des services cryptographiques (signature, chiffrement, de génération de clé, etc.) en utilisant les ressources d'une carte à puce.

Deux approches cohabitent (cf. Figure 1) :

- ⇒ Microsoft : CSP³, complètement remodelé en Base CSP et *mini-driver* avec la sortie de Vista et de Windows Server 2008 ;
- ⇒ PKCS#11 : spécifications de la société RSA.



⇒ 2.1.3.1 CSP

Les API de cryptographie Microsoft (MS CAPI⁴) contiennent des fonctions qui permettent aux applications de chiffrer ou signer numériquement des données, tout en fournissant une protection sur la clé privée de l'utilisateur. Toutes les opérations cryptographiques sont exécutées par des modules connus sous le nom de CSP (*Crypto Service Provider*), qui se présentent sous la forme d'une bibliothèque partagée.

Les paramètres supportés par le CSP dépendent des caractéristiques du composant matériel. Par exemple, le CSP ne proposera pas la génération d'une bi-clé RSA 4096 bits si la carte à puce ne le permet pas.

Les API de cryptographie fournies par Microsoft ajoutent un niveau supplémentaire d'abstraction. Elles permettent d'utiliser les ressources cryptographiques d'une carte à puce sans avoir à se préoccuper de la spécificité des commandes APDU à lui envoyer.

⇒ 2.1.3.2 PKCS#11

Le but recherché par PKCS#11 [2] est le même, mais rajoute un objectif supplémentaire : la standardisation.

Parmi les principaux industriels de la carte à puce, tous ne fournissent pas de middleware PKCS#11 pour Linux et Mac. Ceci n'est pas particulièrement bloquant sur les postes utilisateurs, qui sont souvent sous Windows, mais devient une réelle contrainte sur les serveurs *back office* basés sur des systèmes Linux.

⇒ 2.2 La personnalisation

⇒ 2.2.1 Personnalisation graphique

La carte à puce sous la forme d'une carte pleine sert également souvent de badge d'identification. Pour cette raison, on peut y imprimer des informations pertinentes comme le nom, le prénom, le numéro RH, la photo de son porteur, le logo de l'organisation, etc. Le numéro unique (numéro RH, etc.) imprimé sur la carte peut même être repris dans les certificats délivrés au porteur, créant ainsi un lien fort entre le visuel de la carte et le contenu de sa puce.

Il existe plusieurs techniques d'impression graphique dont la qualité va crescendo :

- ⇒ thermique (thermo-diffusion) ;
- ⇒ sublimation.

Certaines imprimantes de carte (Evolis, Zebra,...) sont désormais disponibles sous la barre des 10000 euros. Toutefois, il reste encore difficile d'envisager d'équiper toutes les succursales d'une organisation si celle-ci est répartie sur tout le territoire (administration,...). Dans la grande majorité des cas, le modèle de personnalisation des cartes reste donc centralisé.

Ces imprimantes offrent aussi des fonctionnalités de personnalisation électrique permettant l'encodage de la puce. Au cours de cette opération, le *KeySet GlobalPlatform*, le code PIN et le code PUK⁵ peuvent être diversifiés. Ainsi, lorsque la carte sort du chargeur et que les contacts se posent sur la puce, les signaux électriques ISO7816 sont redirigés de l'imprimante vers le PC, via une interface série dans le cas des imprimantes Zebra P430i et Evolis Dualys.

Des logiciels (BadgeMaker,...) permettent ensuite de piloter l'imprimante et de réaliser la personnalisation graphique. Ils peuvent être configurés de telle sorte qu'ils appellent un autre programme (exécutable ou dll) lorsque la carte est sous les contacts de l'imprimante.

```
int ChipInit(LPBMCHIPINFO lpci)
{return 0;}

int ChipEncodeEx(LPBMCHIPINFO lpci)
{
    if (Encoding(lpci))
        return 0;

    MessageBox(NULL,szComment,"Erreur dans l'encodage de la puce", MB_OK+MB_ICONERROR);
    return 1;
}
```

Code 1 : Exemple de dll réalisant l'encodage de la puce s'interfaçant avec le logiciel BadgeMaker

Ainsi, les opérations de personnalisation électrique et graphique sont réalisées l'une à la suite de l'autre ou de manière dé-corrélée en fonction des cas utilisateurs.

⇒ 2.2.2 Personnalisation électrique

La personnalisation électrique d'une carte PKI consiste à :

- ⇒ installer les applications (*cardlet*, autre), si cela n'est pas déjà fait ;
- ⇒ paramétrer les applications sur la carte en définissant :
 - 1 la politique de code PIN ;
 - 2 la politique de code PUK (code de déblocage) ;
 - 3 le nombre de conteneurs de bi-clés et de la taille de ces clés ;
 - 4 le jeu de clés Global Platform (KeySet) du fabricant.
- ⇒ générer les bi-clés, installer les certificats sur la carte, éventuellement importer des bi-clés sur la carte.

Déterminer le nombre de conteneurs de bi-clés est déterminant pour la PKI. En effet, si nous prenons l'exemple de l'entreprise dont les cas d'usage de la PKI sont : l'authentification, la signature et le chiffrement, trois conteneurs de clés minimums sont théoriquement nécessaires. Lorsqu'un certificat PKCS#12 est importé dans la carte, il faut également prévoir l'espace nécessaire pour héberger les certificats d'AC⁶ de la chaîne de certification.

Le code PIN est différent pour tous les utilisateurs : plusieurs scénarii de personnalisation et de fourniture du code PIN sont possibles selon le processus d'enregistrement des utilisateurs. D'un point de vue opérationnel, voici les scénarii possibles :

- ⇒ Diversification du code PIN à la personnalisation électrique et envoi du code PIN au porteur par un 2^e canal. Cette transmission est réalisée, par envoi sécurisé. Cela nécessite l'utilisation d'équipements spécifiques sur le modèle du monde bancaire (Pinelope [3], etc.).
- ⇒ L'enregistrement du futur porteur est réalisé en face à face. La carte à puce est personnalisée de telle sorte qu'elle force le changement du code PIN à la première utilisation. L'opérateur d'enregistrement demande alors au porteur de saisir son nouveau code PIN.
- ⇒ En ayant en tête les risques que cela comporte et en conformité avec le niveau de sécurité global que l'on cible, on peut aussi envisager d'envoyer le code PIN à l'utilisateur par courrier électronique.

De même, le code PUK est différent pour tous les utilisateurs. Ce code est particulièrement important pour la PKI, car il permet de faire le lien avec le système de gestion de cartes, le CMS (*Card Management System*).

L'atelier comprendra alors son propre module SAM. L'algorithme de diversification pourra être celui défini par GlobalPlatform. Il est aussi possible de créer son propre SAM avec un schéma de diversification propre.

Selon les processus d'enregistrement, la génération d'une bi-clé et la propagation de son certificat dans la carte à puce du futur porteur sont également réalisables lors de la phase de personnalisation électrique de la carte à puce.

C'est typiquement le modèle qui sera adopté par un opérateur de certification délivrant des cartes à des populations dont il n'a pas la maîtrise et dont il ne peut exiger le face à face.

⇒ 2.3 Mon PIN est bloqué !

Une carte à puce est bloquée quand l'utilisateur a dépassé le nombre maximum de tentatives de présentation du code PIN, nombre paramétré à la personnalisation de la carte.

La fonction déblocage est particulièrement sensible, car elle permet de débloquent une carte.

L'accès à cette fonction doit par conséquent être protégé par un mécanisme d'authentification.

Il s'agira vraisemblablement d'une authentification forte par certificat lorsque ce sera l'officier de sécurité ou la hotline qui réalise le déblocage. Dans le cas où le porteur réalise lui-même le déblocage, le mécanisme d'authentification est plus souvent basé sur un secret personnel (question secrète, etc.).

Plusieurs stratégies de déblocage sont possibles :

- ⇒ débloquent la carte sans changer le code PIN ;
- ⇒ débloquent la carte en forçant le changement du code PIN.

Suivant le scénario choisi, il peut être intéressant de choisir une stratégie de déblocage particulière.

Par exemple, si le porteur distant débloquent lui-même sa carte, on ne souhaite pas forcément lui faire changer son code PIN.

On préférera sans doute garder cette fonctionnalité lors d'un face à face entre le porteur et l'officier de sécurité.

Afin de recalculer un code PUK diversifié, les fonctions de déblocage nécessitent un dialogue entre la carte à puce et un module de sécurité utilisé au moment de la personnalisation de la carte : la carte SAM.

Il existe plusieurs façons d'envisager l'accès à la fonction de déblocage d'une carte :

- ⇒ déblocage en ligne ;
- ⇒ déblocage hors ligne.

⇒ 2.3.1 Déblocage en ligne/mode connecté

La figure 2 illustre l'architecture d'une solution de déblocage en ligne où le SAM est une carte à puce.

Le cas du déblocage en ligne correspond aux cas concrets suivants :

- ⇒ Le porteur a accès au SI de l'entreprise. La fonction de déblocage lui est complètement sous-traitée.
- ⇒ Le porteur est nomade : depuis l'endroit où il se trouve, il peut ouvrir une session sur son poste de travail et avoir un accès internet/intranet pour accéder à la fonction de déblocage.
- ⇒ L'officier de sécurité débloquent la carte du porteur en face à face.

Techniquement, un déblocage en ligne est réalisé directement depuis un navigateur Web (IE, Firefox, etc.). Dans la page web, s'exécute alors un composant (Applet, ActiveX) qui va réaliser les échanges avec la carte SAM distante.

Si le composant est un composant ActiveX signé, l'accès à PC/SC sur le poste distant est relativement simplifié, puisque le composant ActiveX a les droits nécessaires pour accéder aux ressources du poste.

Dans le cas d'une applet, l'accès à la couche PC/SC du poste pourra se faire au travers du *wrapper* jpc/sc.

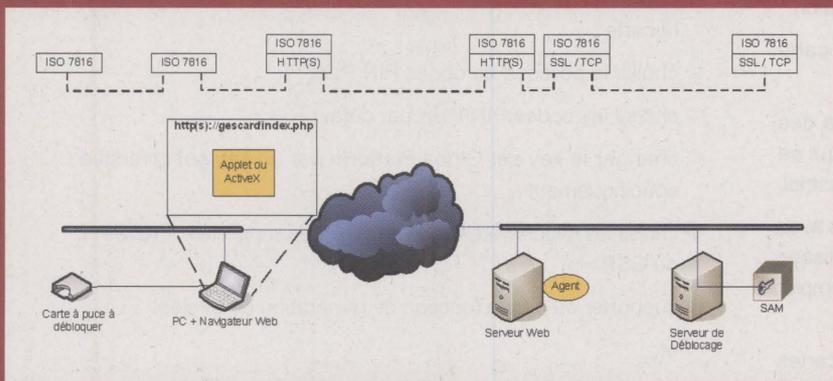
Le composant réalisant le déblocage est dans son ensemble générique. Il possède néanmoins certaines spécificités liées au type de carte.

Pour réaliser une telle fonction, des difficultés techniques existent. Par exemple, la sélection d'une applet dans la carte à débloquent par le composant s'exécutant dans la page Web peut entraîner des ruptures de la session SSL ouverte entre le poste client et le module SAM distant.

⇒ 2.3.2 Déblocage hors ligne/mode déconnecté

Le cas du déblocage hors ligne correspond aux cas réels suivants :

- ⇒ Le porteur est nomade : il peut ouvrir une session sur son poste de travail. Toutefois, il travaille en mode déconnecté.



2 Architecture d'une solution de déblocage

⇒ L'officier de sécurité débloque la carte du porteur en face à face.

Pour débloquer sa carte, le porteur/l'officier de sécurité lancera l'application installée sur le poste.

Certains fabricants de carte fournissent les outils nécessaires pour procéder au déblocage de la carte.

Afin de maîtriser la sécurité de bout en bout, il est également envisageable de développer sa propre application pour le déblocage. Une fois la carte insérée, l'application affiche le numéro de série de cette carte ainsi qu'un diversifiant (à base d'aléa, de la date et de l'heure courantes par exemple). Ces informations sont ensuite communiquées par téléphone à la hotline, avec le nom et le prénom du porteur.

Après authentification du porteur, le code PUK lui est communiqué, chiffré par le diversifiant. Ainsi, le code PUK n'est pas transmis en clair par téléphone.

⇒ 2.4 Recyclage des cartes

On entend par recyclage d'une carte le fait de réutiliser un support déjà personnalisé pour un utilisateur, afin qu'il contienne les clés et données d'un autre utilisateur.

Le retour d'une carte a lieu dans les cas suivants :

- ⇒ changement d'affectation ou d'emploi ;
- ⇒ changement de carte à puce ;
- ⇒ départ définitif (décès, démission,...) ;
- ⇒ retour d'une carte temporaire (prêt, intervenant extérieur, etc.) ;
- ⇒ etc.

Important : avant de recycler la carte, tous les certificats qu'elle contient doivent être révoqués.

Le recyclage des cartes est possible par différents moyens :

- ⇒ un outil d'administration de la carte : livré avec le middleware (GemSafe Toolbox, ACS, OCS, ActivClient,...) ou créé spécifiquement pour le projet PKI ;
- ⇒ un composant s'exécutant dans un navigateur comme pour la fonction de déblocage.

Au préalable, la fonction de recyclage ouvre un canal sécurisé GP⁹ (SecureChannel) avec la carte, puis elle exécute les actions suivantes :

- ⇒ suppression de tous les conteneurs de clés ;
- ⇒ réinitialisation du nombre de tentatives de code PIN (GetResetTryCounter...);
- ⇒ réinitialisation du code PIN par défaut.

Si d'autres informations personnelles sont contenues dans la carte, il peut être envisagé de supprimer les applets de la carte, de les réinstaller, de les réinstancier et de les personnaliser pour le nouvel utilisateur.

Une attention toute particulière sera portée aux données chiffrées qui, si rien n'est fait, seront alors irrécupérables. Selon les cas, il pourra être procédé au recouvrement de la clé de déchiffrement dans un nouveau support, au trans-chiffrement des données, préalablement au retrait du certificat de confidentialité ou simplement rendre les données inaccessibles.

⇒ 3. Intégration de la carte et de la PKI

⇒ 3.1 AE¹⁰ ou guichet d'enregistrement

Dans le modèle classique d'enregistrement de porteur au sein d'une entreprise, l'opérateur en charge de l'enregistrement accède à la fonction d'enregistrement après s'être authentifié. Nous nous intéresserons plus précisément à la situation où l'opérateur s'authentifie avec sa carte.

La fonction d'enregistrement va permettre de générer la bi-clé correspondant au certificat du futur porteur. Elle doit pouvoir déterminer quelle est la carte de l'opérateur de celle du futur porteur afin d'éviter toute écriture sur la carte de l'opérateur.

Avant de générer une bi-clé dans la carte, la fonction d'enregistrement doit d'abord créer un conteneur dans la carte. Pour cela, elle va utiliser le middleware cryptographique (PKCS#11 ou CSP).

Maîtriser le nom du conteneur de clés créé sur la carte s'avère particulièrement important pour les fonctions de recyclage de carte, puisqu'on connaîtra à ce moment-là les conteneurs à détruire.

Afin d'éviter à l'utilisateur de choisir le middleware à utiliser avec sa carte, une fonction d'enregistrement intuitive détectera automatiquement le middleware à utiliser ou permettra de forcer le middleware à utiliser avec la carte.

Après validation de la demande de certificat, lorsque le nom du middleware a été stocké dans le dossier du porteur, la propagation du certificat dans la carte est automatiquement réalisée. Ceci limite les interactions avec l'opérateur d'enregistrement et réduit les manipulations source potentielle d'erreur.

Dans le dossier du futur porteur de certificats, un lien sera fait entre les certificats et la carte où ils seront hébergés.

Le numéro de série de la carte sera alors récupéré. Il est à noter qu'un CSP ne permet pas de récupérer le numéro de

série de la carte. De plus, les numéros de série ont des formats différents : courts ou longs. Un GetCPLC¹¹ (...) retournera le numéro de série de la carte sur 4 octets, tandis qu'un middleware PKCS#11 retournera un numéro de série plus long. Il est à noter que certains fournisseurs proposent des middlewares PKCS#11 dont le format long est complètement différent du format court. D'autres fournisseurs proposent des cartes dont le format court et un sous-ensemble du format long retourné par le middleware PKCS#11. Il faut donc faire attention.

Lorsqu'on utilise la MS CAPI, le choix du middleware se fait à partir de l'ATR de la carte. En effet, dans la base de registre Windows au niveau de chaque CSP, la liste des ATR des cartes supportées est stockée (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Calais\SmartCards).

Ceci peut être particulièrement problématique avec certaines cartes.

En effet, certains éditeurs, tels qu'ActivIdentity, utilisent des cartes de fournisseurs variés dont les ATR sont déjà reconnus par le middleware de la carte déjà installé sur l'OS.

Une AE bien conçue prendra en compte tous ces paramètres pour être la plus indépendante possible de la solution carte qui aura été choisie.

⇒ 3.2 Déploiement du middleware

⇒ 3.2.1 Profil utilisateur

Le middleware n'est pas la seule brique nécessaire au bon fonctionnement d'une solution carte.

Des outils d'administration utilisant ce middleware proposent généralement les fonctions suivantes :

- ⇒ débloquage de la carte (avec ou sans changement de code PIN, de code PUK) ;
- ⇒ propagation/suppression des certificats dans les magasins de certificats lors des événements de retrait et d'insertion de la carte ;
- ⇒ configuration du comportement du middleware ;
- ⇒ visualisation des certificats présents dans la carte permettant le recyclage des cartes (suppression de tous les certificats, etc.), l'importation de certificats au format PKCS#12 ;
- ⇒ création de package MSI¹² pour le déploiement des pilotes des lecteurs, du middleware, des outils, ...

On distinguera plusieurs profils utilisateurs, par exemple : administrateur (opérateur d'enregistrement, officiers de sécurité, etc.), utilisateur final, autre. En fonction de ce profil, il conviendra donc de définir quels sont les outils que l'on souhaite déployer sur les postes utilisateur et comment les déployer.

⇒ 3.2.2 Stratégie de déploiement

En fonction de l'OS cible, des applications préconisées (Firefox est-il le navigateur par défaut ?, etc.) de l'organisation, des *add-on*

sont parfois nécessaires. En effet, en fonction des navigateurs, la propagation et la destruction des certificats contenus dans une carte à puce respectivement à l'insertion et au retrait est réalisée différemment. Le middleware doit donc être configuré de telle sorte que les événements sur la carte déclenchent les comportements appropriés du middleware.

Ceci est surtout vrai pour les applications qui utilisent le CSP pour interagir avec la carte. Ces applications doivent retrouver les certificats propagés dans le magasin de certificat (du poste, de l'utilisateur, autre) géré par l'OS.

Pour les applications reposant directement sur PKCS#11, l'interaction avec la carte, puis la gestion des certificats s'y trouvant est propre à l'application. C'est typiquement le cas de Mozilla Firefox.

Pour accéder à la carte, une solution envisageable est un agent s'exécutant en tâche de fond qui se chargera de créer le périphérique de sécurité en utilisant les NSS¹³ Security Tools [6].

```
modutil.exe -add <Nom du périphérique de sécurité> -libfile <chemin
d'accès à la librairie> -dbdir <répertoire de Firefox> -force
ex : modutil.exe -add Axalto -libfile "c:\Program Files\Axalto\Access
Client\v5\xtCk.dll" -dbdir "c:\Documents and Settings\<Nom de
l'utilisateur>\Application Data\Mozilla\Firefox\Profiles\c6j0xvc4.default"
-force
```

Code 2 : Ajout d'un périphérique de sécurité (carte à puce) sous Mozilla-Firefox

Au final, deux types de déploiements sont possibles :

- ⇒ Un déploiement de package MSI via les GPO¹⁴. Les certificats d'AC de la PKI, le code signé (composant pour l'enregistrement, le déblocage de carte, etc.) et tout autre élément utile pourront également être déployés par les GPO.
- ⇒ Une installation manuelle : les outils sont packagés et mis à disposition (site web de téléchargement, autre). Ils sont installés par les utilisateurs.

⇒ 3.3 Services et ergonomie : oubli, perte et vol

Les principaux usages de la carte à puce comme support dans un service reposant sur une PKI sont :

- ⇒ authentification ;
- ⇒ signature ;
- ⇒ chiffrement.

L'utilisation d'une carte à puce amène à se poser la question de la disponibilité en cas de perte, de vol, d'oubli de la carte ou de blocage du PIN.

On voit ainsi que la disponibilité de la carte de l'utilisateur est particulièrement critique pour les usages suivants :

- ⇒ Authentification système : il n'est pas envisageable de bloquer l'accès au SI aux utilisateurs.
- ⇒ Chiffrement des données : l'utilisateur doit être à même d'accéder aux données personnelles qu'il a chiffrées et qu'il ne peut déchiffrer qu'à l'aide de sa carte.

Les cas de blocage, d'oubli, de perte et de vol de carte doivent donc impérativement être traités dans le cadre d'un projet de PKI ; le cas de déblocage étant particulièrement

caractéristique d'une carte à puce, il a déjà été traité. Nous nous attacherons à décrire ici plus particulièrement les autres cas.

L'ergonomie du service ne doit pas être en reste pour garantir une acceptation de la technologie carte auprès des utilisateurs. À ce titre, le cas de la saisie du code PIN est caractéristique : le fait de taper le code PIN à chaque utilisation de la carte peut s'avérer un facteur de rejet si la fréquence d'utilisation est importante.

D'un point de vue sécurité, même si nous ne nous y attarderons pas, la signature n'est pas moins critique puisqu'un vol de carte avec récupération du code PIN du porteur, permet de signer des documents ayant valeur juridique.

⇒ 3.3.1 Je n'ai plus ma carte : comment faire pour m'authentifier ?

Les bonnes questions à se poser lorsqu'on évalue une solution carte pour du Smartcard logon, et ce, quel que soit le facteur de forme, sont les suivantes :

- ⇒ Type des machines cibles (DELL, HP,...) : des comportements anormaux de certains middlewares cartes peuvent nécessiter la mise à jour du BIOS du PC.
- ⇒ Comportement au retrait et à l'insertion de la carte.
- ⇒ Comportement lorsqu'on insère une clé *USB Mass Storage* alors que la session utilisateur est déjà ouverte : curieusement, des coupures de session intempestives sont occasionnées avec certaines cartes. Le middleware ou le pilote de la carte peuvent être mis en cause.
- ⇒ Comportement lorsqu'on insère une deuxième carte (cas du poste de l'opérateur d'enregistrement).
- ⇒ Présence d'un hub USB (interne ou externe) et répétition des tests précédemment cités.

Lorsque des anomalies persistent, il est particulièrement intéressant de dé-corréler la problématique lecteur de la problématique carte. Par exemple, si la carte est une carte au format SIM, les lecteurs Omnikey6121 ou SCM3320 présenteront des résultats différents aux tests cités qu'un lecteur GemPCKey...

...Les cas de blocage, d'oubli, de perte et de vol de carte doivent donc impérativement être traités dans le cadre d'un projet de PKI...

À noter, certains résultats déconcertants aux tests listés ci-dessus : sur un hub USB (interne ou externe), l'insertion d'une clé USB Mass Storage alors qu'une carte est déjà insérée provoque un verrouillage de la session Windows ! Si on insère la clé USB au bout d'une rallonge courte (0,9m) raccordée au HUB, l'anomalie persiste ou disparaît selon les cas. Le même test au bout d'une rallonge longue (1,8m) fait complètement disparaître l'anomalie.

Certaines solutions cartes présentent donc des problèmes matériels...

Les middlewares ne sont pas non plus en reste : affichage du nombre de certificats ne correspondant pas au nombre réel de certificats dans la carte, message d'erreur lors de l'insertion d'une carte pour réaliser l'authentification en Smartcard logon, affichage d'une fenêtre de réinitialisation de code PIN lors de l'insertion d'une deuxième carte non vierge, etc.

⇒ 3.3.1.1 Solution de secours : cas d'oubli, perte ou vol

Dans le cadre d'un service d'authentification forte au système Windows (*windows logon*), la carte revêt un aspect extrêmement critique (en plus de la nécessaire disponibilité des CRL¹⁵ : accessibilité, gestion du cache, etc.)

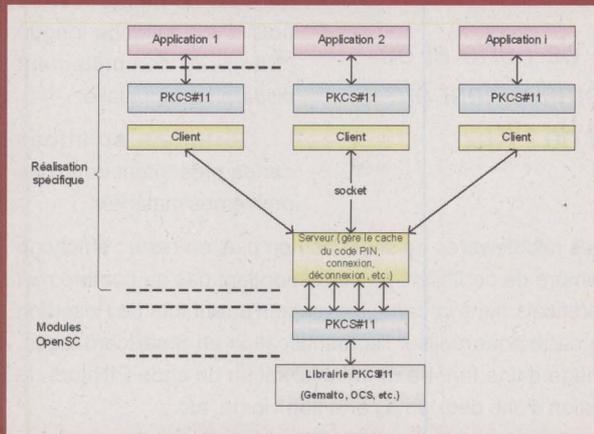
La mise en place d'alternatives à l'authentification système en mode nominal est donc nécessaire. Ces alternatives doivent tenir compte du nomadisme des utilisateurs (mode connecté ou non au SI) lors de l'authentification.

Parmi ces alternatives, on retrouve les solutions suivantes :

- ⇒ Carte de secours : cette solution est particulièrement appropriée à des utilisateurs VIP dont la mobilité et la disponibilité compliquent la tâche de renouvellement de carte. Ils auront en leur possession une deuxième carte prête à l'emploi.
- ⇒ Carte temporaire : avant de déclarer une carte complètement perdue, il est préférable de proposer une carte avec une durée d'utilisation limitée, avec un certificat valable pour la journée par exemple, qui permettra à l'utilisateur de passer normalement sa journée de travail.
- ⇒ Solution d'authentification de secours : il est envisageable de proposer une alternative temporaire à l'authentification système par carte. On peut envisager une solution d'authentification de secours qui peut se présenter sous la forme d'un login/mot de passe ou d'un OTP¹⁶.

Sous Linux, le module `pam_pkcs11`¹⁷ est dépendant de la bibliothèque `pkcs11`. Or, à la différence du CSP, `PKCS#11` n'a pas pour but d'apporter la facilité d'utilisation : typiquement, le CSP propose optionnellement l'implémentation d'une fonction de cache du code PIN qui évite à l'utilisateur de saisir plusieurs fois son code PIN, ce qui n'est pas le cas avec `PKCS#11`.

Pour apporter de la convivialité d'utilisation, il faut donc mettre en place une solution de cache de code PIN. La figure 3 présente un exemple de solution pouvant être mise en place pour combler ces manques sous Linux.



3 Architecture d'une solution de gestion de cache du code PIN

3.3.2. Je ne peux pas déchiffrer mes documents sans ma carte...

3.3.2.1 Solution de secours : cas d'oubli, perte ou vol

Les solutions professionnelles de chiffrement de données intègrent des solutions de recouvrement de données dans le cas où la carte à puce ne peut pas être utilisée (perte, vol, oubli).

Attention, dans tous les cas, il conviendra de définir clairement les procédures amenant à l'accès en clair aux données d'un utilisateur, que ce soit par lui-même, ou, dans des cas plus exceptionnels, par des tiers (absence prolongée, décès, enquête judiciaire, renouvellement de certificat de confidentialité, ...).

Plusieurs solutions peuvent être envisagées :

⇒ Dans le cas où la clé privée de déchiffrement a été séquestrée au moment de sa création par l'AC, l'utilisateur peut demander le recouvrement de sa clé dans une nouvelle carte.

⇒ Dans le cadre d'un système centralisé, on peut imaginer que les données soient chiffrées avec une seconde clé publique : la clé d'autorité du service (dans la pratique, c'est une clé symétrique qui est chiffrée avec les deux clés publiques ; cette clé symétrique étant ensuite utilisée pour chiffrer les données). Ainsi, en cas de besoin, la clé privée de déchiffrement de l'utilisateur n'étant pas disponible, il est possible d'enclencher une procédure aboutissant à l'accès aux données en clair.

⇒ Autre méthode de protection de la clé de chiffrement, cette fois encore dans le cadre d'un système centralisé : la clé de chiffrement de la partition est chiffrée par un autre mécanisme (logiciel, etc.). Ceci permet de déchiffrer les données sans la carte à puce. L'accès à une telle fonctionnalité doit être restreint, entre autres pour les raisons suivantes :

- 1► L'utilisation de cette solution réduit le niveau de sécurité global.
- 2► Les utilisateurs favorisant toujours les solutions les plus simples. Cette solution alternative pourrait bien devenir de fait la solution nominale, ce qui n'est clairement pas le but lorsqu'on a déployé une solution à base de cartes à puce.

Dans ce dernier cas, il est possible d'envisager des solutions d'authentification intimement liées aux solutions de chiffrement. Par exemple, sous Linux, les modules pam sont appelés séquentiellement et prennent en compte le bon déroulement du module pam précédent. On peut envisager par exemple l'enchaînement du module `pam_pkcs11` pour l'authentification système et du module `pam_mount` pour le montage de la partition chiffrée. Le module `pam_mount` prendra alors en compte le résultat de l'étape précédente avant de monter la partition, etc.

Dans ce cas, il est possible d'imaginer une solution d'authentification de secours, appelons la `pam_secours`, qui permettra au module `pam_mount` de monter tout de même la partition chiffrée.

```
auth    sufficient    pam_secours.so use_authok try_first_pass
auth    sufficient    pam_pkcs11.so use_authok try_first_pass
...
session optional     pam_mount.so
```

Code 3 : Exemple de configuration /etc/pam.d/system-auth

Conclusion

La mise en place d'une solution carte dans un projet PKI comporte plusieurs aspects :

⇒ Organisationnel : une carte à puce est avant tout un objet matériel, donc perceptible pour les utilisateurs. Ces derniers

doivent être impliqués pour éviter des levées de bouclier (impression de la photo sur la carte, etc.) et ainsi augmenter les chances de réussite du projet. La résistance au changement est un facteur d'échec à l'adoption de la technologie carte.

L'accompagnement des utilisateurs doit être réalisé à travers des séances de formation. Au-delà de cette problématique d'acceptation, le volet organisationnel doit impérativement prendre en compte les différentes procédures nécessaires d'une part à la gestion du cycle de vie des cartes (délivrance, déblocage, reprise). D'autre part, les procédures pour couvrir des cas d'exception tels que le vol, la perte ou l'oubli.

- ⇒ Fonctionnel : les usages de la carte à puce doivent être déterminés en amont (SmartCard logon, VPN¹⁸, signature, chiffrement, etc.). En effet, certains cas d'usage doivent s'accompagner de mesures techniques et organisationnelles pour être réellement opérants (le meilleur exemple : le chiffrement de données).
- ⇒ Technique : de nombreuses solutions cartes existent sur le marché, intégrant toutes grosso modo les mêmes fonctionnalités. Les tests réalisés dans notre laboratoire ont démontré que toutes n'ont pas le même degré de maturité. Qualifier techniquement la solution choisie avant de commencer un projet s'avère être une étape capitale.
- ⇒ Si on ne considère pas la technologie e-gate, désormais obsolète, les solutions format clé ont des degrés de maturité très variés, moins matures que les solutions carte pleine.

Aussi, sur le plan technique, le maître-mot est donc : « tester, tester et encore tester ! »

La technologie carte évolue pour proposer des solutions adaptées aux demandes du marché. Typiquement, certains acteurs proposent :

- ⇒ des solutions d'authentification OTP, conformes aux spécifications OATH¹⁹, en complément de la solution PKI (Aladdin, ActivIdentity, Gemalto, Kobil,...) ;
- ⇒ des solutions de SSO²⁰ carte (Idactis, Aladdin, etc.) : login/mot de passe sont stockés dans la carte, l'accès pouvant être contrôlé par code PIN et/ou par des données biométriques ;
- ⇒ des solutions de stockage chiffrant à la volée les données (GemSafe Guardian, etc.), pour certaines hébergeant des applications portables (Kobil, etc.).

Bien qu'innovantes et ouvrant de nouvelles perspectives pour les usages de la carte à puce, ces nouvelles offres ne garantissent pas un succès à la carte dans ces domaines. L'avenir ne se limitera-t-il pas au contrôle d'accès ?

Pour le stockage de données chiffrées, des initiatives comme celles du TCG²¹ sont plus avancées, d'autant plus que les PC des constructeurs, membres du TCG, viennent nativement avec un composant TPM²².

L'avantage de ce type de composant est que son utilisation n'est pas limitée au chiffrement de données : il s'inscrit parfaitement dans la stratégie plus globale autour du contrôle d'accès réseau (802.1X).

Pour garantir un très haut niveau de sécurité en environnement mobile, d'autres initiatives répondent déjà à ce besoin [7] à un coût dont l'équation économique reste encore à valider. ■

Notes

- ¹ Personal Identification Number
- ² Personal computer/Smart Card
- ³ Cryptographic Service Provider
- ⁴ Microsoft Cryptographic API
- ⁵ Personal Unblocking Key
- ⁶ Autorité de Certification
- ⁷ Security Application Module
- ⁸ Identification, Authentication and electronic Signature
- ⁹ Global Platform
- ¹⁰ Autorité d'Enregistrement
- ¹¹ Card Production Life Cycle
- ¹² Microsoft Installer
- ¹³ Network Security Services
- ¹⁴ Group Policy Object
- ¹⁵ Certificate Revocation List
- ¹⁶ One Time Password
- ¹⁷ Pluggable Authentication Module
- ¹⁸ Virtual Private Network
- ¹⁹ Open AuTHentication
- ²⁰ Single Sign-On
- ²¹ Trusted Computing Group
- ²² Trusted Platform Module



Remerciements

Nous tenons tout particulièrement à remercier Gaëlle Donnette et Vincent Guyot pour leur relecture minutieuse, Joël Jurdith et Luc Creti pour les échanges que nous avons eus sur ce sujet.



Références

- ⇒ [1] MUSCLE, <http://www.linuxnet.com/middle.html>
- ⇒ [2] RSA Laboratories, *PKCS#11 : Cryptographic Token Interface Standard*, version 2.20, 2004.
- ⇒ [3] Imprimante Pinelope, http://www.verisec.com/documents/pinelope_productsheet.pdf
- ⇒ [4] Global Platform, <http://www.globalplatform.org>
- ⇒ [5] Synergies – Administration Électronique, http://www.synergies-publiques.fr/article.php?id_article=275
- ⇒ [6] Mozilla, <http://www.mozilla.org/projects/security/pki/nss/tools/>
- ⇒ [7] Globull, <http://www.myglobull.com/>

JAVA CARD (U)SIM ET APPLICATIONS SÉCURISÉES SUR TÉLÉPHONES MOBILES

■ mots clés : Java Card / (U)SIM / Java embarqué / téléphone mobile / sécurité

Les cartes à puce sont aujourd'hui considérées comme des ressources sûres du fait de leur structure interne et des validations qu'elles subissent tout au long de leur cycle de vie. Il est donc possible de les utiliser comme éléments de base pour sécuriser différents types d'environnements : accès physiques à des locaux, transactions bancaires, systèmes embarqués, etc. Dans cet article, nous expliquons comment utiliser les cartes (U)SIM Java des téléphones mobiles afin d'apporter de la sécurité, en termes de confidentialité des données et des communications, dans le fonctionnement des applications développées pour ces matériels. Nous présentons les différentes technologies qui interviennent (Java Card, (U)SIM, Java ME) ainsi que la façon de les combiner. Nous terminons par un exemple d'application qui utilise tous les éléments présentés.

➔ 1. Java Card

➔ 1.1 Principe

Une carte à puce est une pièce de plastique contenant un circuit électronique intégré et en particulier un processeur, ce qui lui permet donc de manipuler des données. Elle peut être programmée pour réaliser certaines tâches et peut également stocker de l'information.

En fonction de leur OS (*Operating System*), les cartes à puce se partagent en deux grands groupes :

- ➔ à système de fichier fixé : la structure de fichiers embarquée (donc les applications) ne peut pas être modifiée et ces cartes offrent donc un jeu de fonctions fixe, défini par le constructeur. C'est le cas de certaines cartes bancaires.
- ➔ à système dynamique d'application : on peut ajouter (déployer) des applications sur ces cartes, dynamiquement, de façon contrôlée et sûre. Elles peuvent donc également être mises

à jour. C'est le cas de certaines (U)SIM ou des cartes Java. On parle alors de cartes multi-applicatives.

Quel que soit le cas, ces cartes à puce sont considérées comme des éléments sûrs, du fait que l'ajout ou la modification d'un élément de leur structure interne est soit non autorisé (système de fichier fixé), soit encadré par des procédures strictes.

La carte Java est une carte à puce particulière, multi-applicative, qui utilise la technologie Java Card [1]. Il est possible d'y charger et d'y exécuter des applications écrites dans un dérivé du langage Java. De façon pratique, la machine virtuelle, appelée JCVM (*Java Card Virtual Machine*) dans la technologie Java Card, est découpée en deux parties : la première qui contient l'interpréteur de *bytecode* est située sur la carte, et la deuxième qui contient les autres fonctionnalités classiques d'une machine virtuelle (convertisseur, chargement de classes, vérification du *bytecode*) est située hors de la carte.

⇒ 1.2 Les applets

Une *applet* [2] est une application écrite dans un dérivé du langage Java qui est chargée et qui s'exécute sur une carte Java. Conformément au découpage de la JCVM décrit ci-dessus, la carte Java contient un JCRE (*Java Card Runtime Environment*). Celui-ci fournit des mécanismes de sécurité assurant entre autres la séparation entre le système de la carte et les applications qui s'y exécutent. Le JCRE gère les ressources de la carte, l'exécution et la sécurité des applets. Les applets interagissent avec le JCRE à travers des API spécifiques. Chaque applet embarquée sur la carte est isolée des autres par un pare-feu (firewall) et dispose de son propre contexte d'exécution. Toute applet et le paquetage (*package*) à laquelle elle appartient sont identifiés par un numéro unique appelé AID (*Application ID*).

Il est important de noter que le langage Java Card utilisé pour développer les applets est un sous-ensemble du langage de programmation Java. Il y a donc certaines caractéristiques de Java qui ne sont pas disponibles en Java Card. Ce sont par exemple les types *char*, *string*, *double* et *float*, les tableaux à plusieurs dimensions, le chargement dynamique des classes et les *threads*. Certaines de ces limitations sont levées avec la version 3 de Java Card, mais nous utilisons ici la version 2 qui est celle essentiellement disponible actuellement.

⇒ 1.3 Exemple de code commenté

Le code d'une applet respecte la structure d'un code Java classique. La classe principale dérive de la classe `javacard.framework.Applet`.

Le listing 1 présente un code simple qui ne réalise effectivement aucune opération, mais qui contient les éléments principaux que l'on retrouve dans une applet, à savoir :

- ⇒ le **constructeur**, ici `example_applet` (ligne 16), utilise la méthode `register` qui permet de déclarer ou d'enregistrer l'applet auprès du JCRE de la carte ;
- ⇒ la **méthode** `install` (ligne 28) provoque la création d'une instance de l'applet par le JCRE ;
- ⇒ la **méthode** `process` (ligne 58) traite toutes les communications venant du CAD (*Card Acceptance Device*, le lecteur) qui contient la carte ;
- ⇒ la **méthode** `select` (ligne 38) conduit le JCRE à notifier l'applet du fait qu'elle a été sélectionnée. Le JCRE lui redirigera ensuite les requêtes qui lui sont destinées ;
- ⇒ la **méthode** `deselect` (ligne 47) notifie l'applet qu'une autre applet va être sélectionnée (rappelons que sur les cartes multi-applicatives plusieurs applets peuvent être chargées).

Le lecteur souhaitant effectivement exécuter un code d'applet pourra se référer à la section 4.4.

Java and all Java-based marks are trademarks or registered trademarks of Sun microsystems, Inc. in the United States and other countries. The authors are independent of Sun microsystems, Inc. All other marks are the property of their respective owners.

```

1. package demo_applet ;
2.
3. // paquetages importés
4. // importation spécifique pour l'API Javacard
5. import javacard . framework .*;
6.
7. public class example_applet extends javacard . framework
8. Applet {
9.
10. /**
11. Constructeur par défaut
12. @param baBuffer paramètres d'installation
13. @param sOffset offset de départ
14. @param bLength longueur en octets
15. */
16. protected example_applet ( byte [] baBuffer , short sOffset , byte
17. bLength ) {
18.     register ( baBuffer , ( short ) ( sOffset + 1 ) , ( byte )
19.     baBuffer [ sOffset ] );
20. }
21.
22. /**
23. Methode pour installer l'applet.
24. @param baBuffer paramètres d'installation
25. @param sOffset offset de départ
26. @param bLength longueur en octets
27. */
28. public static void install ( byte [] baBuffer , short sOffset , byte
29. bLength ) {
30.     // création d'une instance de l'applet
31.     new example_applet ( baBuffer , sOffset , bLength ).
32.     register ();
33. }
34.
35. /**
36. @le booléen retourné doit toujours être true .
37. */
38. public boolean select () {
39.     /** @todo : ACTION DE SELECTION A EXECUTER */
40.     // retourne l'état de la désélection
41.     return true ;
42. }
43.
44. /**
45. Invoquée par le système dans le processus de désélection.
46. */
47. public void deselect () {
48.     /** @todo : ACTION DE DESELECTION A EXECUTER*/
49. }
50.
51. /**
52. Methode qui traite une APDU entrante
53. @voir APDU
54. @param apdu APDU entrante
55. @exception ISOException avec les octets de réponse
56. defini dans l'ISO 7816-4
57. */
58. public void process ( APDU apdu ) {
59.     /** @todo : METTRE L'ACTION A EXECUTER ICI */
60. }
61.
62. }

```

Listing 1 : Code applet



⇒ 2. Java Card sur (U)SIM

⇒ 2.1 Présentation

L'(U)SIM est une application spécifique [3] installée sur une carte à puce amovible (généralement insérée dans un téléphone mobile) et qui permet de se connecter à un réseau GSM/3G. Elle fournit aussi des mécanismes d'authentification et de stockage des profils des utilisateurs.

Par extension, on appelle carte Java (U)SIM une carte Java qui embarque une (U)SIM. En d'autres termes, c'est une carte à puce sur laquelle on peut naturellement charger et exécuter des applets Java Card, être connecté à un réseau GSM/3G et enfin avoir accès à certaines informations du profil utilisateur.

⇒ 2.2 Spécificités

L'association des technologies Java Card et (U)SIM dans une carte permet d'avoir accès dans les applets embarquées, grâce

à un jeu d'API spécifiques, à certaines fonctionnalités liées aux (U)SIM. On peut par exemple accéder au système de fichiers (fichier des contacts ou fichier des SMS), manipuler le code PIN, etc.

De façon générale, les cartes à puce ont un caractère passif c'est-à-dire qu'elles fonctionnent en mode client serveur, répondant aux requêtes provenant du CAD (*Card Acceptance Device*, le téléphone mobile dans notre cas) dans lequel elles sont insérées. Il existe des cartes dites « actives » qui envoient des commandes à exécuter au téléphone mobile. Ces commandes sont dites « proactives ». Il est possible avec ces commandes d'afficher des textes sur l'écran du téléphone, de lancer l'émission d'un appel, d'expédier des SMS, etc.

Tous ces éléments se retrouvent dans des applets Java Card spécifiques définies autour de l'(U)SAT. L'(U)SAT (*(U)SIM Application Toolkit*, standard 3GPP 31.111) permet donc à un applet d'utiliser la proactivité, et les API d'accès aux fonctionnalités des cartes (U)SIM.

⇒ 3. Java embarqué sur téléphone mobile

⇒ 3.1 Principe

Java embarqué est construit autour de la plate-forme Java ME (*Java Micro Edition*) qui est une technologie permettant de développer des applications pour les appareils de petite taille et donc à ressources réduites (PDA, téléphone mobile, etc.).

Plus précisément, Java ME [4] est un ensemble de technologies et de spécifications offrant un environnement d'exécution Java pour petits appareils. Les appareils cibles sont des équipements ayant des ressources limitées en matière de mémoire, de puissance de traitement, de capacité d'affichage et d'autonomie en énergie. Java ME est basé sur trois éléments :

⇒ **une configuration** qui fournit les éléments de base et une machine virtuelle compatible avec un large ensemble d'équipements. Java ME possède deux configurations à savoir la CLDC (*Connected Limited Device Configuration*) pour les téléphones mobiles et la CDC (*Connected Device Configuration*) pour les PDA.

⇒ **un profil** qui est lié à une configuration et qui propose des API ciblant une gamme éventuellement plus limitée d'équipements. On trouve par exemple la MIDP qui est un profil possible pour la configuration CLDC et la *Foundation Profile* pour la CDC. Ce sont les profils les plus répandus

⇒ **des paquetages optionnels** qui sont un ensemble d'API très spécifiques et que l'on ne retrouve pas nécessairement dans toutes les configurations.

Comme indiquée ci-dessus, la configuration CLDC est spécialement conçue pour les appareils tels que les téléphones mobiles et elle est associée à la MIDP (*Mobile Information Device Profile*) qui comprend un ensemble d'API permettant de développer les applications Java ME. Nous utilisons la version 1.0 de CLDC et la version 2.0 de MIDP.

⇒ 3.2 Les midlets

Une *midlet* est une application développée en utilisant la plateforme Java ME et qui s'exécute sur des appareils de petite taille, comme les téléphones portables. Un fichier JAD (*Java Application Descriptor*) peut également être associé à une midlet ; il fournit des informations complémentaires concernant l'application (signature de la midlet par un certificat, API particulières utilisées, autorisations données à la midlet, etc.).

Une midlet est censée respecter le principe du « *write once, run everywhere* ». Cela signifie que toute midlet développée en respectant les spécifications d'une configuration et d'un profil donnés doit pouvoir s'exécuter sur tout équipement mobile compatible avec cette configuration et ce profil. Les midlets peuvent être téléchargées et installées directement par les

utilisateurs de téléphones mobiles compatibles avec Java ME à partir de sites web ou de serveurs dédiés.

⇒ 3.3 Exemple de code commenté

Le code d'une midlet reprend la structure d'un code Java classique. La classe principale dérive de la classe `javax.microedition.Midlet`.

Le Listing 2 présente le code d'un « *hello world* » qui contient tous les éléments essentiels que l'on retrouve dans une midlet :

- ⇒ le **constructeur**, ici `exemple_midlet` (ligne 13), contient les éléments d'initialisation de la méthode d'affichage (invocation de `Display.getDisplay(this)`).
- ⇒ la **méthode** `startApp` (ligne 25) est invoquée au lancement de l'application.
- ⇒ la **méthode** `pauseApp` (ligne 22) est invoquée lors de la mise en pause de l'application (par la méthode `NotifyPaused()`).
- ⇒ la **méthode** `destroyApp` (ligne 19) est invoquée lorsque l'application se termine, le paramètre `unconditional` de cette méthode précise si les ressources allouées à la midlet doivent être libérées lors de sa terminaison.

On notera également la présence de deux autres méthodes : `showScreen` (ligne 32) qui permet de porter à l'écran les éléments d'affichage construits par ailleurs et `display_alert` (ligne 37) qui est utilisée pour visualiser le message « Hello world ! »

```

1. package demo_midlet ;
2.
3. import javax . microedition . midlet . MIDlet ;
4. import javax . microedition . midlet .
5. MIDletStateChangeException ;
6. import javax . microedition . Icdui . * ;
7. import java . util . * ;
8.
9. public class exemple_midlet extends MIDlet {
10. private Display display ;
11. private Form form ;
12.
13. public exemple_midlet () {
14.     display = Display . getDisplay ( this ) ;
15.     // création du "form" principal avec ses composants
16.     form = new Form ( " Midlet Example " ) ;
17. }
18.
19. protected void destroyApp ( boolean unconditional ) {
20. }
21.
22. protected void pauseApp () {
23. }
24.
25. protected void startApp () {
26.     showScreen () ;
27.     display_alert ( AlertType . INFO , " INFO " , "Hello world !" ,
28.     5000 ) ;
29. }
30. }
31. // affichage de l'écran principal
32. public void showScreen () {
33.     display . setCurrent ( form ) ;
34. }
35. }
36. // affichage d'un message d'alerte pendant timeout millisecondes
37. public void display_alert ( AlertType type , String title , String text ,
38. int timeout ) {
39.     Alert alert = new Alert ( title , text , null , type ) ;
40.     alert . setTimeout ( timeout ) ;
41.     display . setCurrent ( alert ) ;
42. }

```

Listing 2 : Code midlet

⇒ 4. Communication mobile/carte (U)SIM

⇒ 4.1 Principe

Rappelons que notre objectif est d'expliquer comment utiliser des cartes (U)SIM pour éventuellement sécuriser des applications sur téléphone mobile. Il faut donc comprendre les spécifications à exploiter pour faire communiquer ces deux matériels. Le principe de base est le même que pour une communication entre une carte à puce classique et un CAD, comme cela est défini par la norme ISO 7816-4.

Le protocole d'échange avec une carte à puce se fait en mode série et *half-duplex* de la façon suivante. La carte reçoit des commandes à exécuter via des APDU (*Application Processing Data Unit*). Une APDU est une suite d'octets respectant une norme précise formalisant la communication CAD/carte à puce. Elle comporte les champs suivants :

- ⇒ **CLA**, octet de classe, généralement spécifique à l'application dans laquelle elle est utilisée ;
- ⇒ **INS**, octet d'instruction, définit la commande à exécuter ;

- ⇒ **P1**, paramètre 1, sur 1 octet, lié à l'instruction ;
- ⇒ **P2**, paramètre 2, sur 1 octet, lié à l'instruction ;
- ⇒ **Lc**, sur 1 octet, taille des données ;
- ⇒ **Data**, 0-255 octets, données.
- ⇒ **Le**, définit la longueur des données attendues dans la réponse à cette commande

Dans la **figure 1** (page suivante), c'est la machine à laquelle le CAD est connecté (on dira simplement le CAD) qui initie la communication APDU. La commande est expédiée par le CAD, puis exécutée par la carte qui retourne un code (le *status word*) composé de deux octets (qui valent 90 et 00 en cas de succès de l'exécution de la requête).

Dans le cas d'une commande proactive, c'est-à-dire lorsqu'une carte (U)SIM initie une opération (comme vu précédemment), le schéma des échanges APDU est alors différent (**Figure 2**).

La carte indique au CAD (en réalité le CAD fait du *polling* sur la carte) qu'il y a une commande à exécuter, ce dernier récupère l'information et exécute l'action demandée. Le résultat de l'exécution est ensuite retourné à la carte.

4.2 Communication midlet/applet

Dans la configuration qui nous intéresse ici, le téléphone mobile embarque une midlet et la carte (U)SIM une applet. Il s'agit donc de réaliser une communication entre une midlet et une applet. Ces échanges utilisent le protocole de communication CAD/carte à puce classique défini plus haut.

Dans la mise en œuvre et de façon pratique, la midlet se charge d'ouvrir le canal de communication avec la carte en spécifiant un certain nombre de paramètres (AID de l'applet cible, canal logique à utiliser). L'échange de données se fait ensuite de façon classique (schéma d'échanges APDU) en utilisant les API réservées à cet effet.

4.3 API à utiliser

Pour l'échange de commandes APDU avec la carte à puce en Java ME, on utilise la classe `javax.microedition.Apdu` qui fait partie des paquetages définis dans la JSR-177. Il est également important de noter que toute midlet utilisant les fonctionnalités de la JSR-177 en matière de communication APDU doit être signée par un certificat d'un organisme de certification « connu du téléphone » (Thawte ou Verisign par exemple).

Au niveau de l'applet, il n'y a pas d'API spécifique ; la méthode `process` reçoit les commandes APDU venant du CAD de manière classique, lance le traitement approprié et renvoie la réponse APDU.

4.4 Exemple de code commenté

Dans cet exemple, nous mettons en œuvre une petite application qui, au niveau de la midlet, provoque le lancement par l'applet d'une commande proactive, en l'occurrence l'affichage d'un message sur l'écran du téléphone mobile. Les codes des listings 3 et 4 mentionnés dans la suite sont disponibles à l'URL <http://www.labri.fr/perso/chaumett/papers/misc2008/usim/>.

Le listing 3 présente le code de la midlet correspondante qui utilise les API de la JSR-177 pour communiquer avec l'applet associée. La structure de la midlet reste classique avec les méthodes de base. Les nouveaux éléments au niveau de la classe principale `exemple_midlet_apdu` (ligne 11) sont le tableau d'octets `tempBuffer` (ligne 19) qui contient la commande APDU à exécuter, ainsi que la méthode `thread_apdu` (ligne 55) qui a pour rôle de lancer le thread (classe `Send_APDU`) en charge de gérer la communication avec l'applet.

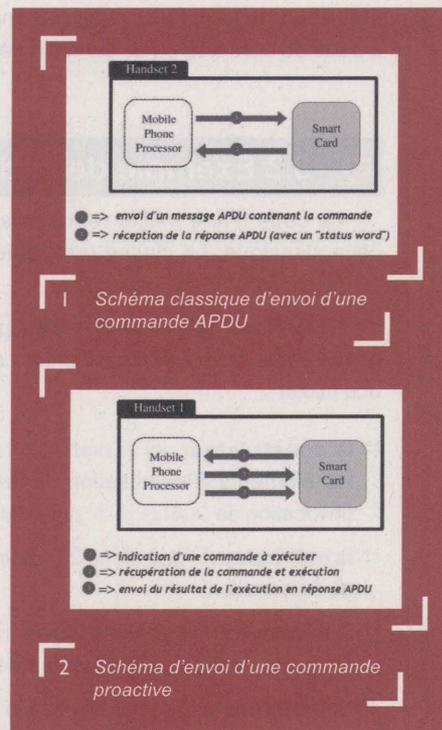
La classe `Send_APDU` (ligne 64) encapsule les appels à l'API nécessaires à la communication midlet/applet. L'objet principal en est `javax.microedition.apdu.APDUConnection` (ligne 5). Il est

tout d'abord utilisé dans la méthode `openAPDUConnection` (ligne 76) pour ouvrir la connexion apdu. Les paramètres de cette méthode sont le type de connexion (apdu dans notre cas) et la cible (dans notre exemple le `target` renvoie à l'AID de l'applet avec laquelle on veut communiquer). L'objet `APDUConnection` est utilisé ultérieurement dans la méthode `closeAPDUConnection` (ligne 88) chargée de fermer la connexion APDU lorsque tous les traitements sont terminés. Enfin, la méthode `sendAPDU` (ligne 98) reçoit en paramètre le tableau d'octets de la commande APDU à exécuter. Elle envoie la commande en invoquant `exchangeAPDU` (ligne 102) et retourne le résultat sous la forme d'un tableau d'octets. La méthode `run` (ligne 116) qui est exécutée au lancement du thread ouvre la connexion APDU, envoie la commande, récupère le résultat, puis ferme la connexion.

Le listing 4 présente le code de l'applet associée. Ce code reste classique. La méthode `process` (ligne 83), qui est exécutée à la réception d'un message APDU, lance la procédure qui conduira finalement à l'exécution de la commande proactive d'affichage d'un message à l'écran du téléphone mobile (message contenu dans le tableau d'octets `messageToDisplay`, ligne 19). L'invocation de `reg.setEvent` (ligne 89) dans la méthode `process` provoque à son tour l'invocation par le `runtime` de la méthode `processToolkit` en réaction à certains événements bien précis. Dans cet exemple, `processToolkit` (ligne 71) lance la méthode `displayText` (ligne 106) lors de la survenue de l'événement `EVENT_PROACTIVE_HANDLER_AVAILABLE` (lignes 72 et 74). Cet événement intervient lorsque le canal de communication entre l'applet et la midlet est libre pour envoyer une commande proactive.

`displayText` (ligne 106) récupère au niveau du système les informations concernant le `handler` (`ProactiveHandler`, ligne 107) de gestion des commandes proactives et invoque la méthode `displayMessage` (ligne 93) chargée de l'envoi du texte à afficher au téléphone mobile. `displayMessage` formate le message avec `initDisplayText` (ligne 97) et l'envoie avec `send` (ligne 99).

Pour tester le fonctionnement de ces codes, il est possible d'utiliser les outils de développement présentés dans la section 5.



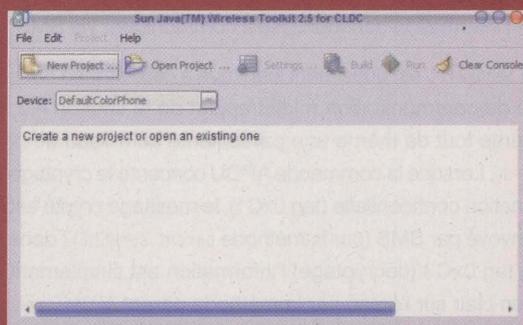
5. Outils de développement

5.1 Sun Wireless Toolkit

Le Sun WTK (*Sun Wireless Toolkit*) est un ensemble d'outils permettant de développer des applications Java compatibles avec les appareils (téléphones mobiles généralement) supportant la plateforme Java ME. Cette suite logicielle permet donc de compiler et de vérifier (au sens du vérificateur de bytecode) les midlets créés, mais il ne propose pas d'éditeur pour saisir le code des applications. Il est à noter qu'il est cependant possible de l'intégrer dans des IDE (*Integrated Development Environment*), tels que Eclipse et Netbeans. Le système de gestion de projets permet également de définir les spécificités des midlets développés en termes de version de CLDC et de MIDP, d'API optionnelles à utiliser, d'autorisations, etc.

On notera aussi que le WTK possède un environnement de simulation assez complet et qui permet de tester des éléments spécifiques comme l'émulation OTA (*over-the-air*) et l'émulation du *push registry* (pour le lancement automatique de la midlet lors de la survenue d'un événement spécifique, la réception d'un SMS par exemple). La signature des midlets est également supportée grâce à un système de gestion des certificats émis par des autorités de certification.

À la fin du processus de développement, le WTK produit le fichier JAR (Java ARchive) et le fichier JAD qui doivent être installés sur le téléphone mobile pour faire fonctionner la midlet.

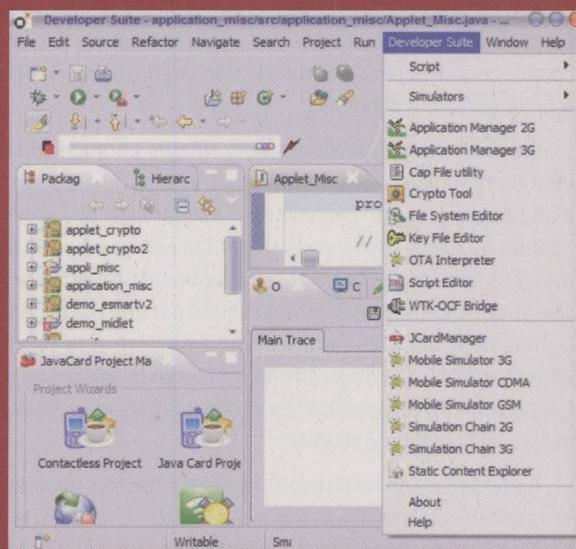


3 Interface graphique du WTK 2.5

5.2 Developer Suite de Gemalto

Developer Suite est une suite logicielle de développement d'applets Java Card (U)SAT de la société Gemalto. C'est un module qui s'intègre à l'IDE Eclipse. Nous l'utilisons dans sa version 3.1.

La Developer Suite est dotée d'un environnement de simulation assez bien fourni offrant de nombreuses fonctionnalités. Il est possible de simuler plusieurs types de cartes (U)SIM, et d'explorer leur contenu. Il est également possible de tester le fonctionnement des applets (U)SAT grâce au *Mobile Simulator*, ainsi que de tester l'envoi et la réception de SMS. Un point intéressant est l'intégration du WTK qui permet de faire des tests sur la communication midlet/applet avant de déployer les applications sur de véritables terminaux. Il est enfin important de noter la présence d'un *Application Manager* qui facilite le chargement des applets développées sur de véritables terminaux.



4 Interface graphique de la Developer Suite 3.1

6. Choix du matériel

6.1 Les téléphones

Les téléphones mobiles que l'on peut utiliser doivent implémenter la JSR-177 et supporter les applets signées par des CA (*Certificate Authority*) connues, comme

Thawte ou Verisign, les midlets utilisant la JSR-177 devant être signés.

Notre choix s'est porté sur le Nokia 5610 Xpressmusic qui remplit toutes les conditions et sur lequel nous avons effectivement réalisé des communications midlet/applet.

⇒ 6.2 Les cartes (U)SIM

Le challenge consiste à trouver des cartes (U)SIM connectées à un réseau GSM/3G (afin de pouvoir bénéficier des services réseau comme l'envoi de SMS) et sur lesquelles il est possible de charger et d'exécuter des applets Java Card et même des (U)SAT.

Les cartes (U)SIM R5/R6 de Gemalto (fournies avec le kit de développement de la Developer Suite) représentent une bonne solution, même si étant de test elles ne sont pas liées à un réseau GSM. Elles permettent néanmoins de tester l'envoi de commandes APDU d'une midlet vers une applet ainsi que certaines commandes proactives des (U)SIM.

⇒ 7. Exemple

Les sources de l'exemple étant un peu trop volumineuses, elles sont disponibles à l'URL suivante : <http://www.labri.fr/perso/chaumett/papers/misc2008/usim/>.

L'exemple que nous avons développé est un code permettant d'enregistrer des données confidentielles (coordonnées bancaires, mots de passe, etc.) dans un téléphone. Les informations sont stockées de façon sécurisée sur la carte (U)SIM et on peut les communiquer (par SMS par exemple) à une tierce personne toujours de façon sûre.

Cette application reprend tous les éléments que nous avons présentés dans cet article. Elle utilise également les possibilités cryptographiques des cartes Java pour chiffrer les informations confidentielles et ainsi pouvoir les transmettre sans les divulguer à un observateur éventuel.

Le scénario de l'application est simple. Lorsque l'utilisateur veut communiquer ses informations confidentielles, il se sert d'un bouton `send` qui déclenche la récupération de l'information chiffrée au niveau de la carte (U)SIM et son envoi par SMS. De manière symétrique, lors de la réception d'un SMS contenant une information sensible communiquée par une tierce personne, le message est automatiquement transféré à la carte pour déchiffrement avant d'être affiché sur l'écran du téléphone mobile. L'application est divisée en deux parties : la midlet sur le téléphone et l'applet sur la carte (U)SIM.

Le code de la midlet (listing 5, voir l'URL citée plus haut) est divisé en quatre blocs, à savoir, la classe principale `Midlet_Misc`, la classe `Send_APDU`, la classe `Receive_SMS` et la classe `Send_SMS`. Cette midlet est chargée de proposer une interface à l'utilisateur qui lui permette de déclencher l'envoi de l'information confidentielle à transmettre (chiffrée) par SMS. La midlet communique avec l'applet appropriée, récupère l'information chiffrée et l'expédie au contact indiqué. Les différentes classes qui ont été définies sont les suivantes :

⇒ La classe `Midlet_Misc` est la classe principale et elle présente une structure classique de midlet. Les différents éléments à noter se situent d'abord au niveau de son constructeur qui définit les éléments de l'interface graphique (`stringItem` pour l'affichage des informations et les boutons `commandExit`, `commandSend` et `commandReceive` pour l'exécution des différentes actions). La méthode `initApp` est exécutée au démarrage de l'application ; elle contient une séquence de lancement du thread chargé de recevoir les SMS (à travers la classe

`Receive_SMS`). La méthode `sendSMS`, elle, se charge tout simplement d'envoyer une chaîne de caractères par SMS en utilisant la classe `Send_SMS`. La méthode `fill_apdu` formate dans un tableau d'octets la commande APDU à envoyer à la carte (U)SIM pour réaliser une opération de déchiffrement. Cette APDU contient l'information à déchiffrer dans son champ de données. La méthode `receivedSMS` est chargée de déclencher les opérations à exécuter (formatage de la commande APDU et envoi de cette commande à la carte) lors de la réception par SMS d'un message contenant une information confidentielle. Enfin, la méthode `commandAction` déclenche les actions liées à chaque bouton de l'interface, `CMD_EXIT` qui ferme l'application, `CMD_SEND` qui récupère l'information dans la carte (en utilisant la classe `Send_APDU`) avant qu'elle ne soit envoyée par SMS et `CMD_RECEIVE` qui permet de lancer les opérations de déchiffrement au niveau de la carte (en utilisant là aussi la classe `Send_APDU`). La variable `smsPort` contient le numéro de port utilisé dans la classe `Receive_SMS` pour récupérer les messages SMS destinés à l'application. Le tableau d'octets `encryptCommand` quant à lui contient la commande APDU utilisée pour récupérer l'information confidentielle à communiquer via SMS.

⇒ La classe `Send_APDU` est quasiment identique à celle utilisée dans l'exemple de communication midlet/applet de la section 4.2. Elle présente tout de même une particularité au niveau de la méthode `run`. Lorsque la commande APDU concerne le cryptage de l'information confidentielle (tag 0xC1), le message crypté est ensuite envoyé par SMS (par la méthode `parent.sendSMS`) ; dans le cas du tag 0xC3 (décryptage) l'information est simplement affichée en clair sur l'écran par la méthode `parent.setMessage`.

⇒ La classe `Receive_SMS` gère la réception de tous les messages transférés par SMS sur le port `smsPort`. Dans le constructeur, on trouve l'ouverture de la connexion avec l'instruction (`MessageConnection`)`Connector.open(smsConnection)` associée à l'objet `smsconn` de type `MessageConnection`. La méthode `closeConnection` ferme la connexion SMS à la fin de tous les traitements. La méthode `run` quant à elle contient une boucle qui permet de recevoir, puis de traiter chaque SMS. Chaque message est reçu en utilisant `smsconn.receive` pour ensuite être traité par l'invocation de `parent.receivedSMS`.

⇒ La classe `Send_SMS` permet l'envoi par SMS d'une chaîne de caractères à une destination donnée. Toutes les actions sont

regroupées dans la méthode `run`. On ouvre tout d'abord la connexion comme dans la classe `Receive_SMS`. L'adresse de destination est ensuite définie avant que le message ne soit envoyé avec `smsconn.send`. La connexion est enfin refermée en invoquant `smsconn.close`.

Le code de l'applet (listing 6, voir l'URL citée plus haut) est lui aussi très classique. Il comprend les éléments présents dans toute applet Java Card, et intègre de plus les méthodes de chiffrement et de déchiffrement 3DES de tableaux d'octets. Les spécificités à noter se situent tout d'abord au niveau du constructeur `Applet_Misc` avec l'invocation de la méthode `initCrypto()`. Cette méthode permet d'initialiser tous les éléments nécessaires à l'utilisation d'une clé 3DES (dont la valeur est stockée dans le tableau d'octets `user3DESCipheringKeyValue`). La méthode `initCrypto()` est aussi chargée de l'initialisation des objets `cipher3DESEnc` et `cipher3DESDec` avec la clé 3DES. Ces objets contiennent respectivement les méthodes pour chiffrer et

déchiffrer les données. La méthode `padUserData` permet de formater correctement les données à chiffrer, la longueur de ces dernières devant être un multiple de 8 octets. Le reste du traitement au niveau de l'applet se retrouve dans la méthode `process`. Dans le cas de la réception d'une commande APDU avec le champ INS de type `INS_CRYPT (0xC1)`, l'information confidentielle est chiffrée avant d'être retournée dans la réponse APDU. L'information est auparavant formatée (longueur multiple de 8) avec `padUserData`, puis la méthode `cipher3DESEnc.doFinal(apduData, (short) 0x00, nbData, apduDataEncrypted, (short) 0x00)` effectue les opérations de chiffrement et le résultat est produit dans le tableau `apduDataEncrypted`. Si au contraire la commande APDU reçue dans la méthode `process` est du type `INS_DECRYPT (0xC3)`, les données contenues dans la commande sont alors déchiffrées avec `cipher3DESDec.doFinal(apduData, (short) 0x00, nbData, apduDataEncrypted, (short) 0x00)` pour ensuite être renvoyées en réponse APDU avec `apdu.sendBytesLong(apduDataEncrypted, (short)0x00,nbData)`.



Conclusion

L'utilisation des cartes à puce Java et de leurs possibilités cryptographiques dans le contexte d'applications embarquées pour téléphones mobiles ouvre de nombreuses possibilités. Il devient en effet possible de sécuriser les données liées aux applications afin de s'assurer que les personnes non autorisées n'y ont pas accès. Il est également possible de sécuriser les communications en les chiffrant. Ce chiffrement peut être mis en œuvre quelle que soit la technologie de communication utilisée : Bluetooth, WIFI, GSM, 3G.

De plus, la mise en œuvre des technologies utilisées est relativement simple et accessible, et les

équipements compatibles devraient se généraliser dans un avenir proche.

Dans le cadre de nos activités de recherche, nous utilisons ces technologies en particulier au sein du projet Multilevel Java Card Grid. Ce projet a pour objectif de proposer une architecture de communication multi-niveau et sécurisée utilisant des téléphones mobiles [5], [6]. Les utilisations potentielles de ce système sont nombreuses. Le paiement par téléphone mobile, le stockage et la communication d'informations sensibles sont autant d'exemples de domaine d'application. ■



Références

- ⇒ [1] *The Java Card technology*, <http://java.sun.com/javacard/index.jsp>
- ⇒ [2] *Java Card applet developer's guide*, <http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html?page=1>
- ⇒ [3] *USIM/USAT support*, http://www.accessdevnet.com/index.php/ACCESS-Linux-Platform-Native-Development/ALP_Telephony-MobileServices.html#1013158
- ⇒ [4] *The Java Micro Edition technology*, <http://java.sun.com/javame/technology/index.jsp>
- ⇒ [5] CHAUMETTE (S.), MARKANTONAKIS (K.), MAYES (K.) ET SAUVERON (D.), *The Mobile Java Card Grid*, e-Smart, 2006.
- ⇒ [6] CHAUMETTE (S.), OUOBA (J.), *The Multilevel Java Card Grid, invited poster*, WISTP 2007, Greece, 2007.
- ⇒ [7] SAUVERON (D.), *La technologie Java Card : présentation de la carte à puce*, La Java Card, RR-1259-01, LaBRI, Université Bordeaux 1, 2001.
- ⇒ [8] OUOBA (J.), *The Multilevel Java Card Grid, Master Thesis Report*, LaBRI, University Bordeaux 1, 2007.
- ⇒ [9] NEMEC (J.), *Simagine 2008 Training, Developing Java STK, NFC and SCWS applications*, Gemalto, Meudon, 2007.

LA PROGRAMMATION JAVA CARD

■ mots clés : API cryptographique / Java Card / Plate-forme programmable

La technologie Java Card fournit une machine virtuelle qui joue le rôle de système d'exploitation pour cartes à puce en apportant sécurité et indépendance vis à vis du matériel. Ainsi, des développeurs indépendants peuvent concevoir et déployer dans des cartes à puce toute une variété de produits et de services sécurisés, dans la poche de leurs utilisateurs. Nous explorons ici l'essence de cette technologie.

⇒ 1. Introduction

Les cartes à puce d'aujourd'hui sont des cartes à microprocesseur capables de fournir des ressources nécessaires pour l'exécution d'applications qui y sont embarquées. En particulier, ces systèmes sont capables de gérer les entrées/sorties entre les applications de la carte et leurs contreparties sur le terminal. Ils implémentent aussi, parfois, un système de fichiers normalisé ISO 7816-4 ou même une base de donnée ISO 7816-7.

Autrefois, la programmation des cartes était le fait des fabricants. Le problème de tels systèmes est que les fabricants de cartes à puce utilisent des logiciels propriétaires – tout en gardant la compatibilité avec l'ISO 7816, qui ne définit finalement qu'un jeu de commandes supportées par la carte. Programmer les applications demande alors une grande connaissance du matériel particulier de la carte cible, et on peut être sûr que la portabilité de telles applications est très faible. « Changer de fournisseur de carte » est alors synonyme de « Récrire l'application du début ». Quand une telle carte est distribuée

à l'utilisateur, il est assez difficile de mettre à jour l'application ou même d'installer de nouvelles applications sur la carte. Dans le cas de cartes multi-applications, cela veut dire que tous les développeurs des applications situées sur une carte doivent s'entendre pour créer la carte dans un souci de coopération.

L'idée derrière Java Card est de définir un standard de plate-forme d'application pour permettre le développement d'applications de manière indépendante de la carte, sans devoir passer par le fournisseur de la carte. Pour cela, on utilise une machine virtuelle au-dessus du système d'exploitation. Sur une machine hôte, on compile l'application, puis on la convertit (en la compressant au passage) en un code objet interprétable par la carte, puis on charge l'application sur la carte (en prenant soin de sécuriser la communication). La machine virtuelle qui se trouve du côté carte peut interpréter le code objet et exécuter l'application. De plus, on y trouve un langage portable et une API standardisée.

⇒ 2. Principes

⇒ 2.1 La base

Une carte à puce Java Card est une carte à microprocesseur qui est capable de faire tourner des programmes écrits dans le langage Java Card. À cet effet, il y a une machine virtuelle Java embarquée sur la carte.

Conceptuellement, une carte Java est un serveur : au lieu de lui envoyer des requêtes HTTP ou un message RCP, on lui envoie une commande APDU.

À la réception, la carte examine la Commande APDU, puis active (sélectionne) une application sur la carte (qu'on appelle une applet) ou la transmet juste à une applet précédemment sélectionnée.

À la suite de quoi, l'applet renvoie une réponse APDU (ou le système d'exploitation de la carte si l'applet ne s'en charge pas).

Le gros avantage de Java Card par rapport à une application native sur la carte est d'être – relativement – indépendant de la plateforme de développement, et de permettre de charger ou de mettre à jour dynamiquement les applications sur la carte, le tout dans un environnement standardisé.

Ces dernières opérations de gestion de l'application sur la carte ne sont pas adressées par le standard Java Card lui-même, mais par la spécification de Global Platform. Celle-ci définit une architecture pour gérer et installer des applications sur des cartes multi-applications, telles que les cartes Java Card de manière sécurisée.

Java Card offre un sous-ensemble très réduit du langage Java. Jusqu'à la version 2, il n'y avait pas de support pour les types `float`, `double` et `long`, et le support pour les `int` était limité. Les `threads` n'y sont pas non plus supportées et le code objet est différent du code objet Java normal.

C'est en grande partie justifié par les restrictions de place qu'on peut trouver sur une carte à puce.

Les spécifications de Java Card sont publiées ici [1]. Ces spécifications détaillent : l'API, le Java Card Runtime Environment (JCRC) et la machine virtuelle Java Card (JCVM). L'architecture Java Card est illustrée dans la figure 1.

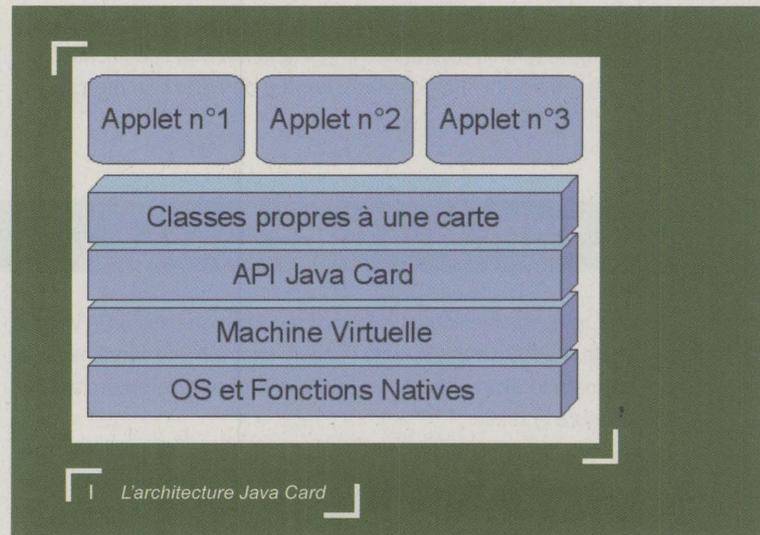
Les avantages de la technologie Java Card est d'utiliser un langage orienté objet – familier à de très nombreux développeurs – par opposition à un langage bas niveau, comme du C ou de l'assembleur, fortement dépendant de la plateforme cible : ici, on s'abstrait de la plateforme. On y trouve aussi des API standardisées.

La sécurité des applications n'est pas non plus négligée :

- ⇒ différents niveaux de contrôle d'accès aux champs et méthodes ;
- ⇒ le typage est fort ;
- ⇒ l'utilisation de pointeurs n'est pas autorisée (on va donc limiter les accès à des parties de la mémoire qu'on voudrait préserver) ;
- ⇒ un firewall sera utilisé pour séparer les différentes applets sur une même plateforme.

Avec Java Card et Global Platform, on va pouvoir gérer plusieurs applications sur la carte.

Un des objectifs de cet article est de montrer comment développer une application simple qui ne fait que renvoyer les `n` premiers octets reçus dans la commande APDU. On va compiler l'application, puis convertir les classes compilées en un fichier CAP (*Converted APplet*). Ce dernier contient du code objet converti, débarrassé de tout ce qui est superflu, et interprétable par une carte Java. Étant donnée une carte particulière, on va charger l'application sur la carte et faire tourner quelques exemples simples d'utilisation.



⇒ 2.2 Historique

Les premières démonstrations de Java Card datent d'octobre 1996. En novembre 1996, une version 1.0 de la spécification sous la forme d'un brouillon de 4 pages est publiée par Schlumberger.

En 1997, Schlumberger et Gemplus créent le Java Card Forum. Cette association regroupe les différents acteurs de la technologie Java Card avec le but de promouvoir la solution Java Card et de définir des choix technologiques, puis de les proposer à Sun qui en fait le « standard ».

En novembre 1997, sort une version 2.0 de la spécification qui est nettement plus conséquente en termes de taille et de précision. L'API est particulièrement différente de celle de la version 1.0.

Pour combler les manques de la spécification concernant la gestion dynamique d'applets, même après que la carte a été livrée à l'utilisateur, Visa introduit en avril 1998 son propre standard de gestion d'applets : Visa Open Platform (VOP), le but étant de définir des méthodes pour pouvoir charger des applets de manière sécurisée. Aujourd'hui, Global Platform est un héritier de cet effort, mais il est produit par un ensemble d'industriels plutôt que par Visa seul. Il est à noter que cette norme se concentre sur la gestion des applications pour des cartes multi-applicatives. Il n'est pas question explicitement de Java Card, et en fait on peut utiliser Global Platform pour d'autres technologies.

En mars 1999, Sun a publié la version 2.1 de la spécification Java Card. Celle-ci définit le format du fichier à charger sur la carte : le fichier CAP. Cette version rend standard l'environnement d'exécution et définit explicitement la machine virtuelle Java Card.

En 2003, la version 2.2 élargit la norme en permettant de faire du RMI (*Remote Method Invocation*) depuis l'application cliente. De plus, on y introduit la notion de « canaux logiques » qui permettent de communiquer en pseudo-simultané avec plusieurs applets.

La version 3.0 de la spécification est parue en avril 2008. Elle représente une avancée importante dans la spécification sur plusieurs points. Il n'existe pas encore de carte commercialisée

respectant cette norme, même si des cartes et des kits de développements ont été distribués et utilisés pour un concours lors de l'événement *Java One* 2008.

⇒ 3. L'API Java Card

On va survoler ici quelques entrées intéressantes de la version 2.1 de l'API. On a choisi cette version, car c'est une version qui est compatible avec une bonne partie des cartes commercialisées à l'heure actuelle.

⇒ 3.1 java.lang

`Object` est l'objet de base de la hiérarchie de classes Java Card. Il reflète le rôle de l'objet `Object` pour un Java plus traditionnel.

`Exception` est l'exception de base. En effet, on supporte les exceptions en Java Card, mais il faut noter que si on lève une exception qui n'est pas rattrapée du côté carte, on va donner une erreur à l'application cliente. C'est notamment le cas à travers les messages ISO-7816.

⇒ 3.2 javacard.framework

`ISO7816` est une interface qui contient tous les éléments Status Word des réponses APDU (par exemple, on pourra l'utiliser pour faire renvoyer un SW 0x6D00, avec `ISO7816.SW_INS_NOT_SUPPORTED`) et les positions initiales dans les commandes APDU des différents champs (comme CLA, INS et autres).

Toute applet est identifiée de manière unique par un numéro en hexadécimal appelé « AID » (*Application Identifier*) sur 16 octets (5 octets fixés par le fabricant de la carte et 11 octets fixés par le programmeur). Les applets appartiennent à des *packages* qui sont aussi identifiés à l'aide d'AID.

`Shareable` est l'interface à implémenter pour tout objet qui doit être partagé entre les applets. Un firewall contrôle les objets partagés et interdit les accès illicites entre applets chargées dans la carte.

Les commandes APDU contiennent 5 champs :

- ⇒ **CLA** : numéro hexadécimal (1 octet), désigne le type de catégorie d'applications (par exemple, CLA=BC pour les cartes de crédit françaises, CLA=A0 pour les cartes SIM, etc.) (champ obligatoire).
- ⇒ **INS** : code instruction (1 octet), contient le type de commande à lancer sur la carte (champ obligatoire).
- ⇒ **P1, P2** : (2 octets), désignent des paramètres éventuels du code instruction (champs obligatoires).

⇒ **Lc** : (1 octet) contient la taille du champ **DATA** (champ optionnel).

⇒ **DATA**: données si (champ optionnel).

⇒ **Le** (*Length Expected*) : (1 octet) nombre d'octets attendus en réponse (champ optionnel, si aucune réponse n'est attendue)

`APDU` est la classe qui gère l'essentiel des entrées/sorties. En effet, c'est par un objet `APDU` qu'on va traiter les données des commandes APDU en entrée. Pour suivre le protocole ISO 7816-4, on va d'abord récupérer l'en-tête par un appel à une méthode `getBuffer()`, ce qui nous donne un pointeur vers un tableau d'octets qui représente l'en-tête de l'APDU en tampon. Puis, on pourra accéder aux champs optionnels de l'APDU, s'il en existe, en faisant appel à la méthode `setIncomingAndReceive()`. Avec cet appel, le tableau d'octets obtenu avec `getBuffer()` inclut autant d'octets que possible sans faire déborder le tampon. On pourra éventuellement compléter avec des appels à la méthode `receiveBytes()`. Le même objet `APDU` est utilisé pour gérer à la fois les entrées et les sorties. Ainsi, on va utiliser la méthode `setOutgoing()` pour indiquer qu'on va utiliser l'objet `APDU` pour faire une sortie, ce qui nous retourne la taille des données attendues en sortie. On peut fixer notre taille de données à retourner en faisant appel à la méthode `setOutgoingLength()`. C'est le contenu du tampon qui va être renvoyé. On va donc vouloir modifier le contenu du tampon. On pourra ensuite renvoyer le contenu du tampon suivi automatiquement d'un *Status Word* (SW, par défaut 0x9000) avec la méthode `sendBytes()`.

`Applet` est la classe de base pour décrire le corps d'une application Java Card. Toute application Java Card doit posséder une classe qui étend cette classe. Parmi les méthodes qui devraient être implémentées par le développeur, on trouve `install(byte[] bArray, short bOffset, byte bLength)` qui permet d'effectuer l'installation de l'applet. On pourra par exemple y initialiser certains champs de l'instance. Cette méthode doit être implémentée dans toutes les applets. Elle est appelée lors de l'installation de l'applet. On y alloue une instance de la classe de notre applet. On doit ensuite enregistrer l'instance auprès du système. La réussite de cette opération rend l'applet sélectionnable via une Commande APDU `select`. En règle générale, la méthode `install` est celle qui va allouer la mémoire pour tous les objets, afin d'éviter les allocations intempestives dans le code. Les paramètres de la méthode peuvent être utilisés pour instancier les objets avec des paramètres particuliers. Cette méthode ne devrait être

utilisée qu'une seule fois au moment de l'installation de l'applet dans la carte. Ainsi, en phase de personnalisation de l'applet, on voudra mettre en place les différents éléments concernant l'utilisateur.

Une autre méthode qu'on va utiliser lors de l'installation est `register()`. Cette méthode permet d'enregistrer l'instance de l'objet `Applet` auprès de l'environnement d'exécution Java Card. Typiquement, on devra appeler cette méthode dans le corps d'`install`. La méthode `select()` servira à préciser ce qui arrive lors de la sélection de l'applet. Cette méthode doit retourner un `boolean`. Celui-ci doit être `true` si l'on doit considérer l'exécution de la méthode comme un succès. En sélectionnant l'AID de l'applet, on va en effet lancer cette méthode ainsi que la méthode `process()` qui va gérer les commandes APDU envoyées à la carte après cette étape de sélection. Cette dernière méthode sera donc le centre névralgique de l'application. On va s'en servir comme d'un centre de distribution des commandes. Lors de la sélection d'une autre AID, la méthode `deselect()` sera lancée. On pourra en profiter pour mettre à jour des compteurs par exemple.

`JCSystem` est une classe qui contient une collection de méthodes statiques concernant la gestion de transactions atomiques, la gestion de ressources, des mécanismes d'effacement d'objet et de partage d'objet. Concernant la gestion de transactions, un trio de méthodes est ainsi fourni : `beginTransaction()`, `commitTransaction()` et `abortTransaction()`. Une autre série de méthodes est utilisable pour réserver des tableaux en mémoire RAM pour une durée déterminée : `makeTransientObjectArray(short length, byte event)` par exemple pour un tableau d'objets. De telles méthodes existent pour les types `boolean`, `byte`, `short`. L'événement (`event`) nécessaire à ces méthodes est celui qui réinitialisera les tableaux. Cela peut être un `reset` ou un `deselect`. Les données en RAM sont perdues immédiatement lors d'une perte de tension. Il est à noter que les opérations sur les tableaux en RAM se font de manière non atomique.

L'interface `PIN` définit quelques méthodes qui doivent être implémentées pour tout objet du type code PIN. L'exemple classique de classe qui implémente cette interface est `OwnerPIN`. Ainsi, on pourra se servir de cette classe pour vérifier la validité d'un code PIN entré (`check()`), pour obtenir le nombre d'essais encore disponibles (`getTriesRemaining()`), pour savoir si le code PIN a été validé ou encore pour remettre à zéro le nombre de tentatives effectuées.

La classe `Util` contient des fonctions utiles généralement à toute programmation d'applet en Java Card. Toutes les méthodes ici sont statiques, et peuvent éventuellement être implémentées de manière native. La classe comporte ainsi des méthodes de comparaison de tableau (`arrayCompare()`), de copie de tableau (`arrayCopy()` et `arrayCopyNonAtomic()`, la dernière ne pouvant jamais utiliser de mécanisme de transaction), de remplissage de tableau (`arrayFillNonAtomic()`) et de manipulation de `short` (`getShort()`, `makeShort()`, `setShort()`).

`ISOException` est le type des exceptions utilisées pour communiquer un SW particulier à l'application cliente. Typiquement, ces exceptions seront levées à partir de la méthode `process()` d'une sous-classe d'`Applet`. On pourra utiliser des SW spécifiques à une application ou des SW propres à la norme ISO 7816-4 (en s'aidant par exemple de l'interface `ISO7816`).

⇒ 3.3 javacard.security

Ce package contient tout ce qui est disponible publiquement en termes de cryptographie sur les plateformes Java Card. Il n'y a donc pas d'implémentation des algorithmes de cryptographie ici, mais essentiellement des interfaces qui définissent la structure de l'architecture cryptographique.

`Key` est l'interface de base pour toutes les clés. On trouvera par exemple des interfaces comme `AESKey` ou `RSAPrivateKey` qui étendent cette interface.

`Checksum` est une classe abstraite qui sert de base aux CRC.

`KeyAgreement` est la classe abstraite à utiliser pour les différentes formes d'algorithmes d'échange de clés de Diffie-Hellman.

`KeyBuilder` est la classe qui permet de fabriquer les clés. La méthode de base est `buildKey()`, qui, étant donné un type de clé, sa taille, et un `boolean` pour le chiffrement de la clé, crée une clé correspondante.

`KeyPair` est une classe qui contient deux clés : une publique et une privée. La version 2.2.2 des spécifications Java Card prévoient des paires de clés pour les algorithmes DSA, RSA et courbes elliptiques.

`MessageDigest` est la classe abstraite utilisée pour les algorithmes de hachage.

`Signature` est la classe abstraite utilisée pour les signatures. Elle comporte des champs statiques désignant les différents types d'algorithmes de hachage qu'on peut rencontrer sur une carte, ainsi que des méthodes à implémenter concernant l'initialisation de l'objet `Signature` (`init()`), la signature de données (`sign()`) et la vérification d'une signature (`verify()`).

⇒ 3.4 javacardx.crypto

Ce package est une extension. Il est sujet à des contrôles concernant l'exportation. Il sert à implémenter l'architecture cryptographique des plateformes Java Card.

`Cipher` est la classe de base pour les algorithmes de chiffrement/déchiffrement. Elle définit un ensemble de constantes désignant les algorithmes. Une méthode statique `getInstance()` permet d'obtenir un objet `Cipher` correspondant à ce que l'on cherche. Puis, on peut faire des `init()`, `update()`, et enfin `finalize()` pour mettre en place les clés, envoyer des données et effectuer l'encodage des données.



4. Développement d'une application simple

4.1 Le développement proprement dit

On va développer une application qui renvoie la commande qui vient d'être envoyée.

L'idée est la suivante :

Si l'on charge, installe, et sélectionne l'applet à partir de son AID, puis qu'on envoie une commande APDU comme celle-ci :

```
CLA INS P1 P2 : 80 10 00 00
Lc : 02
DATA : 01 02
Le : 05
```

On devrait recevoir cette Réponse APDU :

```
DATA : 80 10 00 00 02
SW : 90 00
```

Plus généralement, les Commandes APDU doivent ressembler à ceci :

```
CLA INS P1 P2 : 80 10 XX XX
Lc : XX
DATA : XX XX ... XX
Le : XX
```

L'applet devra renvoyer les **Le** premiers octets de la commande suivis du Status Word **90 00** :

```
DATA : 80 10 XX XX XX XX ... XX
SW : 90 00
```

On aura besoin d'un compilateur Java standard pour compiler les `.java` en `.class`. On va développer avec une version relativement récente de Java 2SE 6. Par souci de compatibilité avec cette version du JDK, on va utiliser la dernière version du kit de développement de Sun, soit la version 2.2.2. En outre, on va utiliser une carte qui est compatible avec la version 2.1.1 du kit de développement. On va donc utiliser les deux versions du kit de développement : une pour compiler et une pour générer le fichier CAP [2].

4.2 Comment écrire une applet Java Card ?

Les API de base de Java Card permettent d'accéder à des fonctionnalités telles que la manipulation d'APDU, les transactions, le partage d'objet, ainsi que des fonctions utilitaires comme les opérations sur les tableaux.

L'utilisation de transactions est un aspect essentiel de la sécurité des applications pour carte à puce puisqu'il faut concevoir

des applications qui prennent en compte le retrait éventuel de la carte.

Les capacités cryptographiques des cartes dépendent de leur portée au niveau international, ainsi que de leur utilisation. Elles sont donc très variables d'une carte à l'autre.

L'idée, ici, est de développer pas à pas une application simple `MyApplet` (dans un fichier `MyApplet.java`) qui va renvoyer les `n` premiers octets de la commande APDU qu'on lui envoie.

Le langage Java a été adapté pour les cartes à puces en ne gardant que le strict nécessaire pour les applications sur la carte. Écrire une application simple est un peu plus compliqué que d'habitude avec Java.

La particularité des cartes est l'architecture client-serveur. Toutes les opérations sont déclenchées par des Commandes APDU. L'essentiel de ce qu'on va faire du côté carte, c'est le traitement des commandes APDU reçues et l'envoi de Réponse APDU. On reçoit un APDU avec éventuellement un champ de données, on analyse et exécute la commande, et on renvoie une réponse. L'essentiel est effectué à travers le package `javacard.framework`, et la classe principale est `Applet`.

```
01: package mypackage;
02: import javacard.framework.APDU; // Pour gérer les Entrées/Sorties
03: import javacard.framework.Applet; // La classe dont on hérite
04: import javacard.framework.ISO7816; // Pour envoyer des Status Words
    appropriés, et pour retrouver les différents champs des APDUs.
05: import javacard.framework.ISOException; // Pour lancer des exceptions
06: public class MyApplet extends Applet {
07:     // L'application doit étendre Applet et implémenter
08:     // la méthode process.
09:     public static final byte APP_CLA = (byte) 0x80; // La constante qui
    définit la classe attendue
10:     public static final byte APP_INS = (byte) 0x10; // La constante qui
    définit l'instruction attendue
11:     public MyApplet() {
12:         super();
13:     }
14:     public static void install(byte[] bArray, short bOffset, byte
    bLength) {
15:         // On crée une nouvelle instance de l'objet MyApplet
16:         MyApplet myapplet = new MyApplet();
17:         myapplet.register();
18:     }
19:     public boolean select() {
20:         return true;
21:     }
22:     public void process(APDU apdu) throws ISOException {
23:         byte buffer[] = apdu.getBuffer();
24:         // La variable locale buffer pointe maintenant sur la Commande APDU
    que l'on traite.
25:         if (selectingApplet())
26:             return;
27:         // selectingApplet() renvoie true quand on est en train de faire
    un select.
28:         // On vérifie qu'on a le bon octet CLA :
29:         if (buffer[ISO7816.OFFSET_CLA] == APP_CLA) {
30:             switch (buffer[ISO7816.OFFSET_INS]) {
31:                 case APP_INS:
```

```

32: // On recupere le champs DATA dans buffer :
33: short lc = apdu.setIncomingAndReceive();
34: // On prepare l'APDU de reponse :
35: short le = apdu.setOutgoing();
36: // L'objet apdu sert à la fois aux entrées et aux sorties
37: apdu.setOutgoingLength(le);
38: // On renvoie les le premiers octets de l'APDU (à partir de
l'octet 0).
39: apdu.sendBytes((short) 0, le);
40: break;
41: default:
42: // Si on n'a pas l'octet INS attendu, on renvoie une exception
spéciale avec un Status Word 0x6D00, ce qui a pour effet d'arrêter
l'exécution de l'application.
43: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
44: }
45: } else {
46: ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
47: }
48: }
49: }

```

Il est à noter que le package est l'unité de base pour une application Java Card. Toutes les classes d'une application sont normalement dans un seul et même package. C'est cette unité qu'on va par la suite charger sur la carte.

⇒ 4.3 Compilation

Il s'agit maintenant de compiler notre programme Java Card.

Tout développeur de Java trouvera relativement naturelle l'étape suivante, à ceci près qu'on utilise `-target 5` pour garder un fichier `.class` qui soit d'une version compatible avec le jar du kit de développement.

```

$ javac -target 5 \
-classpath /chemin/vers/JCDK/java_card_kit-2_2_2/lib/api.jar \
mypackage/MyApplet.java

```

⇒ 4.4 Conversion

Les fichiers `.class` contiennent le code objet qui peut être interprété par une machine virtuelle Java.

En Java Card, on se doit d'aller un peu plus loin, puisque les fichiers `.class` contiennent beaucoup d'informations qui sont inutiles et qui encombreraient dans un environnement restreint par la taille pour une exécution dans une carte à puce.

Il faut donc convertir et regrouper/archiver les fichiers `.class` en un fichier CAP (Converted APplet). Le fichier CAP est un jar qui contient l'ensemble des modules à charger sur la carte.

Les outils fournis par SUN dans le Java Card Development Kit [2] vont permettre d'effectuer cette étape.

On va commencer par initialiser quelques variables :

```

$ export JAVA_HOME=/usr/java/jdk1.6.0_03/
$ export JC_HOME=/chemin/vers/JCDK/java_card_kit-2_2_2/

```

On définit une AID : `0x4D:0x79:0x41:0x70:0x70:0x6C:0x65:0x74:0x01` (c'est-à-dire MyApplet en ASCII).

La carte pour laquelle on développe est compatible Java Card 2.1.1. On va donc utiliser les fichiers d'export de ce kit de développement pour lier notre application aux bibliothèques de base (c'est la variable `exportpath` du fichier de configuration).

On va entrer tous ces paramètres dans un fichier `config.txt`, qu'on va par la suite donner en argument à l'outil de conversion inclus avec le kit de développement.

```

# Le fichier config.txt
-out EXP JCA CAP
-exportpath /chemin/vers/JCDK/jc211/api21/
-applet 0x4D:0x79:0x41:0x70:0x70:0x6C:0x65:0x74:0x01
MyApplet
mypackage
0x4D:0x79:0x41:0x70:0x70:0x6C:0x65:0x74
1.0

```

Le fichier `config.txt` contient dans l'ordre :

- ⇒ Les types des fichiers qu'on veut obtenir :
 - ↳ `.exp` : c'est l'*export file* nécessaire si on veut développer une autre application utilisant le `.CAP` qu'on va générer.
 - ↳ `.jca` (*Java Card Assembly*) : c'est un fichier lisible par un humain contenant le code objet du package.
 - ↳ `.cap` : l'applet à charger.
- ⇒ Le chemin d'accès des export files dont on a besoin. Ici, on a besoin des export files de l'API Java Card 2.1.1.
- ⇒ L'AID de l'applet et le nom de la classe lui correspondant.
- ⇒ Le nom du package à charger.
- ⇒ L'AID du package.
- ⇒ La version majeure.mineure du package.

Pour lancer la conversion, on a juste à lancer :

```

$ ./chemin/vers/JCDK/java_card_kit-2_2_2/bin/converter \
-config config.txt

```

Cette dernière commande devrait générer les trois fichiers demandés par le fichier `config.txt` et, en particulier, le fichier CAP qui nous intéresse ici.

⇒ 4.5 Chargement de l'applet sur la carte

Une fois le fichier CAP généré, il nous faut le charger et l'installer sur la carte.

Il va donc falloir utiliser un outil qui fasse appel au standard Global Platform [3] pour effectuer le chargement. Global Platform définit des mécanismes de chargement et de gestion de l'applet. La procédure à suivre pour charger une applet est, en général, la suivante :

- ⇒ On va dialoguer avec le Domaine de Sécurité de la carte. Il faudra donc au préalable sélectionner ce dernier.
- ⇒ Il va falloir ouvrir un canal sécurisé pour lui envoyer le package. Cette partie de la procédure est effectuée en montrant à la carte qu'on partage un secret avec elle : en général, ce sont les clés MAC (signature) et ENC (encryption).
- ⇒ On va éventuellement procéder à l'effacement des applets et des packages qu'on a pu installer précédemment et qui ont la même AID que notre applet et notre package.
- ⇒ On va procéder au chargement de notre package et à l'installation de notre applet.

Dans de nombreux environnements de développement pour Java Card, un outil est intégré pour permettre de charger et d'installer les applets.

On se propose ici d'utiliser un outil libre : GPSShell [4]. Il est disponible en LGPL pour de nombreux systèmes d'exploitation.

Pour utiliser l'outil GPSShell, il nous faut installer la bibliothèque Global Platform, puis l'outil GPSShell lui-même. Ce dernier utilise la bibliothèque – à travers PC/SC pour communiquer avec la carte.

La carte que nous utilisons ici est une carte compatible avec la version 2.1.1 de la norme Java Card, et avec la version 2.0.1 de Global Platform, et l'AID de son domaine de sécurité est **A000000003000000**.

On peut déjà faire un premier test de la carte avec une série de commandes qui vont lister les exécutables sur la carte. La procédure est relativement similaire à celle de l'installation à ceci près qu'une fois le canal sécurisé ouvert on va juste lancer une commande permettant d'afficher la liste. On pourra utiliser cette commande par exemple pour vérifier qu'on a bien installé notre applet sur la carte. Il n'y a qu'un nombre limité d'authentifications possibles.

```
# Le fichier list.txt
enable_trace
establish_context
card_connect
select -AID a000000003000000
open_sc -security 1 -keyind 0 -keyver 0 \
-mac_key 404142434445464748494a4b4c4d4e4f \
-enc_key 404142434445464748494a4b4c4d4e4f
// Ouverture de canal sécurisé.
// Toutes les clés ici sont standard
// l'option "-security 1" indique qu'on va
// signer avec la clé MAC tous les échanges suivants
get_status -element e0
// On obtient des informations sur le domaine de sécurité,
// les applications et exécutables
// Voir les normes de Global Platform pour plus d'information
card_disconnect
release_context
```

Cette suite de commandes peut ensuite être lancée avec l'outil GPSShell de la manière suivante :

```
$ gpshell list.txt
```

On va de la même manière écrire un script GPSShell pour installer l'application :

```
# Le fichier install.txt
enable_trace
establish_context
card_connect
select -AID a000000003000000
open_sc -security 0 -keyind 0 -keyver 0 -mac_key \
404142434445464748494a4b4c4d4e4f -enc_key \
404142434445464748494a4b4c4d4e4f
// On supprime les applications ainsi que les packages
// précédemment installées ayant les mêmes AIDs
delete -AID 40794170706c657401
delete -AID 40794170706c6574
install -file mypackage.cap -nvDataLimit 2000 \
-instParam 00 -priv 4 -sdAID a000000003000000
card_disconnect
release_context
```

Voici un autre script pour effacer l'application :

```
# Le fichier delete.txt
enable_trace
establish_context
card_connect
select -AID a000000003000000
open_sc -security 0 -keyind 0 -keyver 0 \
-mac_key 404142434445464748494a4b4c4d4e4f \
-enc_key 404142434445464748494a4b4c4d4e4f
delete -AID 40794170706c657401
delete -AID 40794170706c6574
card_disconnect
release_context
```

Voici enfin un script pour tester l'application :

```
# Le fichier send.txt
enable_trace
establish_context
card_connect
select -AID 4d794170706c657401
// On sélectionne l'AID de notre applet
send_apdu -APDU 8010000100102030405060708090a0b0c0d0e0f1016
// On envoie un APDU de test à notre application
card_disconnect
release_context
```

Pour charger notre application sur la carte, il suffit donc de lancer :

```
$ gpshell install.txt
```

On peut ensuite tester notre application :

```

$ gpshell send.txt
enable_trace
establish_context
card_connect
select -AID 4d794170706c657401
Command --> 00A40400094D794170706C657401
Wrapped command --> 00A40400094D794170706C657401
Response <-- 9000
send_apdu -APDU
80100000100102030405060708090A0B0C0D0E0F1016
Command -->
80100000100102030405060708090A0B0C0D0E0F1016
Wrapped command -->
80100000100102030405060708090A0B0C0D0E0F1016
Response <--
80100000100102030405060708090A0B0C0D0E0F10009000
card_disconnect
release_context

```

Ici, on envoie un APDU qui se découpe comme ceci :

⇒ **CLA INS P1 P2 : 80 10 00 00 – 80** correspond à la classe attendue par l'applet.

⇒ **Lc : 10** – soit 16 en hexadécimal.

⇒ **DATA : 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10** – nos 16 octets.

Le : 16 – soit 22 en hexadécimal : on demande donc en retour autant d'octets que ce que l'on envoie.

L'APDU de réponse se décompose comme ceci :

```

DATA : 80 10 00 00 10 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 00
SW : 90 00

```

Il est à noter que l'APDU reçu contient les parties d'en-tête, **Lc**, et **DATA** de la commande APDU. Il ne contient pas l'octet **Le** (16), mais un octet de remplissage **00**.

On pourra éventuellement procéder à un effacement de l'application :

```
$ gpshell delete.txt
```

Conclusion

Java Card est en lui-même un sujet assez vaste, puisque le langage a fortement évolué au cours du temps. L'architecture de Java Card a permis, dans le monde de la carte à puce, de se démarquer du matériel et de libérer le développeur d'application de la contrainte de programmer en assembleur des systèmes sensibles d'un point de vue sécurité. Cela permet également d'ouvrir les cartes à puces en offrant une plate-forme multi-applicative, dans laquelle le chargement des applications se fait de manière dynamique.

D'autre part, la dernière version de Java Card se rapproche plus d'un Java standard en permettant des mécanismes

de *multi-threading* et de communication via un protocole HTTP, en utilisant des objets communs aux programmeurs Java (*String*, *Integer*...), et en utilisant du code objet Java traditionnel (sans conversion) pour charger l'application sur la carte. Avec des plateformes Java Card de grandes capacités mémoire, le développement sur carte à puce en Java donnera l'impression d'utiliser de petits serveurs web, tout en gardant la possibilité de programmer les cartes dans un Java Card plus classique tel qu'il est décrit ici. Cela devrait permettre l'ouverture des cartes vers des programmeurs de Java standard et une liberté de programmation accrue. ■

Références

- ⇒ [1] SUN : Spécifications de Java Card : <http://java.sun.com/javacard>
- ⇒ [2] SUN : *Java Card Development Kit* de Java Card : <http://java.sun.com/javacard/devkit/>
- ⇒ [3] Spécifications Global Platform : <http://www.globalplatform.org/>
- ⇒ [4] GPSHELL : <http://sourceforge.net/projects/globalplatform/>
- ⇒ [5] Un blog avec des didacticiels : <http://javacard.veilles.com/?cat=16>
- ⇒ [6] <http://damien.sauveron.fr/>
- ⇒ [7] HUSEMANN (D.), « *Standards in the smart card world* », *Computer Networks*, volume 36, numéro 4, 16 juillet 2001, pp. 473-487(15).
- ⇒ [8] JSR 268 Smart Card I/O API : <http://jcp.org/en/jsr/detail?id=268>

UN SDK JAVACARD GÉNÉRIQUE OU COMMENT DÉVELOPPER UNE APPLICATION CARTE COMPLÈTE POUR TOUTES LES CARTES JAVA

■ mots clés : *carte à puce / développement logiciel / Java / JavaCard / OpenCard Framework / simulateur*

Les informations que l'on peut trouver ici et là quant au développement d'applications complètes mettant en œuvre des cartes à puce JavaCard programmables sont bien souvent parcellaires. Nous exposons dans cet article les différentes étapes nécessaires à l'assemblage complet d'un kit de développement logiciel JavaCard générique, à savoir n'étant lié à aucun modèle de carte Java en particulier, à partir de logiciels librement téléchargeables (à différencier avec des logiciels libres téléchargeables). Ce SDK permet de tester l'application de manière transparente sur carte ou simulateur. Nous le mettons en pratique sur un exemple d'application de type HelloWorld.

➔ 1. La carte Java

Roland Moréno, un passionné d'électronique, donne naissance à la carte à puce en 1974. À ce moment, il s'agit d'une carte munie d'une simple mémoire du commerce. Il faudra attendre 1978 et Michel Ugon avec son brevet SPOM (*Self Programmable One-chip Microcomputer*) pour voir les premières cartes à puce à microprocesseur permettant d'exécuter des programmes dans un environnement sécurisé encore à ce jour sans égal, qui se révéleront pleinement dans les usages cryptographiques. Ce sont ces mêmes cartes qui peuplent aujourd'hui notre quotidien et que nous nommons par abus de langage « cartes à puce » ou bien encore plus succinctement « carte ».

La programmation des cartes à puce est longtemps restée l'apanage de ceux ayant directement accès à cette technologie, à savoir les fabricants de cartes à puce. Les cartes étaient alors directement programmées en assembleur dédié, la roue devant être inlassablement réinventée à chaque changement de matériel. Bien que certaines cartes à puce aient été spécifiquement

développées pour en faciliter la programmation (certaines même en BASIC [1]), aucun standard ne réussit à s'imposer.

Fin 1996, la division carte à puce de Schlumberger, pas encore devenue feu-Axalto, présente un prototype de carte à puce révolutionnaire. En effet, cette carte est capable d'exécuter des programmes basés sur un nouveau langage censé s'abstraire de toute contrainte architecturale matérielle, Java [2], présenté en 1995 par Sun Microsystems (et rendu libre depuis 2006 [3]). La carte Java était née. Dans la foulée, Sun Microsystems normalise ce langage de programmation basé sur Java et destiné aux cartes à puce. Ce sera donc JavaCard [4]. La version initiale 1.0 sera supplantée dès 1997 par JavaCard 2.0 qui donnera lieu aux premiers produits commercialisés utilisables. En 1999, sort une version majeure à l'API cryptographique fournie, JavaCard 2.1, qui est aujourd'hui encore la version la plus utilisée (le monde de la carte à puce a en effet une forte inertie, essentiellement à cause des processus de certification). En 2002, sort la version

JavaCard 2.2, qui permet notamment d'utiliser plus simplement la carte à partir de programmes écrits en Java tournant sur l'ordinateur hôte, feu-Gemplus ayant adapté la technologie Java RMI aux cartes Java. Il aura fallu attendre 2008 pour que les spécifications de JavaCard 3.0 soient publiées [5], annonciateurs de changements majeurs dans le langage, sans

qu'il n'y ait encore de produits exploitant cette nouvelle mouture du langage JavaCard.

Tout cela pour dire que c'est donc de cartes à puce à microprocesseur JavaCard qu'il s'agit, dès lors que l'on parle abusivement de « cartes Java ».

⇒ 2. Justification d'un tel SDK

Si vous avez une carte Java entre les mains, il est fort à parier que vous devez vous ranger dans une de ces deux catégories : l'utilisateur ou le programmeur. En tant qu'utilisateur, vous n'avez même pas à savoir que votre carte bancaire ou Télécom est une carte Java. Il suffit qu'elle fonctionne comme escompté lors de l'usage que vous en faites. Si, par contre, votre but est de la programmer, il en va tout autrement et vous devrez faire avec le kit de développement logiciel fourni par le fournisseur de votre carte Java. Les raisons ne manquent pas pour avoir envie de faire autrement : à but éducatif, à cause du poids prohibitif d'un SDK propriétaire, pour des raisons de portabilité vers différentes cartes Java, pour pouvoir scripter les actions au lieu de les cliquer, pour avoir un SDK sur un espace de stockage amovible et être en mesure de l'utiliser sur différents systèmes, que sais-je encore.

Lors de son utilisation au sein de programmes, une carte à puce doit être considérée comme un serveur que l'on interroge. La partie du logiciel chargée d'interroger la carte est donc un client vis-à-vis d'elle. Les normes ISO 7816 [6] définissent, entre autres choses régissant la carte à puce, le format des informations que s'échangent ces deux entités : les APDU (*Application Protocol Data Unit*). Une application carte à puce est donc toujours composée d'une partie cliente s'exécutant sur l'ordinateur hôte et d'une partie serveur s'exécutant sur la carte, qui s'échangent des APDU. Dans le cas qui nous intéresse, la partie serveur est écrite en JavaCard (elle aurait pu l'être en assembleur, BASIC ou C). Pour la partie cliente, nous aurions pu choisir du C, C++, Python, Perl ou autre. Notre choix s'est arrêté sur Java pour des

raisons de portabilité du SDK. Bien que la dernière version en date, Java 1.6, supporte l'échange direct d'APDU avec une carte à puce, nous avons décidé d'utiliser l'API Java externe OCF, ou OpenCard Framework [7]. OCF permet aux programmes Java de s'interfacer avec des cartes depuis Java 1.1, et sa portabilité et sa robustesse sont bien meilleures que celles du package récent de Java. OCF va surtout nous permettre de pouvoir tester notre application carte de manière transparente et indifférenciée avec un simulateur de carte ou bien une vraie carte Java.

Nous avons choisi ici d'illustrer la mise en place du SDK en environnement Windows pour des raisons de commodité. Néanmoins, les choix techniques réalisés font qu'il est fonctionnel sous UNIX en portant les scripts et en téléchargeant la version ad hoc des JavaCard Development kits officiels. Il ne faudra juste pas oublier d'installer au préalable la couche système PCSC-Lite du projet MUSCLE [8] afin d'être capable de gérer les lecteurs de cartes à puce à la manière de PC/SC [9] (le middleware intégré aux systèmes de Microsoft depuis Windows 2000, devenu par conséquent standard de fait). Pour ce faire, on peut consulter des aides en ligne [10] si besoin.

Petite précision, les scripts de chargement/installation et d'effacement d'*applets* JavaCard dans une vraie carte Java sont configurés par défaut pour fonctionner avec la carte Java référence CyberFlex eGate 32 ko, disponible sur Internet. Pour utiliser d'autres cartes Java, se référer à la documentation du *loader* d'applet utilisé (qui peut être propriétaire ou non, pourvu que l'on charge le fichier *.cap* issu de la compilation du SDK).

⇒ 3. Construction du SDK

⇒ 3.1 Récupération des différents morceaux logiciel

Différentes pièces de logiciel étant nécessaires à la réalisation de ce SDK, il faut commencer par les télécharger. L'utilisation de OCF requiert à elle seule de récupérer quatre fichiers:

- ⇒ [BaseOCF.zip](#) [11], 1467 Ko ;
- ⇒ [Reference_Impl.zip](#) [12], 2369 Ko ;
- ⇒ [apduio-terminal-0.1.zip](#) [13], 381 Ko ;

⇒ [pcsc-wrapper-2.0.zip](#) [14], 573 Ko.

On va ensuite récupérer à partir de [15] les dernières versions des JavaCard Development kits officiels 2.1 et 2.2 :

- ⇒ [java_card_kit-2_1_2-win.zip](#), 1230 Ko ;
- ⇒ [java_card_kit-2_2_2-windows.zip](#), 29463 Ko.

Téléchargeons à partir de [16] la dernière version du JDK 1.6 Update 7 :

⇒ [jdk-6u7-windows-1586-p.exe](#), 79306 Ko.

Enfin, il faut des logiciels pour pouvoir administrer la vraie carte Java :

- ⇒ `captransf-1.5.zip` [17], 113 Ko *uniquement pour les possesseurs de cartes Java CyberFlex* ;
- ⇒ `GPSHell-1.4.2.zip` [18], 678 Ko.

Une fois ces neuf fichiers téléchargés, nous pouvons passer à leur assemblage.

⇒ 3.2 Assemblage des différents morceaux logiciel

⇒ 3.2.1 L'arborescence

Le SDK est construit dans un répertoire dédié que l'on nomme `JavaCardSDK`.

Dans ce répertoire `JavaCardSDK`, on crée trois sous-répertoires, comme suit :

- ⇒ `JavaCardSDK\misc` (pour les programmes externes téléchargés) ;
- ⇒ `JavaCardSDK\out` (pour le résultat des compilations) ;
- ⇒ `JavaCardSDK\src` (pour les sources des programmes).

On décompresse dans le sous-répertoire `JavaCardSDK\misc` les archives précédemment téléchargées, de manière à avoir les sous-répertoires suivants :

- ⇒ `JavaCardSDK\misc\apduio-terminal-0.1` ;
- ⇒ `JavaCardSDK\misc\captransf-1.5` ;
- ⇒ `JavaCardSDK\misc\gpshell-1.4.2` ;
- ⇒ `JavaCardSDK\misc\jCK2.1.2` ;
- ⇒ `JavaCardSDK\misc\jCK2.2.2` — obtenu en décompressant `java_card_kit-2_2_2-rr-bin-windows-do.zip` de l'archive 2.2.2 téléchargée ;
- ⇒ `JavaCardSDK\misc\JDK1.6` ;
- ⇒ `JavaCardSDK\misc\OCF1.2` ;
- ⇒ `JavaCardSDK\misc\pcsc-wrapper-2.0`.

⇒ 3.2.2 Les fichiers

Il faut créer trois fichiers textes nécessaires au simulateur de carte Java et les placer dans `JavaCardSDK\misc` :

`Header.scr` contient les premières instructions pour initialiser le simulateur :

```
powerup;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
```

`Install.scr` contient l'APDU d'initialisation de l'applet, une fois celle-ci chargée dans le simulateur :

```
// create applet
0x80 0xB8 0x00 0x00 0x0A 0xA0 0x00 0x00 0x62 0x3 0x1 0xC 0x6 0x1 0x7F;
```

`Footer.scr` contient la dernière instruction pour initialiser le simulateur (en l'occurrence pour l'arrêter) :

```
powerdown;
```

Dans `JavaCardSDK`, on édite le fichier de configuration `OCF1.2\lib\opencard.properties` du client, d'une longueur de deux lignes, différemment en fonction de la présence ou non d'un lecteur de carte à puce sur l'ordinateur (seul le simulateur sera actif dans ce second cas).

Version de `opencard.properties` avec lecteur de carte à puce et simulateur :

```
OpenCard.services = opencard.opt.util.PassThruCardServiceFactory
OpenCard.terminals = com.ibm.opencard.terminal.pcsc10.Pcsc10CardTerminalFactory
com.gEMPLUS.opencard.terminal.apduio.ApduIOCardTerminalFactory|mySim|Socket|localhost:9025
```

Version de `opencard.properties` sans lecteur de carte à puce, mais avec simulateur :

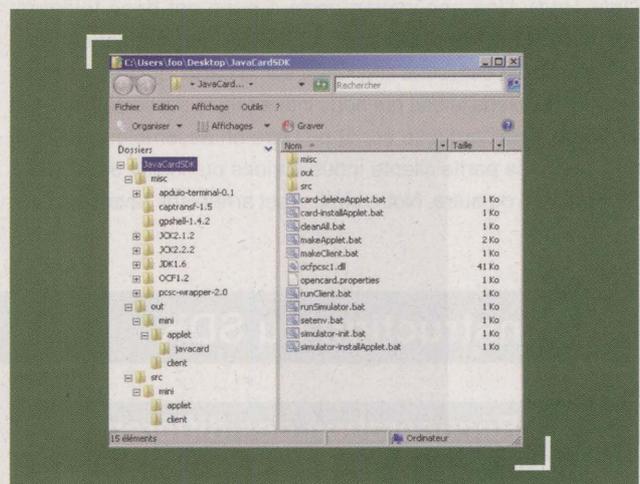
```
OpenCard.services = opencard.opt.util.PassThruCardServiceFactory
OpenCard.terminals = com.gEMPLUS.opencard.terminal.apduio.ApduIOCardTerminalFactory|mySim|Socket|localhost:9025
```

On notera la possibilité d'indiquer au client l'emplacement de la machine dans le réseau sur laquelle le simulateur est à l'écoute. Par défaut, il est sur la machine courante `localhost` au port TCP 9025 (pensez à configurer vos firewalls en conséquence, le cas échéant).

Enfin, on copiera le fichier `OCFPCSC1.d11` du sous-répertoire `JavaCardSDK\misc\OCF1.2\lib` vers `JavaCardSDK`.

⇒ 3.2.3 Les scripts

Tous se trouvent à la racine de `JavaCardSDK`. Ce sont eux qui vont faire fonctionner le SDK. La figure qui suit illustre l'arborescence du SDK après une compilation réussie (on remarque que le répertoire `JavaCardSDK\out` n'est pas vide), avec un focus sur le répertoire principal et ses scripts.



⇒ 3.2.3.1 setenv.bat

Ce script contient toutes les données de configuration d'un projet donné, à savoir l'emplacement des applications nécessaires, le nom des sous-répertoires utilisés, les numéros AID des applets JavaCard ou encore des données techniques quant au type de la carte Java utilisée. Il est appelé par tous les

autres scripts pour les configurer automatiquement. On notera la variable `JC_EXP` permettant de compiler une applet JavaCard à la norme JC2.1 ou JC2.2 en fonction du type de la carte Java utilisée (JC2.1 par défaut), ainsi que les numéros AID des applets et *packages* JavaCard écrits deux fois en fonction des outils utilisés. Si plusieurs projets sont développés sur un même type de carte Java, il suffit de ne changer que la variable `PROJECT` qui définit le répertoire du projet dans `JavaCardSDK\src\`.

```
set PROJECT=mini
set DIR=.
set OUT=%DIR%\out
set SRC=%DIR%\src
set MISC=%DIR%\misc
set CAPTRANSF=%MISC%\captransf-1.5
set GPSHELL=%MISC%\gpshell-1.4.2
set JC21_HOME=%MISC%\jck2.1.2
set JC22_HOME=%MISC%\jck2.2.2
set JAVA_HOME=%MISC%\jdk1.6
set OCF_HOME=%MISC%\OCF1.2
set PCSC_WRAPPER=pcsc-wrapper-2.0
set APDUIO_TERM=apduio-terminal-0.1
set JC21_API=%JC21_HOME%\lib\api21.jar
set JC22_API=%JC22_HOME%\lib\api.jar
set JC21_EXP=%JC21_HOME%\api21_export_files
set JC22_EXP=%JC22_HOME%\api_export_files
set JC_EXP=%JC21_EXP%
set PKGCLIENT=client
set CLIENT=TheClient
set PKGAPPLET=applet
set APPLET=TheApplet
set AIDPACKAGE=0xA0:0x00:0x00:0x00:0x62:0x03:0x01:0x0C:0x06
set PACKAGEAID=A00000006203010C06
set AIDAPPLET=0xA0:0x00:0x00:0x00:0x62:0x03:0x01:0x0C:0x06:0x01
set APPLETAID=A00000006203010C0601
set CARDSECURITYDOMAIN=A000000030000000
set CARDSECURITYDOMAIN2=A000000030000000
set CARDKEY=404142434445464748494A4B4C4D4E4F
set PKGVERSION=1.0
set SIMUSCRIPT=script
```

⇒ 3.2.3.2 cleanAll.bat

Comme son nom l'indique, ce script permet d'effacer les fichiers issus d'une compilation passée.

```
@echo off
call .\setenv.bat
REM effacement fichiers temporaires
del /Q %DIR%\*-
del /Q %SRC%\PROJECT%\PKGAPPLET%\*-
del /Q %SRC%\PROJECT%\PKGCLIENT%\*-
REM effacement client
rmdir /S /Q %OUT%\PROJECT%\PKGCLIENT%
REM effacement applet
rmdir /S /Q %OUT%\PROJECT%\PKGAPPLET%
REM effacement données simulateur
del %OUT%\PROJECT%\APPLET%.scr %OUT%\PROJECT%\script.scr %OUT%\PROJECT%\eeprom
```

⇒ 3.2.3.3 makeApplet.bat

Ce fichier compile l'applet JavaCard et met en place les fichiers nécessaires à son utilisation par le simulateur. En cas d'utilisation avec une vraie carte Java, il suffit juste de charger dans la carte le fichier `.CAP` compilé.

```
@echo off
call .\setenv.bat
echo Compiling...
%JAVA_HOME%\bin\javac.exe -target 5 -g -classpath %JC22_API% -d %OUT%\PROJECT%\SRC%\PROJECT%\PKGAPPLET%\APPLET%.java
if errorlevel 1 goto error
echo %APPLET%.class compiled: OK
echo .
echo Converting...
%JAVA_HOME%\bin\java.exe -classpath %JC22_HOME%\lib\converter.jar;%JC22_HOME%\lib\offcardverifier.jar com.sun.javacard.converter.Converter -nobanner -classpath %OUT%\PROJECT%\ -exportpath %JC_EXP% -d %OUT%\PROJECT%\ -out EXP_JCA_CAP -applet %AIDAPPLET% %PKGAPPLET%.APPLET% %PKGAPPLET% %AIDPACKAGE% %PKGVERSION%
if errorlevel 1 goto error
echo %APPLET%.cap converted: OK
echo .
copy %DIR%\%OUT%\PROJECT%\PKGAPPLET%\javacard\PKGAPPLET%.cap %DIR%\%OUT%\PROJECT%\
echo Scripting...
%JAVA_HOME%\bin\java.exe -classpath %JC22_HOME%\lib\scriptgen.jar com.sun.javacard.scriptgen.Main -nobanner -o %OUT%\PROJECT%\APPLET%.scr %OUT%\PROJECT%\PKGAPPLET%\javacard\PKGAPPLET%.cap
if errorlevel 1 goto error
echo %APPLET%.scr created: OK
echo .
echo Completing script...
copy /A /Y %MISC%\Header.scr + %OUT%\PROJECT%\APPLET%.scr + %MISC%\Install.scr + %MISC%\Footer.scr %OUT%\PROJECT%\SIMUSCRIPT%.scr
del %OUT%\PROJECT%\APPLET%.scr
echo %SIMUSCRIPT%.scr created: OK
echo .
if errorlevel 1 goto error
goto end
:error
echo *****
echo ERROR !
echo *****
pause
goto end
:end
cls
```

⇒ 3.2.3.4 makeClient.bat

Ce script compile le client qui échange des APDU avec le programme JavaCard situé dans la carte, réelle ou simulée.

```
@echo off
call .\setenv.bat
rem opencard.core.*
set CLASSES=%CLASSES%;%OCF_HOME%\lib\base-core.jar
rem opencard.opt.util
set CLASSES=%CLASSES%;%OCF_HOME%\lib\base-opt.jar
echo Compilation...
%JAVA_HOME%\bin\javac.exe -classpath %CLASSES% -g -d %OUT%\PROJECT%\SRC%\PROJECT%\PKGCLIENT%\CLIENT%.java
if errorlevel 1 goto error
echo %CLIENT%.class compiled: OK
echo .
goto end
:error
echo *****
echo ERROR !
echo *****
pause
goto end
:end
cls
```

⇒ 3.2.3.5 runClient.bat

Ce script exécute le client compilé et le place en attente, soit de l'insertion d'une vraie carte Java, soit du lancement du simulateur de carte Java.

```
@echo off
call .\setenv.bat

set CLASSPATH=%JC21_HOME%\lib\apduio.jar
set CLASSPATH=%CLASSPATH%;%OCF_HOME%\lib\base-core.jar
set CLASSPATH=%CLASSPATH%;%OCF_HOME%\lib\base-opt.jar
set CLASSPATH=%CLASSPATH%;%MISC%\PCSC_WRAPPER%\lib\PCSC_WRAPPER%.jar
set CLASSPATH=%CLASSPATH%;%MISC%\APDUIO_TERM%\lib\APDUIO_TERM%.jar

%JAVA_HOME%\bin\java.exe -Djava.library.path=. -classpath %OUT%\%PROJECT%\
%CLASSPATH% %PKGCLIENT%.CLIENT%
if errorlevel 1 goto error

pause
goto end

:error
echo *****
echo ERROR !
echo *****
pause
goto end

:end
cls
```

⇒ 3.2.3.6 runSimulator.bat

Ce script lance le simulateur. Ce dernier aura dû être au préalable initialisé en exécutant tour à tour les scripts `simulator-init.bat` et `simulator-installApplet.bat` préalables.

```
@echo off
call .\setenv.bat

%JC21_HOME%\bin\cref.exe -nobanner -nomeminfo -i %OUT%\%PROJECT%\leeprom
if errorlevel 1 goto error

goto end

:error
echo *****
echo ERROR !
echo *****
pause
goto end

:end
cls
```

⇒ 3.2.3.7 simulator-init.bat

Ce script place le simulateur en attente d'initialisation. Il faut alors exécuter `simulator-installApplet.bat` pour réellement l'initialiser.

```
@echo off
call .\setenv.bat

%JC21_HOME%\bin\cref.exe -nomeminfo -nobanner -o %OUT%\%PROJECT%\leeprom
if errorlevel 1 goto error

goto end

:error
echo *****
echo ERROR !
echo *****
pause
goto end

:end
cls
```

⇒ 3.2.3.8 simulator-installApplet.bat

Ce script installe dans le simulateur, placé au préalable dans un état d'attente par `simulator-init.bat`, l'applet compilée. À la fin de l'installation, les fenêtres des deux processus doivent se fermer en cas de succès.

```
@echo off
call .\setenv.bat

REM apduio
%JAVA_HOME%\bin\java.exe -noverify -classpath %JC21_HOME%\lib\apduio.jar;%JC21_
HOME%\lib\apduio.jar com.sun.javacard.apduio.Main -nobanner %OUT%\%PROJECT%\
%SIMUSCRIPT%.scr

if errorlevel 1 goto error

goto end

:error
echo *****
echo ERROR !
echo *****
pause
goto end

:end
cls
```

⇒ 3.2.3.9 card-installApplet.bat

Ce script installe dans la carte définie dans `setenv.bat` le fichier `.CAP` compilé qui contient l'applet.

Attention, il faut penser à effacer la précédente instance de l'applet en cas de mise à jour de l'applet dans la carte, à l'aide du script `card-deleteApplet.bat`. Il est à noter que la carte Java est une carte multi-application et qu'il est possible de charger plusieurs applets en mémoire, pourvu qu'elles aient chacune leur numéro AID propre, configuré dans `setenv.bat`.

```
@echo off
call .\setenv.bat

%JAVA_HOME%\bin\java.exe -jar %CAPTRANSF%\captransf.jar %JC21_EXP%\OUT%\
%PROJECT%\%PKGAPPLET%\javacard\%PKGAPPLET%.cap

echo mode_201 > %OUT%\config.txt
echo establish_context >> %OUT%\config.txt
echo card_connect >> %OUT%\config.txt
echo select -AID %CARDSECURITYDOMAIN2% >> %OUT%\config.txt
echo open_sc -security 1 -keyind 0 -keyver 0 -mac_key %CARDKEY% -enc_key
%CARDKEY% >> %OUT%\config.txt
echo install_for_load -pkgAID %PACKAGEAID% -nvCodeLimit 500 >> %OUT%\config.txt
echo load -file %OUT%\%PROJECT%\%PKGAPPLET%\javacard\%PKGAPPLET%.cap.transf >>
%OUT%\config.txt
echo install_for_install -instParam 00 -priv 02 -AID %APPLETAID% -pkgAID
%PACKAGEAID% -instAID %APPLETAID% -nvDataLimit 500 >> %OUT%\config.txt
echo card_disconnect >> %OUT%\config.txt
echo release_context >> %OUT%\config.txt

%GPSHELL%\gpsshell.exe < %OUT%\config.txt
del %OUT%\config.txt
```

⇒ 3.2.3.10 card-deleteApplet.bat

Ce script permet d'effacer l'applet précédemment chargée dans la carte, tout du moins celle dont le numéro AID est indiqué dans le fichier `setenv.bat` courant.

```
@echo off
call .\setenv.bat

echo establish_context > %OUT%\config.txt
echo card_connect >> %OUT%\config.txt

echo select -AID %CARDSECURITYDOMAIN2% >> %OUT%\config.txt
echo open_sc -security 1 -keyind 0 -keyver 0 -mac_key %CARDKEY% -enc_key
%CARDKEY% >> %OUT%\config.txt
echo delete -AID %APPLETAID% >> %OUT%\config.txt
echo delete -AID %PACKAGEAID% >> %OUT%\config.txt
echo card_disconnect >> %OUT%\config.txt
echo release_context >> %OUT%\config.txt

%GPSHELL%\gpsshell.exe < %OUT%\config.txt
del %OUT%\config.txt
```

⇒ 4. Utilisation du SDK sur HelloWorld

Pour tester le SDK, nous créons un projet minimaliste nommé fort à propos « mini », qui comprendra les deux sources de l'applet et du client d'une application carte complète mettant en œuvre une carte Java (simulée ou réelle).

Pour ce faire, créez un sous-répertoire `JavaCardSDK\src\mini\`, puis deux sous-répertoires `JavaCardSDK\src\mini\applet\` et `JavaCardSDK\src\mini\client\`, qui contiendront respectivement le source de l'applet (`TheApplet.java`, écrit en JavaCard) et du client (`TheClient.java`, écrit en Java avec OCF).

⇒ 4.1 TheApplet.java

Cette applet est minimaliste, elle se contente de renvoyer un APDU de réponse contenant les octets du tableau `msg` qui représente une chaîne de caractères, quel que soit l'APDU de commande envoyé par l'hôte. Dans cet exemple, la carte contient la chaîne "Hello world!".

```
package applet;
import javacard.framework.*;

public class TheApplet extends Applet {

    protected byte[] msg={{(byte)12,'H','e','l','l','o',' ','w','o','r','l','d','!'}};

    protected TheApplet() {
        this.register();
    }

    public static void install(byte[] bArray, short bOffset, byte
    bLength) throws ISOException {
        new TheApplet();
    }

    public boolean select() {
        return true;
    }

    public void process(APDU apdu) throws ISOException {
        byte[] buffer = apdu.getBuffer();
        if( selectingApplet() == true )
            return;

        Util.arrayCopy(msg, (short)1, buffer, (short)0, msg[0]);
        apdu.setOutgoingAndSend( (short)0, msg[0] );
    }
}
```

⇒ 4.2 TheClient.java

Ce programme Java avec OCF est très simple. Il attend l'évènement insertion d'une carte (ou démarre si une carte est déjà présente dans le lecteur à l'exécution du client), puis affiche l'ATR caractéristique de la carte insérée (réelle ou simulée), tente de sélectionner l'applet en question et lui envoie en cas de succès un APDU de commande vide attendant en retour un nombre indéterminé d'octets (soit `0x00 0x00 0x00 0x00 0x00`).

```
package client;
import java.io.*;
import opencard.core.service.*;
import opencard.core.terminal.*;
import opencard.core.util.*;
import opencard.opt.util.*;

public class TheClient {

    private PassThruCardService servClient = null;

    boolean DISPLAY = true;

    public TheClient() {
        try {
            SmartCard.start();
            System.out.print( "Smartcard inserted?... " );

            CardRequest cr = new CardRequest( CardRequest.ANYCARD,null,null);

            SmartCard sm = SmartCard.waitForCard( cr);

            if (sm != null) {
                System.out.println( "got a SmartCard object!\n");
            } else
                System.out.println( "did not get a SmartCard object!\n" );

            this.initNewCard( sm );

            SmartCard.shutdown();

        } catch( Exception e ) {
            System.out.println( "TheClient error: " + e.getMessage() );
        }
        java.lang.System.exit(0) ;
    }

    private ResponseAPDU sendAPDU(CommandAPDU cmd) {
        return sendAPDU(cmd, true);
    }

    private ResponseAPDU sendAPDU( CommandAPDU cmd, boolean display ) {
        ResponseAPDU result = null;
        try {
            result = this.servClient.sendCommandAPDU( cmd );
            if(display)
                displayAPDU(cmd, result);
        } catch( Exception e ) {
            System.out.println( "Exception caught in sendAPDU: " + e.getMessage() );
            java.lang.System.exit( -1 );
        }
        return result;
    }

    /***** BEGINNING OF TOOLS *****/
    private String apdu2string( APDU apdu ) {
        return removeCR( HexString.hexify( apdu.getBytes() ) );
    }

    public void displayAPDU( APDU apdu ) {
        System.out.println( removeCR( HexString.hexify( apdu.getBytes() ) ) + "\n" );
    }

    public void displayAPDU( CommandAPDU termCmd, ResponseAPDU cardResp ) {
        System.out.println( "--> Term: " + removeCR( HexString.hexify( termCmd.getBytes() ) ) );
        System.out.println( "<-- Card: " + removeCR( HexString.hexify( cardResp.getBytes() ) ) );
    }

    private String removeCR( String string ) {
        return string.replace( '\n', ' ' );
    }

    /***** END OF TOOLS *****/

    private boolean selectApplet() {
        boolean cardOk = false;
        try {
            CommandAPDU cmd = new CommandAPDU( new byte[] {
                (byte)0x00, (byte)0xA4, (byte)0x04, (byte)0x00, (byte)0x0A,
            }
        )
    }
}
```

```

        (byte)0xA0, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x62,
        (byte)0x03, (byte)0x01, (byte)0x0C, (byte)0x06, (byte)0x01
    });
    ResponseAPDU resp = this.sendAPDU( cmd );
    if( this.apdu2string( resp ).equals( "90 00" ) )
        cardOk = true;
    } catch( Exception e ) {
        System.out.println( "Exception caught in selectApplet: " + e.getMessage() );
        java.lang.System.exit( -1 );
    }
    return cardOk;
}

private void initNewCard( SmartCard card ) {
    if( card != null )
        System.out.println( "Smartcard inserted\n" );
    else {
        System.out.println( "Did not get a smartcard" );
        System.exit( -1 );
    }

    System.out.println( "ATR: " + HexString.toHexString( card.getCardID().getATR() ) + "\n\n" );
    try {
        this.servClient = (PassThruCardService)card.getCardService(
            PassThruCardService.class, true );
    } catch( Exception e ) {
        System.out.println( e.getMessage() );
    }

    System.out.println( "Applet selecting..." );
    if( !this.selectApplet() ) {
        System.out.println( "Wrong card, no applet to select!\n" );
        System.exit( 1 );
        return;
    } else
        System.out.println( "Applet selected\n" );

    byte[] cmd_ = {0,0,0,0,0};
    CommandAPDU cmd = new CommandAPDU( cmd_ );
    System.out.println( "Sending blank command APDU, data expected..." );
    ResponseAPDU resp = this.sendAPDU( cmd, DISPLAY );
    byte[] bytes = resp.getBytes();
    String msg = "";
    for( int i=0; i<bytes.length-2; i++ )
        msg += new StringBuffer( "" ).append( (char) bytes[i] );
    System.out.println( msg );
}

public static void main( String[] args ) throws InterruptedException {
    new TheClient();
}
}

```



4.3 Showtime!

Afin de s'assurer de l'état initial du SDK, on commence par exécuter `cleanAll.bat` qui réinitialise le SDK en effaçant les fichiers précédemment générés.

Ensuite, on exécute `makeApplet.bat` et `makeClient.bat`, les fenêtres de ces processus devant se fermer si aucune erreur n'est détectée.

À ce moment, le fichier applet au format `.CAP` est prêt à être chargé dans une carte Java ou dans le simulateur. Il a été copié dans `JavaCardSDK\out\mini\`.

Pour utiliser le simulateur, il faut d'abord l'initialiser en exécutant `simulator-init.bat`, puis `simulator-installApplet.bat`, les fenêtres de ces processus devant se fermer si aucune erreur n'est détectée.

Pour utiliser une vraie carte Java, alors il faut exécuter `card-installApplet.bat` ou bien utiliser un loader particulier pour charger le fichier `JavaCardSDK\out\mini\applet.cap` dans la carte.

Enfin, on exécute `runClient.bat`, puis on insère la carte réelle dans le lecteur ou bien on exécute `runSimulator.bat` pour la carte simulée. La figure qui suit illustre le résultat obtenu.

On retrouve bien l'affichage de l'ATR spécifique à la carte réelle ou virtuelle insérée, suivi de la sélection réussie de l'applet (APDU de retour `0x90 0x00`), puis enfin l'envoi à la carte d'un APDU de commande vide attendant des données en retour, et la carte qui renvoie les octets correspondant à la chaîne de caractères "Hello world!".

Le programme s'est donc déroulé sans anicroche.

Conclusion

Avec ce SDK fait à partir de morceaux de logiciel librement téléchargeables, nous sommes en mesure de prototyper une application carte Java avec la possibilité de la tester sur simulateur au cours du développement, et « en vrai » sur une carte Java réelle à tout moment, sans avoir à modifier le code du client.

La conception même de ce SDK fait qu'il n'a pas à être installé. En effet, les chemins utilisés dans les scripts étant tous relatifs au répertoire racine `JavaCardSDK`, il suffit de copier ce répertoire sur un ordinateur pour pouvoir ensuite utiliser le SDK immédiatement, l'archive contenant le tout (y compris le JDK au complet) ne dépassant pas 90 Mo. ■

Remerciements

Je tiens sincèrement à remercier les différentes entreprises fabricantes de cartes Java (et de lecteurs de cartes à puce) qui m'ont généreusement fourni matière à tester et indirectement poussé à développer cet environnement de développement générique en remplacement des leurs.

Références

Vous pouvez retrouver les références de cet article sur : www.miscmag.com