

GNU

LINUX

MAGAZINE / FRANCE

HORS-SÉRIE

Administration et développement sur systèmes UNIX

HACK / BONUS

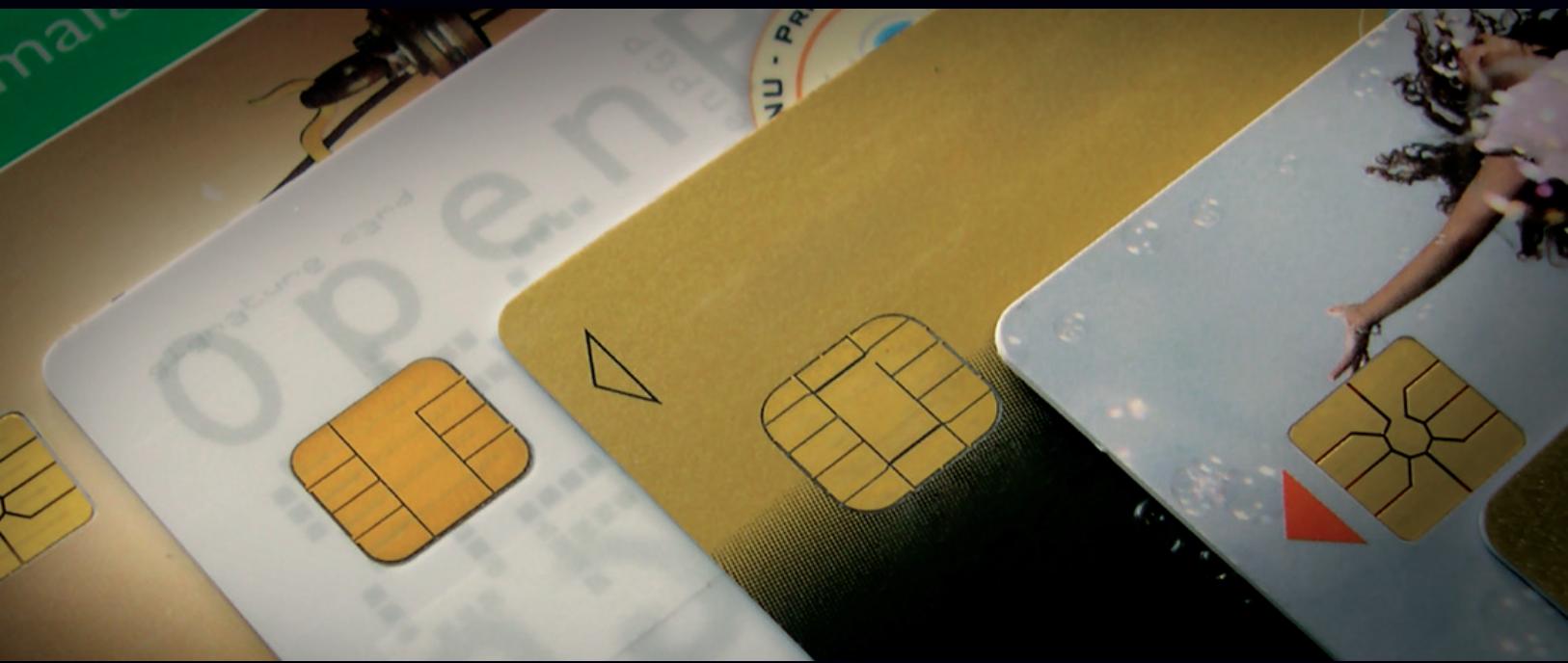


Lisez et exploitez les données RFID avec une carte son, Audacity et Octave (p. 72)



CARTES À PUCE

ADMINISTRATION ET UTILISATION



PROGRAMMATION

► Programmez des applications carte en Perl, Python, Ruby, Java, Caml, Prolog... (p. 51)

SYSADMIN

► Installation, configuration et utilisation des cartes à puce et tokens avec SSH, VPN, Firefox... (p. 60)

TECHNOLOGIE

► Explorez le contenu de votre carte bancaire avec les outils PC/SC Lite (p. 10)



MISC HORS-SÉRIE N°2

SPÉCIAL CARTES À PUCE

Disponible dès le 14/11/2008*
chez votre marchand de journaux



*Comprenez
et utilisez ces
technologies
pour assurer
votre sécurité !*

The cover features a green globe icon and the text "MISC Hors-Série". It includes sections for "PROGRAMMATION" (JavaCard programming), "DOSSIER" (Smart Cards), and "HISTORIQUE" (YesCard). A large headline reads "CARTES À PUCE DÉCOUVREZ LEURS FONCTIONNALITÉS ET LEURS LIMITES". Below the cover is a photograph of several gold-colored smart cards.

HISTORIQUE	SÉCURITÉ	TECHNOLOGIES
Retour sur la YesCard, un aperçu du système bancaire français	Mise en place d'infrastructures de gestion de clés (PKI) utilisant des cartes à puce	MIFARE classic, comment une faille compromet la sécurité de milliards de cartes !

Visitez www.ed-diamond.com pour en savoir plus !

*sous réserve de toutes modifications

SOMMAIRE

ÉDITO

INTRODUCTION

- p. 04 ■ Petite histoire illustrée de la carte à puce, par son inventeur
- p. 07 ■ Les menaces qui pèsent sur la carte à puce

TECHNO

- p. 10 ■ Exploration de carte bancaire BO'
- p. 14 ■ La carte d'identité électronique ou eID : une carte à puce dans tous les portefeuilles belges
- p. 18 ■ Le projet MUSCLE ou la bibliothèque PC/SC Lite

CODE(S)

- p. 28 ■ Développement d'applications pour la carte à puce en langage Python
- p. 40 ■ Mise en place d'un environnement complet de développement d'applications carte pour systèmes UNIX
- p. 51 ■ Wrappers PC/SC ou comment se passer du langage C pour accéder aux cartes à puce

SYSADMIN

- p. 60 ■ Introduction à l'utilisation de smartcards sous GNU/Linux
- p. 70 ■ PAM + USB, l'authentification matérielle du pauvre

HACK / BONUS

- p. 72 ■ Analyse des étiquettes d'identification par radiofréquence (RFID)

ABONNEMENT

- p. 79, 81, 82 ■ Bons d'abonnement et de commande

Ce magazine est édité par Gabessolo (gabessolo@yahoo.fr) le 6 Février 2017

GnLinux Magazine France Hors-série 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 : lecteurs@gnlinuxmag.com
Service commercial : abo@gnlinuxmag.com
Site : www.gnlinuxmag.com
www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Wilhelm
Rédaction : Dominique Grossé
Conception graphique : Fabrice Krachenfels
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08

Impression : VPM Druck 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
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution / N° ISSN : 1291-78 34
Commission paritaire : 09 08 K78 976
Périodicité : Bimestrielle
Prix de vente : 6,50 €
à Yann

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 Linux Magazine France est interdite sans accord écrit de la société Diamond Editions. Sauf accord particulier, les manuscrits, photos et dessins adressés à Linux Magazine France, publiées 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.



Cher lectorat cartophile (ou tout du moins curieux de la carte), bonjour !

Voici donc un numéro hors-série orienté « carte ». Ces dernières années, les cartes à puce se sont immiscées dans notre quotidien pour ne plus en ressortir. Pour s'en convaincre, il suffit de vider son portefeuille et de compter le nombre de cartes à puce (avec ou sans contacts galvaniques apparents) qui le peuplent.

Ce numéro hors-série tente de donner quelques clefs pour permettre d'utiliser les cartes à puce de manière proactive dans les environnements informatiques, et plus particulièrement sur les systèmes GNU/Linux et autres compatibles UNIX.



J'en profite pour vous signaler la sortie prochaine du numéro hors-série « carte à puce » de MISC, à paraître mi-novembre prochain. Complémentaire au numéro présent, il traite du sujet d'une manière plus théorique et centrée sur la sécurité.

Le monde de la carte à puce a des analogies avec celui du logiciel propriétaire. En effet, s'il est relativement aisés de simplement utiliser une carte à puce pour des besoins bien définis (bancaires, télécoms, médical, etc.), il est plus difficile d'accéder aux informations permettant d'en prendre le contrôle pour l'utiliser à des fins plus spécifiques. À la lumière de ce constat, le contenu de ce magazine, futur collector, prend une valeur toute particulière.

Je remercie les auteurs qui ont contribué par leurs articles à la qualité de ce numéro. Qu'ils excusent mes relectures peut-être parfois un peu trop pointilleuses.

Jusqu'à récemment, l'acquisition de matériel permettant la programmation directe de cartes à puce était une entreprise compliquée. Il fallait être fabricant de cartes pour pouvoir en modifier le cycle d'exécution interne. L'émergence de nouvelles technologies a facilité cette programmation et désormais on peut acquérir à travers Internet des kits offrant la possibilité de s'essayer à la programmation de cartes à puce. Ce nouvel état de fait laisse augurer des articles à venir illustrant ces nouvelles possibilités offertes aux développeurs.

Cartement vôtre,

Vincent.GUYOT@{esiea, lip6}.fr

PS: merci à tous ceux, nombreux et parfois dans l'ombre, qui ont permis cette réalisation, et tout particulièrement Marie-Claire et Thierry.

INTRODUCTION



Il y a quelques années maintenant, la mise au point d'une nouvelle technologie posait les bases de ce qui allait devenir une industrie mondiale. Aujourd'hui, la carte à puce a investi notre quotidien de manière durable. Son inventeur, Roland Moreno, revient pour nous sur cette période où tout a commencé.

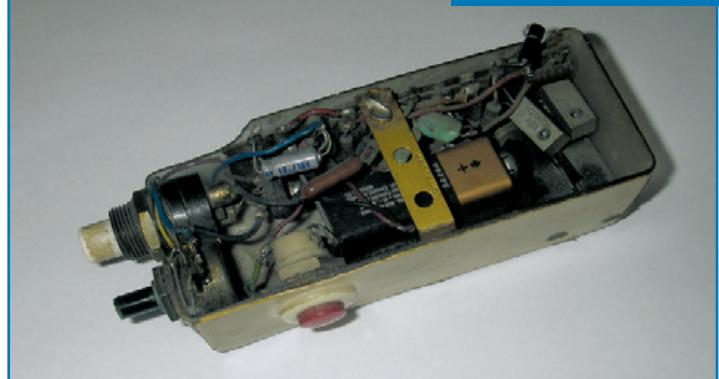
Petite histoire illustrée de la carte à puce, par son inventeur

Auteur : Vincent Guyot

C'est un autodidacte de l'électronique qui est le père de la carte à puce. En effet, Roland Moreno se destinait initialement aux lettres, avant de se passionner pour l'électronique. Dès 1964, il dépose un brevet pour calculer automatiquement la vitesse moyenne en voiture, à partir de la célèbre relation électrique $U=R.I$. Cette invention ne rencontrera pas le succès escompté, la faute à l'industrie automobile et sa forte inertie.

Roland Moreno est un inventeur. Depuis ses dix ans, il ne cessera d'inventer des choses à l'utilité discutable. Sa « Matapof » (voir figure 1), une machine à tirer à pile ou face, lui donne ses premiers gallons médiatiques dès 1968.

Figure 1 : La Matapof



Pour la petite histoire, c'est en fumant des substances « exotiques » que lui vient l'idée de la carte à puce, en début d'année 1974. À ce moment, la carte à puce n'est d'ailleurs encore qu'une sorte de « bague à puce » (voir figure 2).

Figure 2 : La bague originelle



INTRODUCTION

En effet, après avoir appris l'existence et la disponibilité de composants de mémoire électronique, Roland Moreno a l'idée de monter une puce électronique de mémoire sur une chevalière, de manière à l'avoir toujours sur soi. Cette chevalière est munie de contacts similaires aux contacts galvaniques des cartes à puce actuelles. Elle est destinée à être insérée dans un lecteur capable d'en lire la mémoire, déjà à l'époque, à l'image d'un porte-monnaie électronique.

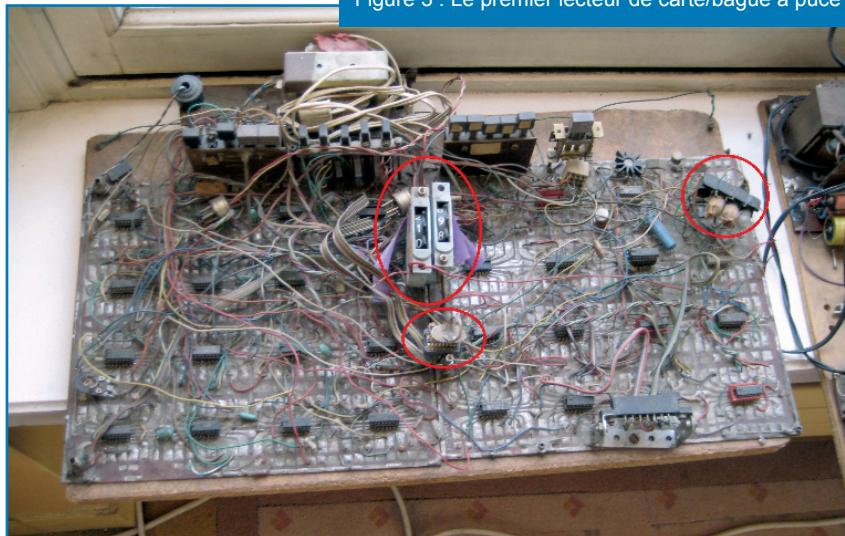


Figure 3 : Le premier lecteur de carte/bague à puce

noms de sociétés sur mesure ; il vend aussi, à bon prix, des actions de sa société **Innovatron**.

La première carte à puce ne ressemble pas vraiment aux cartes que nous côtoyons aujourd'hui (voir figure 4). Elle n'est pas du tout flexible, mais au contraire très rigide et cassante, comme tout circuit imprimé.

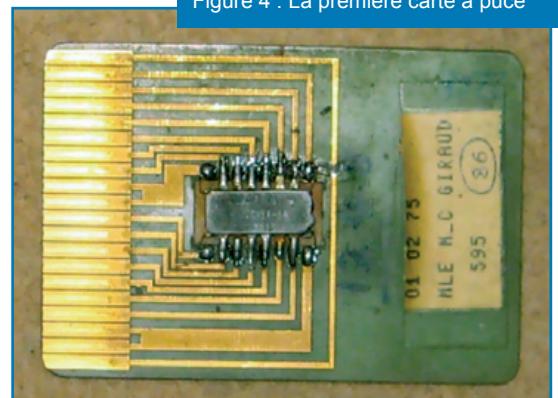


Figure 4 : La première carte à puce

Il faut dire qu'elle est destinée à fonctionner dans un lecteur de plusieurs dizaines de kilos, où circule une tension de 21 volts. Sur la figure 5, on peut voir un tel lecteur, avec le clavier de saisie et la fente d'insertion de la carte encadrés de rouge. Ce lecteur de cartes à puce est l'ancêtre du TPE (Terminal de Paiement Électronique), encore à venir.



Figure 5 : Le premier lecteur de carte à puce

Entourés de rouge sur la figure 3, on reconnaît les différents éléments qui préfigurent l'utilisation moderne de la carte à puce. Au centre se situent deux roues crantées qui servent à entrer le code PIN, à deux chiffres, ainsi que le montant de la transaction, lui aussi sur deux chiffres. En dessous, on voit la bague reposant sur son socle-lecteur. Plus à droite du montage électronique, se trouvent deux petites lampes s'éclairant selon l'acceptation ou non de la commande effectuée, en fonction de la validité du code PIN et du crédit disponible dans la mémoire de la bague.

Ce montage est présenté début 1974 aux banques françaises, qui cherchaient alors un moyen de rendre plus sûre la carte bancaire. À ce moment, il s'agit d'une simple mémoire électronique sans aucun moyen de protection, le terminal contrôlant le déroulement des opérations. Le très grand intérêt porté par les banques à son invention naissante conforte Roland Moreno dans son idée.

À cette époque, sa société Innovatron n'a pas de capital propre et poser un brevet est inconcevable.

Il va néanmoins réussir à déposer en 1974 son premier brevet relatif à une carte à puce à mémoire, avec l'aide de François Marquer, un ingénieur en brevet qui lui fait crédit.

Jean-Pierre Leroy, électronicien génial, l'aide dans la concrétisation de ses idées folles.

Roland Moreno vit alors chichement des revenus d'une autre invention, le **Radoteur**, grâce à laquelle il crée et vend des

C'est grâce à cette forte tension que la sécurité est apportée à la carte. En effet, elle permet de brûler un fusible qui interdit alors toute modification ultérieure de la carte. Les brevets relatifs à la sécurité de la carte à puce à mémoire sont publiés en 1975, où Roland Moreno sécurise la carte avec ses célèbres **circuits inhibiteurs**. Sur la figure 6, on peut voir le schéma de cette carte à mémoire sécurisée avec le fameux fusible entouré de rouge à gauche du stylo, en forme de petit triangle inversé vert.

INTRODUCTION

Petite histoire illustrée de la carte à puce,
par son inventeur

C'est d'ailleurs à cause d'un défaut de réalisation des premiers lecteurs de cartes commercialisés qui ne fournissaient pas une tension assez élevée (en plus d'avoir un registre défaillant) que le code PIN pouvait être retrouvé. L'équipe technique de Roland Moreno avait mis en lumière ce problème et réalisé le premier « casseur » de la première génération de cartes bleues à puce (voir figure 7).

Cet appareil essaye tour à tour tous les codes PIN jusqu'à trouver le bon. L'absence de forte tension dans l'appareil empêche que ne brûle le fusible de protection en cas d'erreur, et le code PIN à 4 chiffres est révélé en quelques minutes de recherche exhaustive.



Figure 6 : La carte à puce à mémoire protégée

Professionnellement, Roland Moreno licencie aujourd'hui sa technologie Calypso, qui est l'exploitation commerciale de la technologie sans contact déjà présente dans son brevet de 1974. Cette technologie est par exemple utilisée en France dans les Pass Navigo et Vélib, et le sera dans les futurs Pass Moneo sans contact.

À titre plus personnel, Roland Moreno est désormais un unixien convaincu qui ne peut plus se passer de sa configuration bi-écran (pas moins de soixante pouces de diagonale à eux deux) « siglée » de la pomme (voir figure 8). Il continue à inventer, avec toujours plus

ou moins de succès, et espère sortir fin 2009 une nouvelle édition de son livre *La Théorie du bordel ambiant* (1990, aux éditions Belfond).

Pour plus de détails sur cette période, je vous renvoie à l'ouvrage de Roland Moreno *Carte à puce, l'histoire secrète* (2002, édité par l'Archipel).



Figure 8 : Roland Moreno



Figure 7 : Le premier « casseur » de carte bleue

Roland Moreno vend des licences. Malheureusement, son peu d'expérience dans les contrats le dessert. Il devra attendre de longues années avant de voir les premières rentrées d'argent arriver : 9 ans pour la carte à logique câblée, 16 ans pour la carte à microprocesseur.

À titre d'exemple de revenus engendrés par la carte à puce, la télécarte française utilisable dans les cabines téléphoniques de France Télécom à partir de 1983 était vendue chez les buralistes environ quarante francs français de l'époque. Sur cette somme, quatre francs étaient pour le buraliste, trois francs pour la régie publicitaire de la publicité imprimée sur la télécarte et seulement quatre centimes allaient dans la poche de Roland Moreno.

Depuis 1998, les brevets de la carte à puce ne lui rapportent plus de royalties.

Avec le recul, Roland Moreno reconnaît aujourd'hui qu'il n'a pas anticipé le marché énorme engendré par la carte SIM, suite à l'avènement de la téléphonie mobile. À l'époque, on pensait surtout aux chaînes de télévision payantes, marché qui n'a jamais vraiment décollé.

AUTEUR : VINCENT GUYOT



Professeur ingénieur à l'ESIEA, tout particulièrement au sein des mastères spécialisés « Sécurité de l'Information et des Systèmes » et « Network and Information Security », et chercheur collaborant avec le Laboratoire d'Informatique de Paris 6. Randonneur depuis 1994, mais pas au sens où sa femme l'entend...

REMERCIEMENTS

Je tiens à remercier M. Moreno d'avoir accepté de me recevoir pour l'entretien qui a servi de base à cet article, et de m'avoir laissé photographier ses morceaux d'histoire. Mes remerciements vont également à Mme Lucas, pour sa patience, sans qui cette rencontre n'aurait jamais pu avoir lieu.



La carte à puce est présente dans notre vie quotidienne lorsque nous utilisons notre téléphone portable ou lorsque nous payons par carte bancaire. Elle permet au titulaire de la carte de s'authentifier auprès de l'émetteur de la carte (un opérateur de téléphonie mobile ou une banque) pour obtenir un accès à des services.

Dans cet article, nous décrivons succinctement les caractéristiques techniques d'une carte à puce et présentons les principales attaques physiques qui peuvent être menées lors de son utilisation, ainsi que les protections usuelles permettant de s'en protéger.

Les menaces qui pèsent sur la carte à puce

Auteurs : Arnaud Boscher, Sylvain Gautier

1

La carte à puce

Une carte à puce est un objet portable permettant l'authentification de son propriétaire et le stockage d'informations. Elle est utilisée par l'industrie bancaire (carte bancaire et porte-monnaie électronique), par les opérateurs téléphoniques (carte SIM/USIM), par les États (passeports électroniques, cartes d'identités électroniques), en tant que titre de transport (Pass Navigo), par les organismes médicaux (carte Vitale), etc. Son usage est aujourd'hui très répandu et l'on considère que la très grande majorité des individus possède au moins une carte à puce sur lui.

Dans son utilisation la plus courante, un émetteur de cartes (une banque ou un opérateur de téléphonie mobile par exemple) fournit à son client une carte qui contient des éléments qui lui sont propres : les clés cryptographiques. Grâce à ces clés, l'émetteur de la carte pourra authentifier la carte à l'aide d'un protocole cryptographique et avoir ainsi l'assurance que la carte est valide. Le code PIN sert à débloquer l'utilisation de la carte, donc à identifier le porteur de la carte.

Une carte est composée d'un micro-processeur, d'une mémoire vive, d'une mémoire morte, d'une mémoire non volatile, de périphériques cryptographiques et d'éléments de communication avec le monde extérieur.

Le micro-processeur (ou CPU pour *Central Processing Unit*) a en général une architecture 8 bits (contre 32 ou 64 bits sur les PC) et est cadencé à une fréquence de quelques dizaines de MHz (contre plusieurs GHz sur les PC). La mémoire vive (ou RAM pour *Random Access Memory*) est limitée à quelques Ko (2 à 4 Ko). Elle contient les variables de travail nécessaires aux opérations de l'application. La mémoire morte (ou ROM pour *Read Only Memory*) contient le système d'exploitation de la carte. Sa taille varie entre 64 Ko et 256 Ko. La mémoire non volatile peut être une EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) ou une mémoire flash d'environ 64 Ko. Elle sert à contenir les informations personnelles du propriétaire de la carte : les clés cryptographiques, le code PIN, le numéro de la carte, les données personnelles du propriétaire...

Il s'agit donc d'un véritable ordinateur en miniature, mais dans un environnement à ressource très restreint permettant une fabrication à faible coût. De nouvelles applications telles que la télévision sur mobile nécessitant plus de puissance de calcul. Les cartes à puce récentes sont constituées d'un micro-processeur 32 bits, tel que l'ARM7, et embarquent des mémoires Flash de plus grande capacité (plusieurs Mo).

Les cartes à puce communiquent avec le monde extérieur (le téléphone portable ou le terminal de paiement, par exemple) à travers un protocole de communication qui peut être « contact » ou « sans contact ». Une carte contact nécessite un lien physique entre la carte et le lecteur, contrairement aux

cartes sans contact où la carte n'a besoin que d'être approchée à quelques centimètres du lecteur pour pouvoir communiquer. C'est le cas par exemple des Pass Navigo utilisés pour accéder au métro parisien.

2

Menaces contre les cartes à puce

Une carte à puce sert donc à protéger l'accès à des services, à des biens, et, par conséquent, peut constituer une cible pour des personnes mal intentionnées désirant obtenir ces services de manière illégale.

Le but pour un attaquant est donc de retrouver les clés cryptographiques utilisées de manière à pouvoir les réutiliser. Si l'algorithme cryptographique est de mauvaise qualité, des attaques mathématiques sont envisageables sur l'algorithme lui-même. En connaissant des entrées et des sorties de l'algorithme, il est alors possible de retrouver de l'information sur la clé utilisée. Heureusement, la plupart des applications carte à puce utilisent des algorithmes cryptographiques standardisés comme le 3DES (*Triple Data Encryption Standard*), l'AES (*Advanced Encryption Standard*) ou le RSA (du nom de ses inventeurs Rivest, Shamir et Adleman) contre lesquels aucune attaque n'est connue à ce jour.

L'algorithme étant considéré sûr d'un point de vue mathématique, un autre chemin d'attaque est de se concentrer sur son implémentation dans un module de chiffrement. Les attaques présentées ci-dessous sont dénommées « attaques par canaux cachés ». Les canaux cachés désignent les informations fournies par le module de chiffrement autres que le message en clair (en entrée de l'algorithme) et le message chiffré (en sortie). En effet, les modules de chiffrement mettent un certain temps pour exécuter l'algorithme, consomment du courant et émettent des rayonnements électromagnétiques.

2.1

Attaque temporelle

Un premier exemple d'attaque par canaux cachés est illustré par les **attaques temporelles** (ou **Timing Attacks** en Anglais). Le principe consiste à mesurer le temps d'exécution d'un algorithme pour essayer d'en déduire de l'information sur la clé inconnue utilisée par l'algorithme cryptographique. En effet, si le temps de calcul est plus ou moins long selon la valeur de clé secrète, cela permet de restreindre les clés possibles. Comme exemple, considérons le cas suivant de l'implémentation de la vérification d'un code PIN à 4 digits :

```
For (i = 0 ; i < 4 ; i++)  
{  
    If (PINCandidat[i] != PIN[i])  
        Return(False);  
}
```

Selon que le premier digit est correct ou non, le temps de réponse de la carte ne sera pas le même. Ainsi, avec une telle implémentation, en soumettant 10 essais avec le premier digit différent, une personne mesurant avec précision le temps de réponse de la carte à puce est en mesure de retrouver le premier digit du code PIN. Par conséquent, 40 essais maximum sont

nécessaires pour connaître entièrement le code PIN, alors que théoriquement il faudrait en moyenne 500 essais. Cette attaque pour retrouver le code PIN est uniquement théorique, car les cartes possèdent un compteur d'erreur de code PIN qui bloque la carte après un certain nombre d'essais infructueux (généralement 3). Mais, elle illustre parfaitement comment la mesure du temps d'exécution peut révéler des informations confidentielles.

2.2

Attaque par analyse de courant ou de rayonnement électromagnétique

Pour pouvoir fonctionner, une carte à puce a besoin d'une source extérieure de courant, fournie par le lecteur de carte. Cette source peut être observée par un attaquant, et même enregistrée, dans le but de retrouver les informations secrètes contenues dans la carte. Par exemple, l'écriture en RAM d'un octet nécessitera plus ou moins de courant selon la valeur à écrire et selon la valeur déjà présente en mémoire à cet emplacement.

On distingue deux types d'attaques par analyse de courant : les attaques simples (ou *Simple Power Analysis* en Anglais) et les attaques différentielles (ou *Differential Power Analysis* en Anglais).

2.2.1

Attaques simples (Simple Power Analysis)

Les attaques simples utilisent une seule courbe de consommation de courant, obtenue lors de l'exécution de l'algorithme. En effet, la consommation de courant varie selon les opérations élémentaires effectuées par l'algorithme : addition, multiplication, stockage en RAM, etc. Il est ainsi possible de visualiser l'exécution de l'algorithme et de retrouver les grandes parties qui le composent telles que des rondes du DES ou des étapes du RSA. Néanmoins, en pratique, les implémentations de ces algorithmes intègrent des contre-mesures décrites plus loin dans cet article.

2.2.2

Attaques différentielles (Differential Power Analysis)

Les attaques différentielles nécessitent l'enregistrement de plusieurs milliers de courbes de courant. Différents traitements statistiques sont ensuite appliqués sur ces courbes pour essayer de découvrir les données secrètes manipulées par la carte.

Lorsque la carte à puce effectue une authentification, elle émet également un rayonnement électromagnétique. Ce rayonnement peut également être mesuré et utilisé dans le but de retrouver les clés secrètes employées.

2.3

Attaque par fautes

Ces attaques nécessitent un équipement approprié (oscilloscope, sonde électromagnétique, cage de Faraday, etc.), afin d'observer et d'enregistrer proprement les signaux mesurés.

Il existe divers moyens pour perturber une carte pendant un calcul. Les deux principaux sont les suivants : appliquer une brusque variation de tension à la carte ou appliquer brièvement une source lumineuse (laser...). Cette dernière méthode nécessite un prétraitement chimique sur la carte, le but étant

d'enlever le plastique pour faire apparaître le composant et ses différents constituants : CPU, blocs mémoires, bus, etc.

Si la carte renvoie un résultat erroné à une authentification cryptographique, la connaissance de cette valeur erronée peut servir à retrouver de l'information sur la clé utilisée.

Une autre raison d'appliquer ce genre d'attaque est de faire accepter une authentification sans la connaissance des clés. À titre d'exemple, un attaquant pourrait imaginer de soumettre n'importe quel code PIN à la carte, appliquer une variation ou un flash lumineux au moment de la vérification de ce code de façon à ce que celle-ci ne soit pas effectuée. Ainsi, le faux code PIN présenté serait reconnu comme valide.

Le matériel nécessaire pour ces attaques est plus complexe que pour les attaques précédentes, surtout pour les attaques utilisant une source lumineuse.

3

Protections

Afin de se prémunir contre les attaques décrites précédemment, une carte à puce intègre de nombreux mécanismes de protection. Ces mécanismes se situent aussi bien au niveau logiciel qu'au niveau matériel.

Par exemple, pour rendre inefficace les attaques par analyse de temps, le système d'exploitation de la carte est implémenté de façon à ce qu'une opération soit effectuée à temps constant. Pour revenir à l'exemple de la vérification d'un code PIN, tous les digits sont vérifiés même si le premier est détecté comme erroné. Au niveau matériel, le micro-processeur rajoute aléatoirement quelques cycles de façon à désynchroniser l'exécution du code. Ainsi les différentes parties sensibles ne sont jamais exécutées exactement au même instant. Ce principe de désynchronisation peut également être implémenté au niveau logiciel.

En ce qui concerne les attaques observant les signaux émis pendant un calcul, l'implémentation du logiciel masquera les données secrètes avec des valeurs aléatoires afin d'éviter d'utiliser les véritables valeurs directement. Les fabricants de puces s'assurent que les diverses opérations ont à peu près la même signature électrique.

La protection classique contre les attaques par faute au niveau logiciel consiste à doubler les opérations critiques. Par exemple, le résultat du protocole d'authentification sera calculé deux fois et renvoyé uniquement si les deux résultats sont identiques. Ce n'est pas une protection parfaite puisque si un attaquant est capable de générer deux fois la même erreur lors de deux calculs, cette erreur ne sera pas détectée. Néanmoins, cela nécessite de provoquer deux erreurs, ce qui est plus compliqué à mettre en œuvre. C'est également une solution très coûteuse en temps.

Au niveau matériel, les composants pour carte à puce disposent de mécanismes supplémentaires par rapport à des microcontrôleurs classiques : des détecteurs de sécurité. Par exemple, un détecteur de tension est présent qui détecte toute variation brusque et informe le logiciel de cette détection, mettant ainsi en défaut l'attaque consistant à modifier la tension. De la même façon, des détecteurs de lumière sont présents. Concernant la protection contre le prétraitement chimique, les composants intègrent diverses couches de métal afin de rendre beaucoup plus difficile l'accès aux différents blocs mémoires. Ces couches de métal disposent également de mécanismes qui permettent d'informer le logiciel en cas de tentative d'enlèvement de ces couches.

4

Conclusion

Ces attaques demandent un certain investissement financier au niveau des équipements, mais ils sont tous disponibles dans le commerce. C'est pourquoi les cartes à puce actuelles intègrent ces contre-mesures. Afin de pouvoir mesurer le niveau de sécurité, les produits cartes à puce sont souvent évalués par des laboratoires indépendants selon différents schémas, le standard le plus répandu étant les critères communs, définis dans la norme ISO/CEI 15408. L'émetteur et le possesseur de la carte ont ainsi une idée du niveau de sécurité du produit.

RÉFÉRENCES

- RANKL (W.), EFFING (W.), *Smart Card Handbook*, Wiley, (http://www.mazon.fr/Smart-Card-Handbook-W-Rankl/dp/0470856688/ref=sr_1_1?ie=UTF8&s=english-books&qid=1221144103&sr=8-1)
- Cartes à puce : ISO/IEC 7816 www.iso.org
- Attaques physiques : <http://www.dpabook.org>
- Attaques physiques : http://www.crypto.rub.de/en_sclounge.html
- Attaques physiques : <http://www.sidechannelattacks.com/>
- Critères Communs : <http://www.commoncriteriaportal.org/>

AUTEURS :

Arnaud Boscher, Sylvain Gautier



Sachez pour commencer qu'il n'y a aucun risque physique pour votre carte à ce que vous regardiez le contenu mémoire de la carte (c'est-à-dire si vous vous limitez à effectuer des commandes de lecture).

La carte bancaire n'est autre qu'un programme exécuté par une carte à puce. Cette application respecte la norme de communication ISO-7816 : elle répond à une mise sous tension par le lecteur sous la forme d'un ATR (Answer To Reset) ; puis les échanges entre le lecteur et la carte sont au format APDU.

Exploration de carte bancaire B0'

Auteurs : Pierre Dusart, Damien Sauveron

1

Commande APDU

Sans entrer dans les détails, les commandes APDU envoyées par le lecteur sont composées de 5 paramètres (CLASse, INSTRUCTION, Paramètre1, Paramètre2, Longueur), chacun représenté par un entier sur 1 octet (0 à 255).

Pour la carte bancaire dotée de l'application bancaire française (masque B0'), le premier octet dans la commande APDU vaut BC en hexadécimal (classe CLA=0xBC).

Le deuxième octet INS dépend de la commande que l'on veut effectuer. Par exemple, la valeur INS vaut :

- 0xB0 pour la lecture d'octets ;
- 0x20 pour la présentation du code confidentiel (dit "PIN") ;
- 0x40 pour la validation (ratification code PIN).

Une commande complète (avec les 5 paramètres) peut être en notation hexadécimale :

BC B0 09 E0 02.

2

Réponse APDU

La carte répond à la commande

- par des données s'il y a lieu ;
- par un statut de transaction de 2 octets, par exemple 90 00 si tout s'est bien passé. Il existe un grand nombre de statuts d'erreurs dont certains sont documentés dans la norme ISO7816.

Pour une carte bancaire, la réponse à la commande APDU précédente (BC B0 09 E0 02) doit être 3F E5 90 00. En décomposant la commande, il s'agit d'une lecture (Instruction B0) à l'adresse mémoire 09 E0 de 02 octets. La carte répond naturellement les deux octets qu'elle a lus (3F E5) et le statut de la transaction (90 00). Cela est valable pour toutes les cartes bancaires françaises.

Cet échange permet au terminal de paiement de tester si, la carte insérée est de type « carte bancaire ». Par conséquent, si lors de l'expérimentation, vous n'obtenez pas ce résultat en utilisant votre carte, n'allez pas plus loin dans l'exploration et référez-vous à la remarque en conclusion de cet article.

Pour faire des transactions, vous aurez bien sûr besoin d'un lecteur de cartes et d'avoir installé au préalable pcsc-lite, le driver (IFD Handler) relatif à votre lecteur et éventuellement une application permettant de dialoguer avec la carte. Dans cet article, nous utiliserons l'outil scriptor (développé en perl et nécessitant donc l'installation du wrapper pcsc-perl) pour faire nos transactions. Un article dans ce numéro vous explique comment installer les différents logiciels.

À présent, si vous lancez l'outil scriptor, que le lecteur est connecté et que les logiciels cités ci-dessus sont présents et bien configurés, celui-ci attend que vous lui passiez une commande APDU pour l'envoyer au lecteur (et donc à la carte).

3

Lecture de la zone de pointeurs

La mémoire de la carte bancaire est découpée en différentes zones. Certaines zones sont en lecture libre (par exemple : la zone de lecture), d'autres sont à accès protégé (par PIN pour la zone de travail) et d'autres complètement secrètes (la zone secrète contenant les clefs).

Malheureusement, les adresses des différentes zones ne sont pas fixes. Elles dépendent du fabricant, de la carte, ...

Heureusement, une des zones, appelée « zone d'adresses » ou « zone de pointeurs », est à adresse fixe. Elle permet de retrouver les adresses des autres zones : la zone secrète, la zone de gestion, la zone confidentielle, la zone de travail, la zone de lecture et la zone de fabrication.

Voyons comment retrouver l'une de ces adresses : nous allons trouver dans la zone de pointeurs l'adresse ADL qui correspond à l'adresse mémoire de la zone de lecture. Les adresses se trouvent à la suite les unes des autres. La zone de pointeurs commence à l'adresse 09C0, mais la zone à lire pour trouver la valeur correspondante à ADL se trouve à l'adresse 09C8 (il y a auparavant l'adresse AD1 de la clef émetteur située

```
login@host:~$ scriptor
No reader given: using SCM SCR 3311 (21120644200173) 00 00
Using T=0 protocol
Reading commands from STDIN
```

Pour réaliser la transaction que nous venons d'étudier, tapez la commande APDU BCB009E002, puis appuyez sur la touche [entrée].

```
Transaction APDU / Lecture Carte CB
BCB009E002
> BC B0 09 E0 02
< 3F E5 90 00 : Normal processing.
```

dans la zone secrète). On lit 2 octets de l'adresse 09C8. La transaction est donc :

```
Transaction APDU / Lecture Zone Pointeur
BCB009C802
> BC B0 09 C8 02
< 1F FF 90 00 : Normal processing.
```

Cela pourrait être simple et on aurait pu obtenir directement l'adresse ADL. Mais, non ! En raison des contraintes de place et d'intégrité mémoire, l'adresse a été « compressée ». Il faut donc la décompresser en prenant les 11 premiers bits de la réponse et en concaténant deux zéros devant et trois zéros derrière. Le nombre hexadécimal 1FFF s'écrit en binaire 0001 1111 1111 1111 donc on n'utilise que les 11 bits de poids fort (0001 1111 1111), nombre auquel les zéros sont concaténés (0000011111110000). Ce nombre décompressé vaut 07F8 en hexadécimal. Il faut vraiment faire ce calcul car cette valeur change selon les cartes ; les valeurs possibles peuvent être également 07D0, 07F0 ou 08E0 sur d'anciennes cartes.

4

Lecture de la zone de lecture libre

Maintenant que l'on dispose de l'adresse ADL, regardons ce qu'il y a dedans ! Pour lire, la commande APDU est toujours de la même forme, soit BC B0 puis l'adresse de lecture (ici ADL = 07F8), puis le nombre d'octets à lire (0x10 soit 16 octets). Pour les cartes actuelles, on obtient ceci :

```
Transaction APDU / Lecture Zone ADL
BCB007F810
> BC B0 07 F8 10
< 2E 03 30 33 FF 90 00 : Normal processing.
```

Le statut est toujours 9000, ce qui correspond à une transaction sans statut d'erreur.

Le début de cette zone est constitué d'un en-tête de 4 octets. La première partie de l'en-tête est un tag valant 2E 03, appelé Prestataire 03 et qui contenait la fameuse valeur d'authentification de 320 bits, valeur qui fut réécrite sous forme d'une vraie fausse signature pour créer des Yescards [3]. Le problème est aujourd'hui réglé avec des FF partout. La taille de la zone se trouve dans l'en-tête juste après la valeur du tag, soit ici 0x30. La place mémoire réservée pour la valeur du champ Prestataire 03 est donc de 0x30, soit 48 octets.

A la suite du Prestataire 03 se trouve le Prestataire 02. C'est une zone qui contient le numéro de la carte, les dates de début et fin de validité, le nom du porteur, la devise. Il suffit donc de sauter les 4 octets d'en-tête (2E 03 30 33) et les 48 octets de contenu du Prestataire 03, pour arriver au Prestataire 02. Mais les adresses sont indexées en quartets, il faut donc multiplier par deux pour obtenir la bonne adresse. On doit trouver en repartant de l'adresse ADL : $(0x07F8)+(0x30+4)*2=0x0860$.

Une autre méthode est d'augmenter petit à petit la valeur de l'adresse mémoire dans la commande de lecture pour arriver au début du prestataire 02 : il commence par 2E 02.

```
Transaction APDU / Lecture Zone ADL
BCB008603C
> BC B0 08 60 3C
```

```
< 2E 02 38 F1 30 04 97 81 37 48 00 15 31 90 2F FF 32 01 07 02 32 50 09 04
32 50 54 97 34 D5 22 04 32 49 4C 4C 32 04 74 15 34 45 20 20 32 02 02 30
20 20 20 32 02 02 30 20 F1 52 90 00 : Normal processing.
```

Il faut éliminer les 3 de redondance tous les 8 quartets :

```
2e0238f1(3)0049781(3)7480015(3)1902fff(3)2010702(3)250
0904(3)2505497(3)4d52204(3)2494C4C(3)2047415(3)44520
20(3)2020202(3)0202020(3)2020202(3)020f1529000
```

Après l'en-tête, nous trouvons certaines données inscrites sur l'embossage de la carte : le numéro de la carte (4978174800151902), le code usage (201), la date de début de validité (0702), le code langue 250 (français), la date de fin de validité (0409), le code devise 250 (franc), l'exposant 5 (unité) puis 497. Le nom du titulaire de la carte se trouve à la suite en code ASCII : 4D5220 (MR)42494C4C20 (BILL) 4741544520 (GATES).

5

Lecture de la zone d'historique

Cette zone est à accès protégé. Elle n'est accessible pour la lecture qu'après la saisie du code confidentiel (PIN). Commençons par déterminer à quelle adresse se trouve cette zone. Cette adresse varie selon les cartes ; il faut donc rechercher cette adresse dans la zone de pointeurs. Elle correspond à l'adresse de la zone de Travail ADT. En lisant à l'adresse 09CC, on obtient l'adresse ADT qui nous intéresse après décompression (11 premiers bits gardés puis concaténation de deux zéros devant et 3 zéros derrière). On peut trouver comme adresse ADT des valeurs comme 02B0, 02F8, 0430, ...

Si on essaie de lire cette zone sans précaution :

```
Transaction APDU / Lecture Zone Historique
BCB002B03C
> BC B0 02 B0 3C
Can't get info: Transaction failed.
```

La communication avec la carte est interrompue. La carte n'a pas répondu (carte muette). La carte doit être remise à zéro (Il suffit de l'enlever du lecteur et de la remettre) pour pouvoir reprendre la conversation. Il faut donc absolument présenter une clef confidentielle (PIN) pour accéder à cette zone.

Le PIN n'est pas présenté directement. Il faut d'abord déterminer la séquence de présentation du PIN. Le PIN est constitué de 4 chiffres qui sont présentés sous la forme de 4 octets : cette fois, chaque chiffre est transformé en binaire sur 1 quartet (4 bits) pour obtenir au final une suite de 16 bits. On concatène deux zéros devant et deux « un » derrière et 3 séries de quatre « un » derrière. Cette valeur sera présentée en hexadécimal. Par exemple, un PIN de 1234 sera transformé en binaire : 0001 0010 0011 0100, puis avec les ajouts 00 0001 0010 0011 0100 11 1111 1111 1111, soit 0x048D3FFF. Après une conversion manuelle de votre propre PIN, il est conseillé de vérifier votre résultat en consultant le site www.parodie.com où un mini-convertisseur est fourni [2].

5.1

Présentation PIN

Il faut maintenant transmettre le PIN converti à la carte. La commande APDU sera BC20000004 (où 04 est la longueur des données à envoyer) suivie du PIN converti de 4 octets. Ainsi, pour un PIN de 1234 comme dans l'exemple précédent, la transaction sera :

```
Transaction APDU / Présentation PIN
BC20000004048D3FFF
> BC 20 00 00 04 04 8D 3F FF
< 90 00 : Normal processing.
```

Le statut 9000 confirme que tout s'est bien passé. On pourrait croire que c'est parce que le bon code PIN a été présenté. Il n'en est rien : avec un code PIN complètement faux, la carte aurait répondu de la même façon ! Elle confirme simplement que la commande a été bien reçue sous forme de 4 octets.

Ratification PIN : C'est au cours de cette étape que l'on demande à la carte de valider le PIN présenté. Cette commande, contrairement aux autres présentées plus haut, n'est pas sans danger ! Si la carte reçoit un faux PIN ou mal converti, elle indiquera que le code est faux et il ne restera plus que 2 essais pour fournir le bon PIN. Il faut se rappeler qu'au bout de 3 mauvais essais, la carte se bloque définitivement. Cette étape peut donc déverrouiller les zones mémoire en accès restreint si vous présentez correctement un bon PIN, mais à l'inverse peut également diminuer les essais de PIN encore possible. Toute présentation correcte remettra le compteur à 3 essais. Aussi, il suffit d'essayer une fois, si cela ne marche pas, un prochain achat en magasin ou retrait d'argent dans un distributeur automatique permet de remettre le compteur à la valeur maximale. Maintenant que vous êtes prévenu du risque que vous prenez en cas de mauvaise manipulation, la commande de ratification du PIN est :

```
Transaction APDU / Ratification PIN
```

```
BC40000000
> BC 40 00 00 00
Can't get info: Transaction failed.
```

Ici encore, la carte ne renvoie pas de statut de réponse et le lecteur renvoie une erreur (carte muette, non conforme avec la suite). Vous pouvez maintenant accéder en lecture aux zones à accès restreint. Pour cela, il faut relancer scriptor

```
sans avoir retiré la carte du lecteur. Ici, c'est l'adresse ADT (adresse de la zone de Travail) qui a été choisie :
```

```
Transaction APDU / Lecture Zone Travail
```

```
BCB002B020
> BC B0 02 B0 20
< 30 00 07 05 32 10 02 58 33 90 00 9D 30 00 07 06 33 78 00 40 33 78 00 C6
FF FF FF FF FF FF FF 90 00 : Normal processing.
```

Cette zone contient un petit historique des paiements qui ont été effectués.

6

Décodage de l'historique

Toujours pour des problèmes de place mémoire, l'historique a été compressé. Nous proposons le programme suivant (écrit en SciLab : www.scilab.org) pouvant être aisément converti dans d'autres langages si nécessaire. On pourra également utiliser le programme **ADT.exe** fourni dans le livre de P. Gueulle [1].

```
chaine = "30000705321002583390009d3000070633780040337800c6fffffffffffff9000";
// La chaine doit commencer par un 3
// sinon on s'est trompé dans l'adresse ADT
for i=1:length(chaine)/8,
bloc=part(chaine,1+(i-1)*8:8+(i-1)*8);
if part(bloc,1:4)=='3000',
annee=part(bloc,5:6);
mois=part(bloc,7:8);
elseif part(bloc,1)=='3',
jour=2*hex2dec(part(bloc,3));
valeur=hex2dec(part(bloc,4:8));
if(valeur>2^19),
jour=jour+1;
valeur=valeur-2^19;
```

```
end;
printf('%i/%s/%s : %5.0f F\n',jour,mois,annee,valeur)
end;
```

On obtient les transactions suivantes :

```
2/05/07 : 600 F
18/05/07 : 157 F
15/06/07 : 64 F
15/06/07 : 198 F
```

Deux remarques peuvent être faites : la compression choisie permet de gagner de la place et est complètement adaptée aux transactions bancaires. L'historique de cette carte s'est arrêté en juin 2007 alors que, pourtant, d'autres achats ont été faits depuis.

En fait, l'application bancaire française évolue et migre vers la nouvelle norme internationale orchestrée par trois sociétés : Europay, Mastercard et Visa (EMV) [4].

7

Valeur d'authentification

La valeur d'authentification contenue dans le prestataire 03 n'a plus été utilisée après l'affaire des Yescards [3]. Cette valeur reposait sur les secrets trop faibles mathématiquement. Elle a été remplacée par la valeur contenue dans le Prestataire 16 (qui se trouve à la suite du Prestataire 03). La taille de ce prestataire est de 0x80 octets (soit 128 octets). On peut donc stocker une valeur de 1024 bits, ce qui permet de s'affranchir de nouvelles Yescards pendant la période de transition vers la nouvelle norme EMV.

La carte testée est une carte sur laquelle cohabitent deux applications bancaires : l'application B0' et l'application EMV (carte mixte). Il semble que les nouvelles cartes émises soient des cartes 100% EMV sur lesquelles l'application B0' n'existe plus.

RÉFÉRENCES

- [1] Livre *PC et cartes à puce* de Patrick Gueulle, Editions ETSF
- [2] Site Parodie, <http://www.parodie.com/monetique/explorer.htm>

■ [3] Yescards, <http://fr.wikipedia.org/wiki/YesCard>

■ [4] EMV : Europay Mastercard Visa,
<http://www.emvco.com/>

AUTEURS : PIERRE DUSART, DAMIEN SAUVERON



Pierre Dusart - Maître de Conférences à l'Université de Limoges.



Damien Sauveron - Maître de Conférences à l'Université de Limoges. Vice-Président du Groupe de Travail 11.2 « Small Systems Security » de l'IFIP. Membre du Groupe de Travail 8.8 « Smart Cards » de l'IFIP.

Ancien contributeur sur le projet pcsc-lite
Plus d'information sur : <http://damien.sauveron.fr/>



Depuis 2004, la carte d'identité belge est progressivement remplacée par une version électronique – à travers une carte à puce – qui, à part être une carte d'identité, peut aussi être utilisée pour faire de nombreuses autres choses intéressantes.

La carte contient deux certificats électroniques, utilisables pour signer des emails ou s'authentifier auprès de sites web comme celui permettant de régler ses impôts en ligne en Belgique. Le logiciel nécessaire à l'usage de la carte est disponible pour Windows, GNU/Linux et Mac OS X. Comme c'est un logiciel libre, il existe des paquetages permettant de l'installer sous les principales distributions GNU/Linux (Debian, Ubuntu, Mandriva et SUSE/OpenSUSE).

La carte d'identité électronique ou eID : une carte à puce dans tous les portefeuilles belges

Auteur : Wouter Verhelst

1

Images numériques

Techniquement, cette carte d'identité est une carte Java qui implémente un large sous-ensemble des standards PKCS#11 et PKCS#15. Son usage nécessite néanmoins la modification des bibliothèques de cartes à puce, la raison étant que là où le propriétaire d'une carte à puce a le droit d'en modifier le contenu, il ne devrait pas être possible d'aucune sorte au possesseur d'une carte d'identité d'en modifier les données. La carte d'identité est un document du gouvernement et le fait de pouvoir la modifier permettrait, peut-être même encouragerait, les contrefaçons, ce qui n'est pas souhaitable. Les données officielles contenues dans la carte sont donc uniquement lisibles.

La carte eID contient tout d'abord toutes les données imprimées sur la carte elle-même : nom, prénom, date de naissance, genre, nationalité, commune d'émission, numéro d'identité national, date de validité, numéro de la carte ; ainsi que, eh oui, la photo du porteur de carte. La carte contient également d'autres données, non imprimées, car étant amenées à changer dans le temps, comme l'adresse du propriétaire de la carte.

Les clefs secrètes de la carte ont l'attribut **userConsent** activé, ce qui signifie que quel que soit l'usage qui en est fait, l'utilisateur doit avoir entré son code PIN avant de les utiliser, sinon la carte refusera d'effectuer l'opération. L'idée est que l'on ne devrait pas être en mesure de faire secrètement deux demandes de signature à la carte et seulement demander le code PIN une seule fois, créant ainsi une vraie-fausse signature d'un document jamais vu. Ceci est particulièrement important pour l'usage public de la carte eID, où la loi requiert l'utilisation de lecteurs de cartes munis de pinpad pour entrer le code PIN. Ces périphériques ignorent les codes PIN envoyés au lecteur par logiciel depuis l'ordinateur sur lequel ils sont branchés, rendant caduque l'attaque précédemment décrite. Néanmoins, cela signifie aussi que les bibliothèques interfaçant la carte doivent prendre cette particularité en compte. Le logiciel distribué sur le site web gouvernemental des cartes

eID le fait en ouvrant une boîte de dialogue qui demande de manière explicite d'entrer son code PIN lorsqu'il est demandé (voir Figure 1). À l'inverse, la distribution OpenSC gère cette situation en faisant échouer l'opération si l'utilisateur n'a pas entré son code PIN avant d'utiliser une clef secrète de sa carte. Des pilotes pour certains lecteurs de cartes ont été retirés du code source officiel, parce que le gouvernement a publié une spécification technique disponible pour les lecteurs de cartes officiellement supportés par leur logiciel, qui utilise PC/SC. On a supposé que les pilotes des lecteurs de cartes qui n'utilisaient pas ce *framework* n'étaient pas nécessaires. On peut remarquer que **libopencsc** et **libbeidlibopencsc** sont très similaires, le second étant en effet basé sur le premier.

Figure 1 : La boîte de dialogue qui s'affiche quand l'utilisation de clef d'authentification est requise



2

Deux certificats

La carte eID contient deux certificats électroniques. Le premier est le certificat d'authentification, utilisé dès que l'authentification est requise. On peut l'utiliser pour se connecter à un site web ou pour se connecter à un serveur SSH, après l'avoir converti en clef SSH.

L'autre certificat est le certificat de signature. Quand ce certificat est utilisé, une signature légale est créée, cela peut servir pour signer des emails ou des documents importants.

Les certificats peuvent être lus depuis la carte en utilisant l'outil **pkcs15-tool** d'OpenSC. Voici la commande, pour avoir la liste des certificats disponibles sur la carte :

```
wouter@country:~$ pkcs15-tool -c
X.509 Certificate [Authentification]
Flags : 3
Authority: no
Path : 3f00df005038
ID : 02
[...]
```

(la sortie écran contient également des données sur le certificat de signature, et deux certificats CA, que nous avons omis ici pour des raisons de concision)

Ensuite, trouvez la valeur du champ **ID** du certificat que vous voulez lire, et spécifiez-la à **pkcs15-tool -r** :

```
wouter@country:~$ pkcs15-tool -r 02 > eid.crt
```

Cela produira un certificat au format PEM. Si cela vous intéresse, vous pouvez obtenir des informations textuelles détaillées à partir du certificat, en utilisant le programme OpenSSL :

```
wouter@country:~$ openssl x509 -in eid.crt -noout -text
```

Bien que la carte eID contienne deux clefs privées, elles ne sont protégées que par un seul code PIN. Cela a été fait pour éviter toute confusion éventuelle. De plus, il n'est pas possible de chiffrer ou déchiffrer de données avec la carte, juste d'en signer.

3

En pratique

Bien sûr, tout cela ne sert à rien si on ne peut rien faire avec la carte. Le logiciel officiellement soutenu par le gouvernement possède une application de gestion graphique, un greffon Mozilla, deux bibliothèques de programmation, et quelques programmes en ligne de commande.

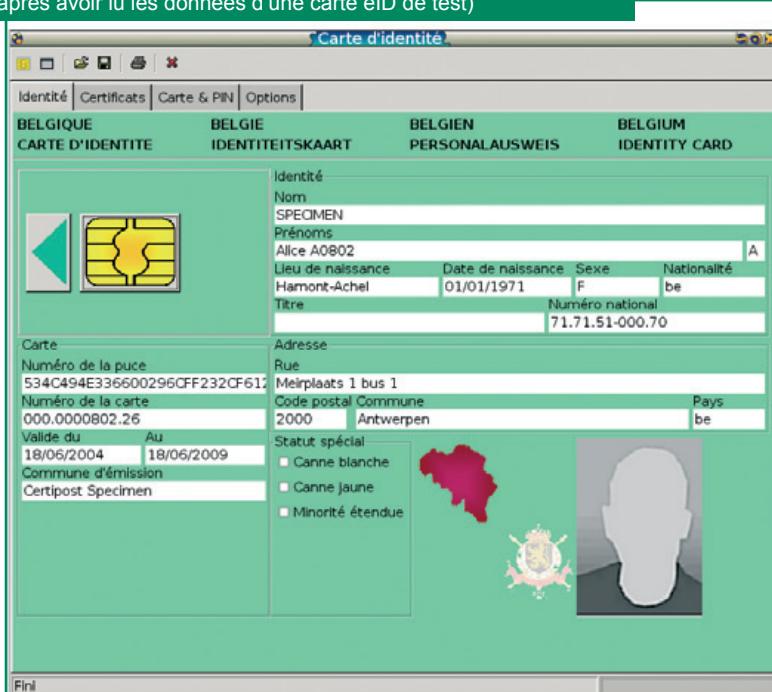
Les programmes en ligne de commande sont particulièrement pratiques pour déboguer. La plupart du temps, ils ne sont pas nécessaires. Les bibliothèques de programmation sont plus utiles. La première, **libbeidlibopencsc**, est la version modifiée de **libopencsc** que nous avons déjà décrite. Elle est

basée sur OpenSC 0.8, l'API étant exactement la même que celle de **libopencsc** de la même version. Avec la seconde, **libbeid**, nous avons une bibliothèque très simple qui fournit un petit nombre de fonctions vraiment restreint qui permet à l'utilisateur de lire les données identitaires de la carte eID. Elle utilise actuellement **libbeidlibopencsc** pour accéder à la carte.

L'outil de gestion graphique utilise **libbeidlibopencsc** et **libbeid** pour lire les informations des certificats de la carte, pour permettre à l'utilisateur de changer son code PIN, et de

lire les données identitaires de la carte. Les données d'identité peuvent alors être imprimées ou sauvegardées. Si vous voulez être capable de lire des données à partir des cartes d'identité de personnes et que vous n'avez aucun besoin spécifique particulier, alors cette application fera probablement ce dont vous avez besoin. Bien sûr, vous devriez faire attention à l'enregistrement de données des personnes, au regard de la loi belge sur la propriété de 1992.

Figure 2 : L'application graphique beidgui (après avoir lu les données d'une carte eID de test)



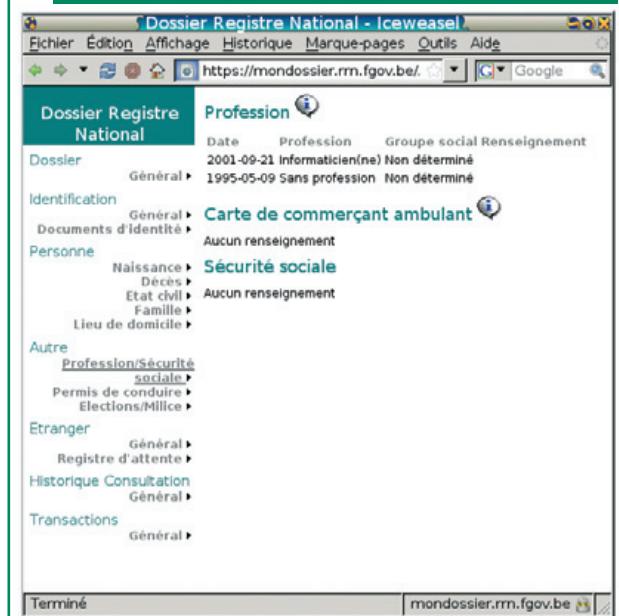
Néanmoins, pour la plupart des utilisateurs, la partie de logiciel la plus intéressante est le greffon Mozilla. Grâce à lui, on peut signer des emails, s'authentifier auprès de sites web ou même signer des documents dans OpenOffice. Son utilisation est vraiment simple : installez d'abord le greffon, puis activez-le dans le logiciel que vous souhaitez utiliser, et

enfin autorisez l'accès aux certificats. Vous serez alors en mesure d'utiliser le greffon.

Son installation dans Firefox et Thunderbird est vraiment simple. Une explication précise de la procédure est disponible sur le site web officiel des cartes eID.

Le plus connu des sites web utilisables par une carte eID est bien sûr celui du paiement en ligne des impôts belges, mais il en existe de nombreux autres. Un autre site web intéressant est celui du registre national. Véritable pierre angulaire du gouvernement électronique belge, il ne permet pas seulement de connaître ses propres données personnelles gardées par le gouvernement, il rend également possible l'impression à partir de son ordinateur personnel d'un certain nombre de documents qu'il fallait auparavant aller retirer en ville ou à la mairie.

Figure 3 : Je suis un informaticien.



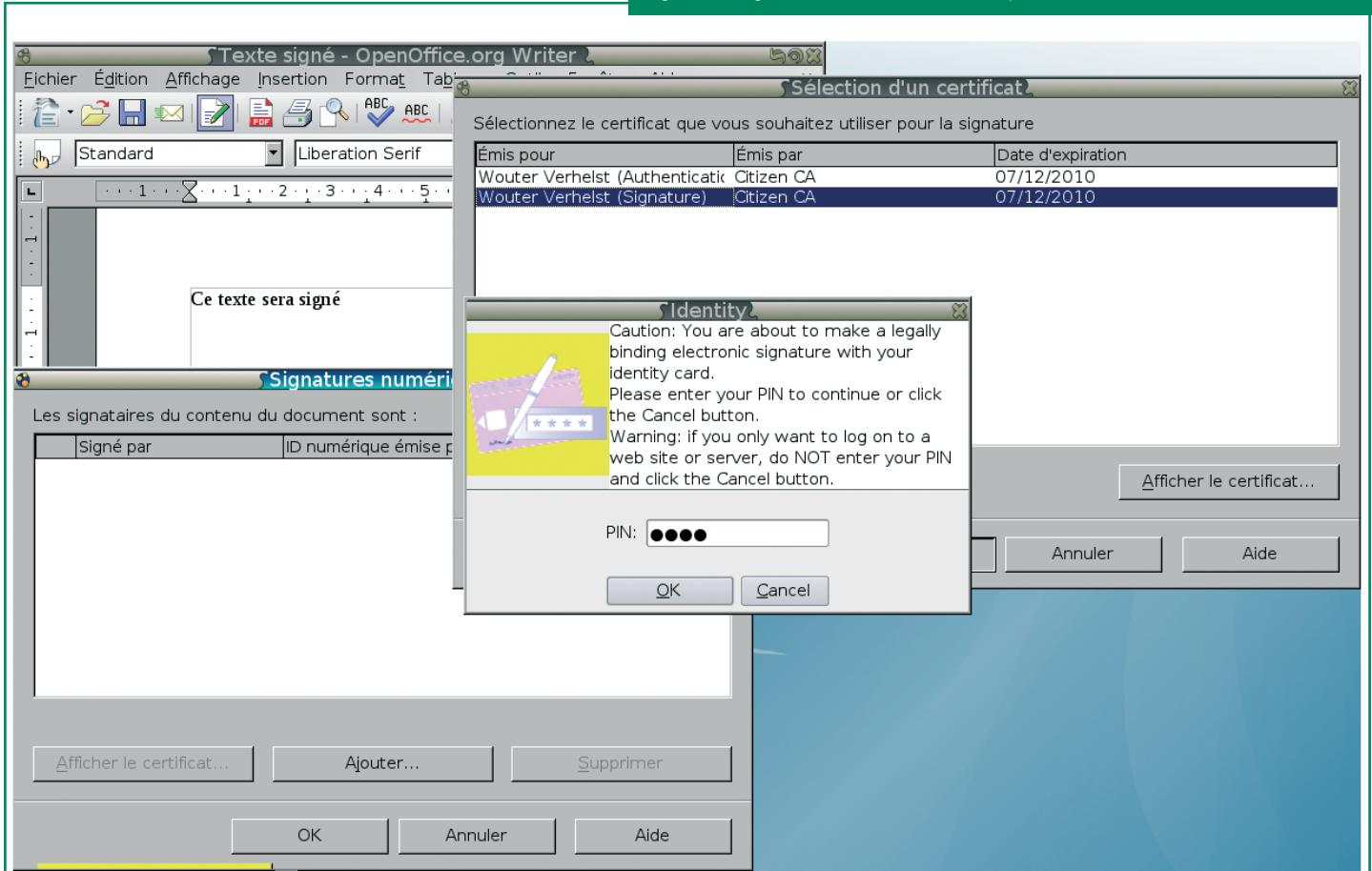
4

OpenOffice

Malheureusement, le guide d'installation n'explique pas comment permettre l'utilisation de la carte eID dans OpenOffice. Il n'est malgré tout pas très difficile d'y remédier. Premièrement, il faut activer le greffon dans Firefox. Ensuite, paramétrez la variable d'environnement **MOZILLA_CERTIFICATE_FOLDER** avec le chemin du répertoire contenant votre profil Mozilla/Firefox. Si vous démarrez maintenant OpenOffice et allez dans **Fichier --> Signatures numériques**, vous découvrirez que vous êtes capable de signer

des documents avec le bouton « Ajouter... ». Remarquez que pour signer un document, il devra avoir été au préalable enregistré au format OpenDocument, et que la signature s'appliquera à la version du document enregistrée sur le disque et non celle ouverte en mémoire. Donc, la boîte de dialogue vous montrera les deux certificats « Signature » et « Authentification » ; mais seul le certificat « signature » permettra de signer. N'utilisez donc pas l'autre.

Figure 4 : Signer des documents dans OpenOffice



5

Conclusion

Avec la carte d'identité électronique, le gouvernement belge a fourni à ses citoyens un bel objet technologique qui leur permet de ne pas seulement interagir avec leur gouvernement, mais également de signer numériquement des documents et des courriers qu'ils peuvent s'envoyer les uns les autres. Si le chiffrement n'est pas requis, et que vous avez la nationalité belge, alors la carte d'identité électronique peut vraiment combler vos besoins.

À PROPOS DE L'AUTEUR :

Wouter Verhelst est un consultant indépendant en logiciel libre. Durant son temps libre, il est aussi développeur Debian et s'occupe des paquetages de la carte d'identité électronique belge.

Cet article a été traduit par Vincent Guyot.

LIENS

- Les impôts en ligne belges : <http://www.tax-on-web.be/>
- La carte d'identité électronique belge : <http://www.eid.belgium.be/>
- Guide d'installation : http://eid.belgium.be/fr/binaries/INST_FR_LIN_tcm146-22470.pdf
- Le registre national : <https://mondossier.rnr.fgov.be/>



Le projet *pcsc-lite* est une implantation en logiciel libre de l'API WinSCard (aussi appelée PC/SC) disponible sur Windows pour interagir avec les cartes à puce.

Le projet MUSCLE ou la bibliothèque PC/SC Lite

Auteurs : Ludovic Rousseau, David Corcoran

1

Historique

Le MUSCLE (*Movement for the Use of Smart Cards in a Linux Environment*) a démarré en 1997 à Sugar Land, Texas, USA, avec David Corcoran alors étudiant de l'université de Purdue et David Sims, l'ancien directeur technique de Schlumberger (Schlumberger est devenu entre temps Axalto, puis Gemalto). Schlumberger produisait un des premiers lecteurs de carte à puce grand public appelé « Reflex 60 » et son équipe avait pour mission de développer un pilote pour Linux.

Après plusieurs semaines de recherche d'informations, David Corcoran avait produit le pilote du lecteur Reflex 60 pour Linux. Étant donné le récent mouvement du logiciel libre et l'intérêt de Linux, il était naturel de le publier en ligne. Le pilote et finalement le code source ont été publiés sur <http://www.linuxnet.com/>. Une liste de discussion a été créée pour le site et quelques documents de référence sur les cartes à puce et les lecteurs ont été ajoutés. En quelques semaines, des personnes ont commencé à soumettre des pilotes et du code source – beaucoup utilisant la spécification CT-API.

Pendant ce temps, Microsoft essayait de standardiser l'accès aux cartes à puce et aux lecteurs à travers l'architecture PC/SC. On a demandé à Corcoran quelle serait la possibilité d'avoir un équivalent de l'architecture PC/SC sur Linux et il s'est mis immédiatement au travail. Les premières versions de PC/SC pour Linux étaient très lourdes. Elles utilisaient CORBA. Puis est venue une version utilisant les RPC et, enfin, la version actuelle utilisant un mécanisme simple et multiplateforme de communication interprocessus. Cette dernière version est alors appelée **pcsc-lite**. L'API d'accès au lecteur a été standardisée et publiée afin que les fabricants de lecteurs puissent ajouter le support de leurs lecteurs à travers une API publiquement disponible. Plusieurs lecteurs devinrent rapidement utilisables avec cet intergiciel (*middleware*).

La popularité est venue rapidement pour **pcsc-lite** et le projet MUSCLE est parti en tournée : plusieurs fois à la *Linux Expo*, *Comdex* et des conférences cartes à puce en Europe. La liste de discussion a atteint les 1 200 membres et reste à cette valeur depuis.

À cette époque (année 2000), la bibliothèque permettait de porter des applications Windows sur des systèmes non-Windows, mais le comportement interne de l'API PC/SC avait des différences subtiles.

Une suite de tests a été écrite pour effectuer environ 60 à 70 tests basés sur des usages communs et obscurs de l'API. Une fois que cette suite de tests a eu dicté le comportement de PC/SC sur Linux, des versions stables ont été rendues disponibles. À partir de ce moment, plusieurs sociétés ont commencé à l'adopter.

Apple Computer, Sun Microsystems, HP, et la majorité des distributions Linux incorporent **pcsc-lite** aujourd'hui. En fait, beaucoup de systèmes embarqués non-Windows utilisent **pcsc-lite** et ses pilotes pour gérer leurs cartes à puce.

1.1 David Corcoran

David est à l'origine du projet MUSCLE. Il est l'auteur des trois versions de PC/SC pour Unix : version CORBA, RPC et version lite. La version actuelle du code est en grande partie issue de son travail.

1.2 Ludovic Rousseau

Comme dans beaucoup de projets logiciels libres, il est entré par la petite porte en tant qu'utilisateur. Il a commencé à utiliser **pcsc-lite** comme coauteur d'un pilote pour les lecteurs de la famille GemPC [**gempc**] en 2001.

2 MuscleCard

L'applet MUSCLE était une spécification *card edge* pour une applet JavaCard publiée durant l'été 2001 par ses auteurs Tommaso Cucinotta et David Corcoran. À cette époque, le langage JavaCard présentait une opportunité intéressante de fournir une application cryptographique embarquée dans une carte à puce qui fonctionnerait indépendamment du fabricant de la carte. Une fois installée, l'applet permet de faire apparaître la plupart des cartes JavaCard comme ayant les mêmes capacités cryptographiques à un intericiel externe. Le but de l'applet est simple et permet de rendre l'applet simple à comprendre et utiliser, mais aussi la rend extensible pour qu'elle puisse être étendue dans l'avenir. Les premières versions de l'applet étaient disponibles sous forme de fichiers *class* (standard Java) et CAP (*Converted APplet* standard JavaCard) pendant la création initiale. Les versions suivantes incluent le code source de l'applet pour qu'elle

Il a ensuite soumis des correctifs et améliorations au logiciel **pcsc-lite** maintenu à l'époque par David Corcoran. Il a rapidement obtenu le droit d'accès au dépôt CVS et a fait quelques *releases* de versions bêta incluant ses modifications. La dernière version de **pcsc-lite** par David en tant que *release manager* date de novembre 2002. Depuis 2003 (**pcsc-lite** version 1.1.2beta4), Ludovic a pris la tête du développement, de la maintenance et a assumé le rôle de *release manager*. Dans le même temps, il proposait des améliorations au paquet Debian de **pcsc-lite** et en est devenu le mainteneur officiel en juin 2002.

Le dépôt CVS était hébergé sur une machine personnelle de David aux États-Unis d'Amérique. Ce n'était pas très rapide pour un accès en SSH depuis la France. Il a alors créé le projet pcsclite [**pcsclite**] sur le site d'hébergement Alioth de Debian en avril 2003. Il a ensuite migré la base de CVS à subversion en août 2005.

1.3 Damien Sauveron

Durant sa thèse de doctorat (soutenue en 2004), Damien a réalisé un environnement de calcul distribué sur carte à puce. L'idée est de répartir les calculs sur plusieurs (dizaines de) cartes à puce. Il a alors beaucoup participé au développement de **pcsc-lite** pour supporter plus de 10 lecteurs et pour permettre d'avoir des accès concurrents sur deux (ou plus) cartes à la fois.

puisse être compilée et installée sur une grande variété de plateformes carte. L'applet a une structure simple incluant un gestionnaire mémoire d'objets permettant la création, la destruction et la modification d'objets sur la carte. Elle fournit également des capacités cryptographiques de génération de clé RSA, signature, déchiffrement et importation de clé. Elle a des règles de contrôle d'accès basiques basées sur les objets : propriété de lecture, écriture, effacement ou utilisation qui pouvait être : ALWAYS (toujours), PIN (après vérification d'un code PIN), EXTERNAL AUTH (après authentification cryptographique) ou NEVER (jamais).

L'applet a ensuite été étendue ces dernières années par Karsten Ohme pour compléter ses capacités cryptographiques par rapport au design originel. L'applet fonctionne sur la plupart des cartes à puce basées sur JavaCard 2.1 ou versions plus récentes.

3 Description de l'API WinSCard

Cette liste de fonctions reprend la classification de l'API PC/SC telle que décrite par Microsoft sur le MSDN [**msdn**]. Les fonctions grisées ne sont pas supportées par **pcsc-lite**, parce qu'elles ne sont jamais utilisées (ou presque) et que personne n'a eu besoin de les planter.

3.1 Interrogation de la base de données carte à puce

Fonction	Description
SCardGetProviderId	Rechercher l'identificateur (GUID) du fournisseur de service primaire pour la carte donnée.
SCardListCards	Rechercher une liste de cartes précédemment présentées au système par un utilisateur spécifique.
SCardListInterfaces	Rechercher les identificateurs (GUID) des interfaces fournies par une carte donnée.
SCardListReaderGroups	Rechercher une liste de groupes de lecteurs qui ont été précédemment présentés au système.
SCardListReaders	Rechercher la liste de lecteurs dans un ensemble de groupes nommés de lecteurs.

La fonction **SCardListReaderGroups()** retourne toujours la chaîne de caractères **SCard\$DefaultReaders**. La fonction **SCardListReaders()** ne tient pas compte du paramètre **mszGroups** donnant le nom du groupe et retourne donc tous les lecteurs connectés sur la machine.

3.2

Gestion de la base de données carte à puce

Fonction	Description
SCardAddReaderToGroup	Ajouter un lecteur à un groupe de lecteurs.
SCardForgetCardType	Retirer une carte à puce du système.
SCardForgetReader	Retirer un lecteur du système.
SCardForgetReaderGroup	Retirer un groupe de lecteurs du système.
SCardIntroduceCardType	Présenter une nouvelle carte au système.
SCardIntroduceReader	Présenter un nouveau lecteur au système.
SCardIntroduceReaderGroup	Présenter un nouveau groupe de lecteurs au système.
SCardRemoveReaderFromGroup	Retirer un lecteur d'un groupe de lecteurs.

Aucune des fonctions de ce groupe n'est supportée par **pcsc-lite**.

3.3

Contextes du gestionnaire de ressources

Fonction	Description
SCardEstablishContext	Établir un contexte PC/SC pour accéder aux autres fonctions de l'API.
SCardReleaseContext	Libérer un contexte établi.

3.4

Support du gestionnaire de ressources

Fonction	Description
SCardFreeMemory	Libérer la mémoire retournée par l'utilisation de SCARD_AUTOALLOCATE .

3.5

Tracking de carte à puce

Fonction	Description
SCardLocateCards	Rechercher une carte dont la chaîne de caractères d'ATR correspond à un nom de carte fourni.
SCardGetStatusChange	Bloquer l'exécution jusqu'à ce que l'état actuel des cartes change.
SCardCancel	Terminer l'action (bloquante) en cours.

La fonction **SCardLocateCards()** utilise la base de registre sous Windows. Il est difficile d'écrire une fonction équivalente sous Unix.

3.6

Accès aux lecteurs et cartes à puce

Fonction	Description
SCardConnect	Se connecter à une carte.
SCardReconnect	Rétablissement une connexion à une carte.
SCardDisconnect	Terminer une connexion à une carte.
SCardBeginTransaction	Commencer une transaction, bloquant l'accès de la carte aux autres applications.
SCardEndTransaction	Terminer une transaction, en permettant à d'autres applications d'accéder à la carte.
SCardStatus	Fournir l'état actuel du lecteur.
SCardTransmit	Transmettre un APDU et récupérer le résultat.

Les fonctions de ce groupe sont les principales. En particulier, **SCardTransmit()** est utilisée pour tous les échanges entre l'application et la carte à puce.

3.7 Accès directs

Fonction	Description
SCardControl	Envoyer une commande directe du lecteur.
SCardGetAttrib	Obtenir un attribut du lecteur.
SCardSetAttrib	Modifier un attribut du lecteur.

La fonction **SCardControl()** permet de communiquer avec le lecteur plutôt qu'avec la carte. Cette fonction est utilisée par exemple pour demander aux lecteurs de classe 2 (lecteur avec un clavier et éventuellement un écran) de faire une vérification de PIN en mode sécurisé (le PIN est alors envoyé à la carte par le lecteur sans passer par l'ordinateur).

3.8 Exemple d'utilisation

Cet exemple est très simple et ne fait aucune gestion de code d'erreur. Il utilise le premier lecteur pour envoyer une commande ISO 7816-4 de sélection de fichier (0xA4 = SELECT FILE) sur le fichier racine 0x3F00.

```
#include <stdio.h>
#include <stdlib.h>

#ifndef __APPLE__
#include <PCSC/winscard.h>
#include <PCSC/wintypes.h> /* for Windows types DWORD, LONG, etc. */
#else
#include <winscard.h>
#endif

int main(void)
{
    LONG rv;
    SCARDCONTEXT hContext;
```

```
LPSTR mszReaders;
DWORD dwReaders, dummy;
SCARDHANDLE hCard;
DWORD dwActiveProtocol, dwSendLength, dwRecvLength, dwAtrLen;
SCARD_IO_REQUEST pioRecvPci;
BYTE pbRecvBuffer[10];
BYTE pbSendBuffer[] = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
BYTE pbAtr[MAX_ATR_SIZE];
int i;

rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
rv = SCardListReaders(hContext, NULL, NULL, &dwReaders);
mszReaders = malloc(sizeof(char)*dwReaders);
rv = SCardListReaders(hContext, NULL, mszReaders, &dwReaders);
printf("reader name: %s\n", mszReaders);
rv = SCardConnect(hContext, mszReaders, SCARD_SHARE_SHARED,
    SCARD_PROTOCOL_T0, &hCard, &dwActiveProtocol);
dwAtrLen = sizeof(pbAtr);
rv = SCardStatus(hCard, NULL, &dummy, NULL, NULL, pbAtr, &dwAtrLen);
printf("ATR: ");
for(i=0; i<dwAtrLen; i++)
    printf("%02X ", pbAtr[i]);
printf("\n");
dwSendLength = sizeof(pbSendBuffer);
dwRecvLength = sizeof(pbRecvBuffer);
printf("sending: ");
for(i=0; i<dwSendLength; i++)
    printf("%02X ", pbSendBuffer[i]);
printf("\n");
rv = SCardTransmit(hCard, SCARD_PCI_T0, pbSendBuffer, dwSendLength,
    &pioRecvPci, pbRecvBuffer, &dwRecvLength);
printf("response: ");
for(i=0; i<dwRecvLength; i++)
    printf("%02X ", pbRecvBuffer[i]);
printf("\n");
rv = SCardDisconnect(hCard, SCARD_UNPOWER_CARD);
rv = SCardReleaseContext(hContext);
return 0;
}
```

Pour compiler l'exemple, il faut utiliser **gcc -framework PCSC sample.c** sur Mac OS X et **gcc \$(pkg-config --cflags --libs libpcsc-lite) sample.c** sur GNU/Linux.

4 Portabilité

L'intericiel **pcsc-lite** a été conçu pour fonctionner sur n'importe quel système d'exploitation de type Unix. GNU/Linux est la plateforme de développement actuelle, mais **pcsc-lite** a aussi une version maintenue par Sun pour Solaris et une version maintenue par Apple pour Mac OS X.

4.1 Solaris

La version de Sun pour Solaris est différente de la version officielle pour supporter la fonctionnalité de Zones du système d'exploitation et pour supporter les clients légers Sun Ray (tous équipés de lecteurs de cartes à puce pour l'authentification de l'utilisateur).

Sun est parti de la version 1.3.2 (août 2006) pour ces modifications. Elles ne sont donc pas directement intégrables dans la version courante de **pcsc-lite** (1.4.101 actuellement). Une branche **branches/Solaris/** a été créée sur le dépôt subversion pour accueillir le code que Sun vient de rendre public en juin 2008.

Une intégration, dans la version officielle, des modifications de Sun serait souhaitable pour tout le monde, mais c'est un gros travail qui peut prendre un peu de temps.

La version officielle de **pcsc-lite** fonctionne correctement sur Solaris. Elle peut donc suffire dans le cas (simple) d'une station de travail avec un lecteur de carte connecté localement.

4.2 Mac OS X

La version pour Mac OS X est différente de la version officielle pour supporter Rosetta, l'émulateur permettant de faire tourner des applications PowerPC sur un Mac Intel, pour supporter le mécanisme de mise en veille du système et le mécanisme de hotplug de lecteurs USB utilisant l'IOKit (API spécifique à Mac OS X), plus d'autres modifications mineures.

Dans le cas de Rosetta, le démon pcscd est en code natif Intel (petit boutien, *little endian*) lancé par le système alors que la

bibliothèque **libpcslite** est en code PowerPC (grand boutien, *big endian*) puisque liée à l'application PowerPC. Il faut donc faire des conversions entre petit et grand boutien dans la communication entre le démon **pcscd** et la bibliothèque **libpcslite**.

Apple inclut **pcsc-lite** depuis Mac OS X 10.2 (Jaguar, août 2002). Ils sont partis de la version 1.1.1 disponible à l'époque et l'ont modifiée pour leurs besoins. Apple n'a depuis jamais essayé de faire intégrer leurs modifications (même les plus

minimes) dans la version officielle. Entre Tiger (10.4) et Leopard (10.5), ils ont énormément fait évoluer le code dans une direction qui leur est propre. Il va être très difficile de faire converger la branche Apple avec la branche officielle. Est-ce même la volonté d'Apple ?

Apple ayant sa propre branche (ils ont *forké* le code), ils ont aussi leurs propres bugs. La page [\[sca\]](#) liste une dizaine de gros bugs du **pcsc-lite** d'Apple sur Leopard (version 10.5.4).

5

Différences avec WinSCard de Microsoft

Le but de **pcsc-lite** est de fournir une API identique à l'API WinSCard de Microsoft. Il ne devrait donc pas y avoir de différences. Il en existe cependant quelques-unes.

5.1

Sémantique différente

SCardEstablishContext() est la première fonction appelée et permet d'obtenir un contexte PC/SC qui sera utilisé par les autres fonctions de l'API. Cette fonction utilise un paramètre **dwScope** qui peut prendre les valeurs **SCARD_SCOPE_USER**, **SCARD_SCOPE_TERMINAL**, **SCARD_SCOPE_GLOBAL** ou **SCARD_SCOPE_SYSTEM**. Seule la valeur **SCARD_SCOPE_SYSTEM** est supportée par **pcsc-lite**.

SCardStatus() donne l'état de la carte dans un lecteur donné. Les valeurs possibles du paramètre **pdwState** sont **SCARD_ABSENT**, **SCARD_PRESENT**, **SCARD_SWALLOWED**, **SCARD_POWERED**, **SCARD_NEGOTIABLE** et **SCARD_SPECIFIC**. Sur Windows, le paramètre **pdwState** prend l'une de ces valeurs. Alors qu'avec **pcsc-lite** le paramètre est une combinaison binaire de ces valeurs (un champ de bits).

5.2

Fonctions manquantes

L'API WinSCard définit aussi des fonctions permettant de renommer localement un lecteur avec **SCardIntroduceReader()**, de faire un lien entre un ATR de carte et un service avec **SCardIntroduceCardType()** ou de trouver dans quel lecteur se trouve une carte d'un nom donné avec **SCardLocateCards()**. Ces fonctions sont liées à la façon dont les cartes sont enregistrées dans le système (base de registre Windows).

Les fonctions manquantes n'existent pas dans **pcsc-lite** simplement par manque d'intérêt. Si quelqu'un a besoin d'une fonction non présente, le plus simple est de soumettre un patch sur la liste de développement de **pcsc-lite** [\[liste muscle\]](#).

5.3

Fonction supplémentaire

La fonction **pcsc_stringify_error()** permet de récupérer la version texte d'un code d'erreur PC/SC. Cela évite que chaque application PC/SC n'ait sa propre table de conversion. Le message « *No smart card inserted* » est quand même plus parlant que la valeur **0x8010000C**.

6

Pilotes de lecteur

pcsc-lite est un intericiel (*middleware*) qui ne sait pas parler à un lecteur de carte à puce. Pour cela, il faut utiliser un pilote spécifique au lecteur utilisé. L'interface de programmation entre **pcsc-lite** et le pilote s'appelle **ifdhandler** [\[ifdhandler\]](#). Elle en est à la version 3.

6.1

Pilotes CCID

La grande majorité des lecteurs de carte à puce sont maintenant des lecteurs connectés sur le bus USB et remplacent les lecteurs utilisant un port série. Les principaux fabricants de lecteurs ont même travaillé ensemble pour définir une interface de programmation commune. Ainsi, tout lecteur de carte à puce respectant la spécification CCID (*Integrated*

Circuit(s) Cards Interface Devices) peut être utilisé avec le même pilote générique. Il n'est plus nécessaire d'avoir un pilote spécifique par lecteur.

La norme CCID ne concerne pas uniquement des lecteurs USB. Un lecteur ExpressCard (remplaçant du PCMCIA) permet de se connecter sur le bus USB et donc d'être vu comme un lecteur USB normal. Il existe aussi un lecteur (GemPC Twin serial) se branchant sur le port série et utilisant le jeu de commandes CCID. Mais, dans ce cas, la communication utilise un protocole ad hoc qui n'est pas l'USB.

6.1.1

libccid

En 2003, Ludovic Rousseau a commencé l'écriture d'un pilote pour **pcsc-lite** implantant la spécification CCID. La

plateforme cible principale est GNU/Linux, mais avec une contrainte de portabilité maximale. Il a alors été choisi d'utiliser la bibliothèque **libusb** pour abstraire la couche USB de la plateforme. La bibliothèque **libusb** étant disponible pour Linux, FreeBSD, NetBSD, OpenBSD, Darwin et Mac OS X, le pilote est en théorie utilisable sur chacune de ces plateformes. En pratique, le pilote est disponible dans les principales distributions GNU/Linux, dans FreeBSD (*ports/devel/libccid/*), ainsi que dans NetBSD (*security/ccid*).

Ce pilote [**pilote CCID**] est disponible en logiciel libre (licence LGPL v2). Il est utilisé officiellement par quelques fabricants de lecteurs de carte à puce (Gemalto par exemple) pour GNU/Linux et/ou Mac OS X.

NOTE

Avant d'acheter un lecteur, pensez à vérifier qu'il est dans la liste des pilotes supportés. Certains lecteurs CCID sont bogus et posent problème. Évitez-les.

6.1.2 ifd-CCID

À la même époque (2003), Jean-Luc Giraud (coauteur avec Ludovic Rousseau du pilote pour lecteurs GemPC 410 et 430) a également commencé à écrire un pilote identique, mais pour Mac OS X [**ifd-CCID**]. Jean-Luc a suivi les recommandations d'Apple et a utilisé la bibliothèque IOKit pour l'interaction avec la couche USB. Il a également utilisé la licence BSD pour complaire aux juristes d'Apple.

7

Aspects sécurité

PC/SC est juste une API de communication bas niveau. Les aspects authentification et intégrité de la communication entre le PC et la carte à puce ne sont pas pris en compte. La situation est assez similaire à l'interface de communication réseau (type *socket Unix*). N'importe quel programme peut établir une communication avec un service. La sécurité est mise en place par une couche supplémentaire, en général TLS (*Transport Layer Security*). Dans le cas des cartes à puce, la couche de sécurisation utilisée en général s'appelle *secure messaging* et ne sera pas décrite dans cet article.

Par défaut, n'importe quel programme peut utiliser l'API PC/SC et établir une connexion avec une carte à puce présente dans un lecteur. Le problème est alors qu'un programme peut envoyer une commande à la carte alors que la carte est déjà utilisée par un autre programme. La carte ne sait pas identifier et différencier les différents programmes et va mélanger les commandes.

PC/SC propose plusieurs mécanismes pour éviter ces problèmes : un accès exclusif à la carte et les transactions.

Ce pilote a été intégré par Apple dans Mac OS X 10.4 (en 2005). Il n'a pas évolué depuis. Le pilote souffre de limitations et peut avantageusement être remplacé par le pilote **libccid**.

6.1.3 Pilote ccid d'OpenCT

Le projet OpenCT (que nous verrons plus loin) propose aussi un pilote générique CCID. Ce pilote n'offre pas plus de fonctionnalités que **libccid** et n'est utilisé qu'avec OpenCT, et donc pas avec l'API PC/SC.

6.1.4 Autres

Certains fabricants de lecteurs fournissent leur propre pilote, en logiciel propriétaire (*proprietary*) pour tous les cas que je connais. L'avantage d'un tel pilote peut être le support d'une fonction non CCID du lecteur. C'est le cas par exemple de l'interface sans contact de certains lecteurs de la société SCM Micro.

6.2 Pilotes non CCID

Le site MUSCLE répertorie une liste de pilotes [**muscle drivers**]. Ces pilotes sont en général assez vieux et pas forcément très bien maintenus.

7.1 Accès exclusif

La fonction **SCardConnect()** de connexion à une carte permet de spécifier le mode de partage avec les autres applications. Les valeurs possibles du paramètre **dwShareMode** sont **SCARD_SHARE_SHARED**, **SCARD_SHARE_EXCLUSIVE** et **SCARD_SHARE_DIRECT**.

La valeur **SCARD_SHARE_DIRECT** permet de se connecter au lecteur plutôt qu'à la carte. Cela est utilisé pour récupérer des attributs du lecteur ou du pilote du lecteur par exemple.

La valeur **SCARD_SHARE_SHARED** laisse aux autres applications la possibilité de se connecter à la carte. C'est la valeur utilisée en général.

La valeur **SCARD_SHARE_EXCLUSIVE** garantit que l'application sera alors la seule pouvant communiquer avec la carte. Les autres applications recevront le code d'erreur **SCARD_E_SHARING_VIOLATION**. Cette allocation exclusive de la ressource peut être pénalisante si l'application garde

l'exclusivité pendant une très longue durée. PC/SC fournit alors un mécanisme d'exclusivité avec une granularité plus fine : la transaction.

7.2 Transaction

Une transaction PC/SC permet d'obtenir un accès exclusif à la carte sans avoir besoin de se connecter et déconnecter en mode exclusif. Les fonctions PC/SC sont **SCardBeginTransaction()** et **SCardEndTransaction()**.

Une séquence classique utilisant une transaction PC/SC est la suivante :

1. **SCardBeginTransaction()** ;
2. présentation du code PIN à la carte pour ouvrir l'accès aux fichiers protégés ;
3. déplacement dans le système de fichiers de la carte jusqu'au fichier désiré ;
4. lecture et/ou écriture du fichier ;
5. invalidation du code PIN dans la carte pour éviter qu'une autre application ne bénéficie des accès privilégiés ;
6. **SCardEndTransaction()**.

En suivant scrupuleusement cette séquence, aucune autre application (qui ne connaît pas le code PIN secret) ne peut accéder au contenu protégé de la carte.

La fonction **SCardEndTransaction()** possède le paramètre **dwDisposition** qui permet de spécifier s'il faut laisser la carte à puce en l'état (**SCARD_LEAVE_CARD**), forcer un reset de la carte (**SCARD_RESET_CARD**) ou mettre la carte hors tension (**SCARD_UNPOWER_CARD**). La différence entre **SCARD_RESET_CARD** et **SCARD_UNPOWER_CARD** est que, dans le premier cas, l'alimentation de la carte est maintenue. Le contenu de la RAM

de la carte n'est donc pas perdu. Ce reset est appelé « reset à chaud ». Dans le deuxième cas, l'alimentation est coupée et le contenu de la RAM de la carte est alors perdu. Ce reset est appelé « reset à froid ».

7.3

Limitation par le système d'exploitation

Il existe des environnements dans lesquels les deux précédentes restrictions ne sont pas suffisantes (par exemple sur un système militaire multi-utilisateur). Il est alors possible d'utiliser les mécanismes de sécurité du système d'exploitation pour limiter l'utilisation de PC/SC aux utilisateurs d'un groupe Unix particulier.

Ce mécanisme est utilisable grâce à l'architecture interne de **pcsc-lite**, qui utilise une architecture client-serveur. La partie client est constituée d'une bibliothèque (**libpcslite**) utilisée par le programme utilisant l'API PC/SC. La partie serveur est constituée d'un démon (**pcscd**) qui gère les lecteurs de carte et assure le partage des ressources (en imposant les accès exclusifs par exemple).

Les lecteurs de carte ne sont utilisables que par un processus privilégié (**pcscd** fonctionne en tant qu'utilisateur *root*) donc le serveur **pcscd** est le point de passage obligé pour les applications PC/SC. La communication entre **libpcslite** et **pcscd** utilise une *socket* du domaine Unix (donc locale à la machine). Il est alors possible de spécifier les droits d'accès Unix sur cette socket (qui est un fichier sur disque) pour que seuls les utilisateurs d'un groupe particulier, par exemple le groupe **smartcard**, puissent accéder à la socket et donc communiquer avec le serveur **pcscd**.

Dans une installation standard de **pcsc-lite**, ce contrôle d'accès n'est pas utilisé et n'importe quel utilisateur peut utiliser PC/SC. L'administrateur du système peut restreindre les accès si nécessaire.

Le mécanisme décrit ici est purement Unix et n'est pas fourni par PC/SC. Microsoft fournit peut-être quelque chose de similaire.

8

API concurrentes : OpenCT et CT-API

OpenCT et CT-API sont deux API concurrentes à PC/SC. Elles sont également utilisées directement par une application, mais fournissent un niveau de services différent.

8.1 OpenCT

D'après Andreas Jellinghaus, coauteur d'OpenCT : « OpenCT a été écrit à une époque où écrire un pilote pour **pcsc-lite** était difficile, **pcsc-lite** se comportait de façon amusante (avec tout le monde trop occupé pour aider à trouver pourquoi), et quelques centaines de lignes de code ont résolu le problème (en remplaçant **pcsc-lite**). OpenCT est toujours disponible, parce qu'il fonctionne assez bien dans de nombreux cas et

est facile à configurer et à débuguer. » La première version d'OpenCT (version 0.1.0) est sortie en août 2003.

Contrairement à PC/SC, OpenCT propose de lire et d'écrire sur des cartes synchrones (cartes à mémoire, type carte de cabine téléphonique). Il faut cependant que le lecteur et son pilote supportent cette possibilité.

OpenCT propose des pilotes pour des lecteurs n'ayant pas de pilote pour **pcsc-lite**. Cela devrait être la seule raison d'utiliser OpenCT. Toujours d'après Andreas : « En termes de features et de pilotes supportés, **pcsc-lite** et ccid sont la voie à suivre de nos jours. Et avec Ludovic mainteneur actif des deux projets, alors que personne ne travaille sur OpenCT, il est préférable d'investir dans **pcsc-lite**. »

8.2 CT-API

CT-API (*Card Terminal Application Programming Interface*) est encore plus ancien qu'OpenCT. La documentation de CT-API 1.1 date de 1996, donc bien avant la création de **pcsc-lite**.

Comme OpenCT, cette API permet d'utiliser des cartes synchrones et des cartes asynchrones. L'API (très minimaliste) ne contient que 3 fonctions **CT_init()**, **CT_data()** et **CT_close()**.

L'API est très bas niveau pour la gestion des lecteurs. Par exemple, il faut que l'application sache sur quel port série est branché le lecteur de carte. Il n'y a pas de fonction donnant la liste des lecteurs connectés.

Il existe assez peu d'applications utilisant cette API. Il faut noter qu'OpenCT propose une interface CT-API. Il est donc possible d'utiliser une application utilisant CT-API avec des lecteurs modernes (CCID) en passant par OpenCT.

CT-API n'est pas spécifique à Linux. Certains fabricants de lecteurs proposent un pilote (logiciel propriétaire) fourni par cette API sur Windows, en plus de fournir un pilote pour PC/SC.

9 Historique de pcsc-lite

9.1 Passé et présent

L'outil statsvn a été utilisé pour générer quelques graphiques à partir des données du dépôt Subversion (converti depuis un dépôt CVS en août 2005).

Pour **pcsc-lite**, nous avons le découpage en contributions suivant : (voir ci-contre).

Depuis avril 2003, le projet est hébergé sur le serveur communautaire Alioth de Debian. Les noms en ***-guest** sont utilisés par ceux qui ne sont pas développeurs Debian officiels. L'historique inclut aussi la période avant 2003 pendant laquelle le code était sur la machine personnelle de David Corcoran avec des noms sans suffixe **-guest**.

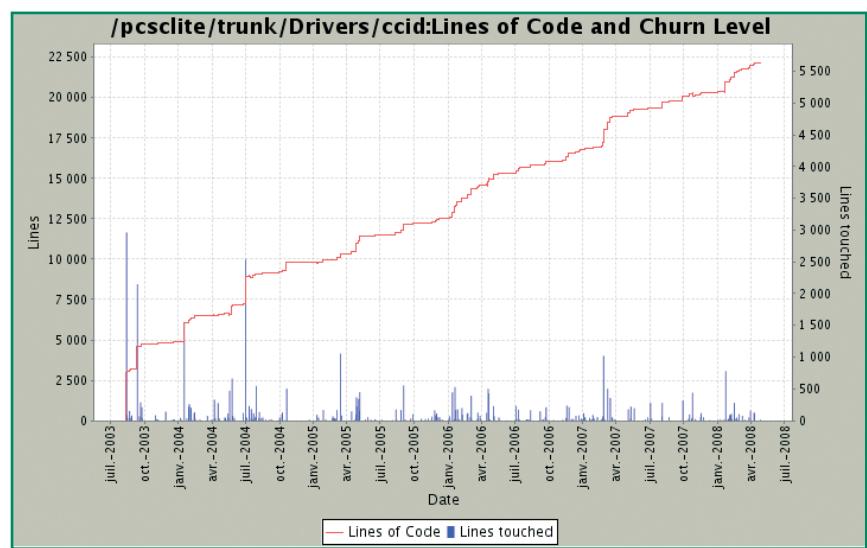
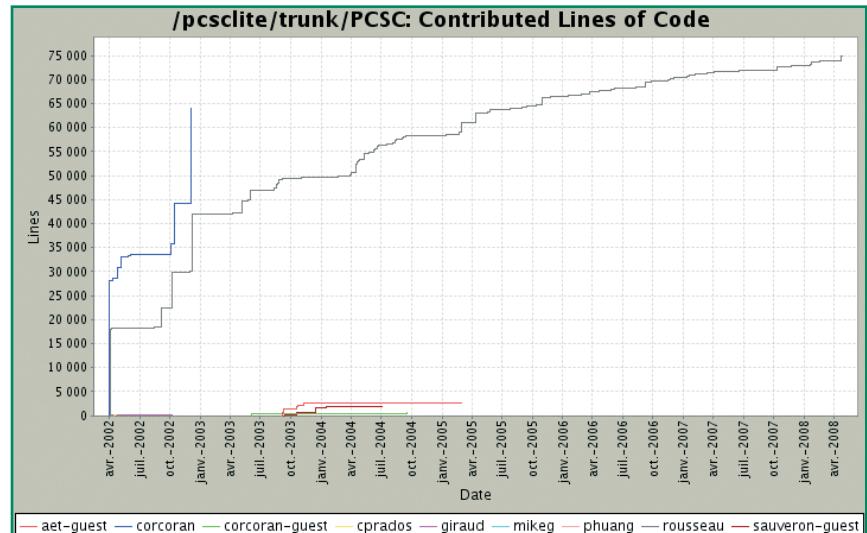
Jusqu'en 2003, David Corcoran est le principal contributeur. En 2004, Damien Sauveron améliore le support multi-lecteur. Antti Tapaninen (aet) a amélioré la qualité générale et la portabilité du code. Ludovic Rousseau est le contributeur le plus régulier.

Pour **libccid**, nous avons un seul contributeur : (voir ci-contre).

Même si Ludovic Rousseau est le seul à modifier le code, certaines modifications ou corrections de bug viennent d'utilisateurs.

Pour les deux projets, on remarque une évolution régulière qui n'a pas vraiment ralenti.

Si l'API de **pcsc-lite** ne devrait pas (beaucoup) évoluer, l'intégration de **pcsc-lite** dans le système évolue (utilisation récente de **libhal** pour détecter les branchements/débranchements de lecteurs par exemple) et évoluera encore.



9.2 Futur proche

9.2.1 Exécution du démon en tant que simple utilisateur

Pour l'instant, le démon pcscd s'exécute en tant que root pour pouvoir accéder aux lecteurs de cartes. À l'origine, les lecteurs étaient connectés sur un port série et les droits d'accès par défaut étaient restreints au super-utilisateur. pcscd devait donc s'exécuter avec les priviléges de root pour utiliser le(s) lecteur(s) de carte.

De nos jours, la plupart des lecteurs de carte utilisent un port USB. Par défaut, les droits d'accès aux périphériques USB sont restreints au super-utilisateur, le problème reste donc le même et pcscd doit s'exécuter avec les priviléges de root.

Le développement de l'infrastructure udev sur Linux permet d'écrire des règles afin que les droits d'accès à un périphérique soient spécifiés lors du branchement de ce périphérique. Il est donc possible d'avoir un contrôle plus fin des droits d'accès pour qu'un programme non-root puisse accéder aux lecteurs de carte à puce, mais que les utilisateurs normaux du système ne puissent toujours pas accéder directement aux lecteurs.

9.2.2 Démarrer le démon à la demande

Une autre amélioration de l'infrastructure de *hotplug* de Linux est HAL (*Hardware Abstraction Layer*). Avec ce mécanisme, il devrait être possible de ne démarrer pcscd que lorsqu'un lecteur de carte est connecté.

Mac OS X utilise déjà un mécanisme similaire en utilisant l'infrastructure launchd qui lance securityd qui lance pcscd uniquement lorsqu'un lecteur de carte est effectivement connecté à la machine.

9.3 Futur plus lointain

9.3.1 Gestion de la mise en veille

Pour l'instant, **pcsc-lite** ne semble pas gérer correctement la mise en veille et l'hibernation du système. Lors du réveil du système, il faudrait que chaque pilote reconfigure le(s) lecteur(s) dont il avait la charge avant la mise en veille.

Une contribution est la bienvenue, le développeur principal de **pcsc-lite** manquant de motivation personnelle, car n'utilisant plus d'ordinateur portable.

9.3.2 PC/SC version 2

La version 2 de la spécification PC/SC étend la spécification pour couvrir d'autres familles de cartes à puce.

9.3.2.1 Cartes sans contact

Le support des cartes sans contact (ISO 14443) n'est pas prévu dans PC/SC v1. Certains lecteurs et pilotes s'arrangent pour que la carte soit visible comme si c'était une carte à

contact. Ce n'est cependant pas toujours possible et certaines possibilités offertes par les cartes et lecteurs sans contact ne sont pas utilisables.

PC/SC version 2 (partie 3) définit des mécanismes à utiliser par l'application pour utiliser une carte sans contact avec un lecteur sans contact. Il s'agit principalement d'aspects de sécurité comme charger un jeu de clés cryptographiques dans le lecteur.

Les commandes pour le lecteur sont passées par l'appel **SCardTransmit()** dans de faux APDU que le lecteur doit intercepter et ne pas transmettre à la carte. C'est assez étrange, puisque l'appel **SCardControl()** permet de dialoguer avec le lecteur directement. Encore une spécification coécrite par Microsoft :-).

9.3.2.2 Cartes synchrones

Les cartes synchrones, aussi appelées « cartes à mémoire », sont des cartes avec une puce très limitée. Typiquement, c'est la carte téléphonique française (utilisée dans les télécabines).

Il n'y a encore jamais eu de requêtes pour supporter ce type de carte au niveau de **pcsc-lite**. De plus, la norme CCID ne définit rien pour les cartes synchrones. Il faut donc utiliser des commandes propriétaires pour les lecteurs CCID qui peuvent utiliser ces cartes.

9.3.2.3 Cartes multi-applicatives

Windows identifie une carte en fonction de son ATR (*Answer To Reset*). C'est avec l'ATR qu'une carte est entrée dans la base (de registre) avec **SCardIntroduceCardType()**. Le problème est que, de nos jours, une carte à puce est une plateforme (JavaCard, .NET ou même MULTOS) sur laquelle il est possible de charger plusieurs applications. Il n'est donc plus possible d'identifier une application à partir d'un ATR, une même carte embarquant plusieurs applications.

PC/SC version 2 définit un composant supplémentaire : *ICC Service Provider* (*ICC = Integrated Circuit Card* ou carte à puce). Ce composant se place au-dessus du *Resource Manager* (l'API WinSCard). **pcsc-lite** ne fournit déjà pas les services de gestion de la base de donnée carte à puce. Ce composant de PC/SC version 2 supplémentaire sera hypothétiquement disponible un jour dans **pcsc-lite**.

9.3.2.4 Pilotes avec capacités étendues

PC/SC version 2 partie 10 définit une interface pour utiliser les lecteurs avec un clavier intégré (pinpad). Cette fonctionnalité est déjà supportée par le pilote **libccid**.

Par contre, il n'est pas possible d'avoir un contrôle fin du clavier ou de l'écran. PC/SC version 2 définit un autre module appelé *IFD Service Provider* (*IFD = Interface Device* ou lecteur de carte) au-dessus du Resource Manager (l'API WinSCard) pour répondre à ces besoins.

Comme il est déjà possible de passer des commandes au lecteur avec **SCardControl()**, l'intérêt d'avoir un composant logiciel (en C++) en plus est tout relatif.

9.3.3**Support de Windows**

Il peut sembler aberrant de faire un portage de **pcsc-lite** sur Windows, puisque **pcsc-lite** est censé copier le fonctionnement de la bibliothèque **WinSCard.dll** de Windows. En fait, il existe des cas d'utilisation pour lesquels **pcsc-lite** est plus performant que la version fournie par Microsoft. En particulier, pour le support de plusieurs lecteurs simultanément,

la version de Microsoft semble limitée et pose des problèmes de stabilité et de performances.

Utiliser plus de 2 ou 3 lecteurs sur une même machine n'est pas une configuration courante, mais c'est une configuration classique sur une machine de personnalisation de carte qui peut alors configurer plusieurs cartes en parallèle.

Cette idée n'est restée qu'à l'état de discussion par manque de motivation et de moyens, mais pourrait être mise en œuvre si nécessaire.

10**Conclusion**

Le projet MUSCLE a maintenant plus de 10 ans. Il a produit le composant logiciel **pcsc-lite** et des pilotes de lecteur de carte. **pcsc-lite** est quasiment incontournable pour utiliser des cartes à puce sur des systèmes d'exploitations autres que Windows. Il permet surtout de fournir l'API WinSCard déjà disponible sur Windows et donc d'avoir le même code source applicatif sur toutes les plateformes.

Les problèmes de portabilités entre plateformes ne sont pas résolus pour autant, puisque les interfaces de plus haut niveau utilisant la carte à puce pour fournir des services cryptographiques ne sont pas les mêmes sur les 3 principales plateformes. Microsoft utilise l'API (propriétaire) CAPI (*Cryptographic Application Programming Interface*), Apple utilise CDSA (*Common Data Security Architecture* produit par l'Open Group) et Mozilla et autres applications non liées à une plateforme utilisent PKCS#11 (*Public Key Cryptographic Standards* partie 11, standard produit par la société RSA Labs).

La situation n'est pas désespérée, puisque le projet CoolKey [**coolkey**] de Red Hat permet de fournir un module CSP pour

CAPI sur Windows, un module TokenD pour CDSA sur Mac OS X et un module PKCS#11 pour le reste des applications. CoolKey utilise d'ailleurs l'applet MuscleCard du projet MUSCLE.

AUTEURS : LUDOVIC ROUSSEAU, DAVID CORCORAN

Ludovic Rousseau - Utilisateur de GNU/Linux depuis 1994.

Développeur Debian depuis 2001



David Corcoran a commencé le projet libre MUSCLE en 1997, comme un moyen d'utiliser les cartes à puce sur différentes plateformes à l'université de Purdue. Après avoir sévi comme consultant « carte » pour différentes sociétés multinationales et agences gouvernementales, il a créé la société Trust-Bearer et fournit actuellement des solutions en ligne liées à l'utilisation des cartes à puce.

BIBLIOGRAPHIE

- **[gempc]** <http://ludovic.rousseau.free.fr/softwares/ifd-GemPC/index.html> – *GemCore based PC/SC reader drivers*.
- **[ifd-CCID]** <http://homepage.mac.com/jlgiraud/ifd-CCID/ifd-CCID.html> – *Drivers for CCID Smart Card Readers*.
- **[ifdhandler]** <http://pcslite.alioth.debian.org/ifdhandler-3/> – *MUSCLE PC/SC IFD Driver API*.
- **[coolkey]** http://directory.fedoraproject.org/wiki/CoolKey_CoolKey – *Fedora Directory Server*
- **[liste muscle]** <http://www.musclecard.com/list.html> *Linuxnet.com* – *MUSCLE – Linux Smart Card Development, Mailing List Support*.
- **[msdn]** [http://msdn2.microsoft.com/en-us/library/aa380149\(VS.85\).aspx](http://msdn2.microsoft.com/en-us/library/aa380149(VS.85).aspx) – *Smart Card Resource Manager API*.
- **[muscle drivers]** <http://www.musclecard.com/sourcedrivers.html> *Linuxnet.com* – *MUSCLE – Linux Smart Card Development, Mailing List Support, Card Reader Drivers*.
- **[pcslite]** <http://pcslite.alioth.debian.org/> – *Middleware to access a smart card using SCard API (PC/SC)*.
- **[pcsc v2]** <http://www.pcscworkgroup.com/specifications/overview.php> – *PC/SC Workgroup Specifications Overview*.
- **[pilote CCID]** <http://pcslite.alioth.debian.org/ccid.html> *CCID free software driver*
- **[sca]** <http://www.opensc-project.org/sca/wiki/LeopardBugs>



Les applications autour de la carte à puce communiquent avec celle-ci à travers l'envoi de séries d'octets appelées APDU. Python est un langage scripté dont la syntaxe facilite la manipulation de listes d'octets, tout en ayant une performance comparable à celle des langages compilés.

*Cet article est une introduction au développement d'applications Linux en langage Python autour de la carte à puce. L'installation et la configuration de **pyscard**, un framework Python de programmation d'applications pour la carte à puce, sont détaillées.*

*Outre la communication de base avec la carte, **pyscard** est un framework orienté objet incluant plusieurs classes facilitant la manipulation de cartes, lecteurs de carte, et services de haut niveau. Plusieurs exemples d'utilisation de **pyscard** sont présentés, ainsi que l'utilisation de **pycrypto**.*

Développement d'applications pour la carte à puce en langage Python

Auteur : Jean-Daniel Aussel

1

Introduction

Les cartes à microprocesseur, communément appelées « cartes à puce », sont des cartes ayant généralement la forme d'une carte de crédit, et embarquant un microprocesseur pouvant communiquer avec le monde extérieur à travers un port série à l'aide d'un protocole *half-duplex*. Ces cartes s'interfacent la plupart du temps avec des terminaux dédiés, tels que les terminaux de paiement ou les téléphones mobiles. Mais, dans de nombreuses applications, les cartes peuvent s'interfacer avec un ordinateur personnel qui contient une application dédiée à cette carte. Dans ce dernier cas, un lecteur de cartes est utilisé pour accéder à la carte.

Le groupe de travail PC/SC a défini une API standard d'interfaçage des cartes et lecteurs de cartes avec un système d'exploitation. L'implémentation de référence de PC/SC sur Linux est PC/SC Lite, une implémentation en langage C. La majeure partie des applications sur Linux utilisent PC/SC Lite pour communiquer avec les cartes, et sont naturellement écrites en C ou C++. Un framework Java est également disponible pour faire communiquer des applications Java avec les cartes à microprocesseur. Le framework OCF (*Open Card Framework*), qui s'appuie sur PC/SC Lite à travers la bibliothèque OCF to PC/SC Shim, et certaines applications sont aussi écrites en langage Java. La majeure partie des applications PC sur Linux est donc écrite en langage C, C++ et Java.

Cet article décrit l'utilisation de **pyscard**, un framework de développement d'applications en langage Python pour les cartes à microprocesseur. Les avantages de Python par rapport aux autres langages de programmation ne sont pas discutés, le choix d'un langage de programmation étant généralement plus passionnel ou idéologique qu'objectif.

2

Installation et configuration des dépendances

Cette section décrit l'installation et la configuration des paquetages nécessaires à l'exécution de **pyscard**. Certaines de ces dépendances sont probablement déjà installées par défaut dans certaines distributions Linux, comme **python** et **pcsc-lite**. Certaines sont disponibles dans les distributions majeures, mais non installées

par défaut, comme **swig** et **python-devel**, et finalement certaines ne sont pas disponibles dans les distributions majeures et doivent être téléchargées et installées individuellement, comme **pycrypto**.

2.1 Interpréteur Python

L'interpréteur et les bibliothèques **python** sont généralement installés dans les principales distributions Linux. Sinon, un paquetage binaire est habituellement fourni pour l'installation. En dernier recours, il vous faudra télécharger Python à partir de www.python.org, et le construire.

2.2 Pré-requis pour la construction de pycard

Dans le cas où une distribution binaire de **pycard** n'est pas disponible pour votre distribution Linux (c'est-à-dire dans la majeure partie des cas), il vous faudra le reconstruire à partir des sources. **pycard** est constitué d'une extension en C de Python, c'est-à-dire une bibliothèque dynamique **scard**, **pyd**, et d'une bibliothèque en pur Python. La construction de cette extension native nécessite le compilateur **gcc**, les bibliothèques de développement Python nécessaires à la construction d'une extension en C, du paquetage **libpcslite-devel** qui contient les définitions et structures nécessaires à la construction de **scard.pyd**, et finalement de **swig**, le *Simplified Wrapper and Interface Generator*. Là encore, **swig** est généralement disponible en paquetage binaire pour les principales distributions, mais peut être téléchargé à partir de www.swig.org et reconstruit.

2.3 Pré-requis pour l'exécution de pycard

pycard est construit au-dessus de PC/SC Lite, et requiert **pcsc-lite** pour son exécution. De plus, la plupart des lecteurs de cartes à microprocesseur implémentent aujourd'hui le standard CCID (*Circuit Card Interface Device*), et il vous faudra installer le pilote CCID. Les paquetages **pcsc-lite** et **ccid** sont disponibles dans la plupart des distributions, mais peuvent aussi être téléchargés de pcslite.alioth.debian.org.

2.4

Pré-requis optionnels

Le développement d'applications autour de la carte à microprocesseur nécessite souvent, si ce n'est tout le temps, des calculs cryptographiques. La bibliothèque Python de référence pour la cryptographie est **pycrypto**, le *Python Cryptography Toolkit*, disponible sur le lien www.amk.ca/python/code/crypto.

Les exemples de cet article sont des applications console. Le framework wxPython peut être utilisé pour développer des applications avec interface utilisateur, et certains exemples de **pycard** sont basés sur wxPython. Il vous faudra installer le paquetage wxPython ou le télécharger à partir de www.wxpython.org et le construire, si vous voulez exécuter les exemples en mode graphique.

2.5

Pré-requis matériels

Finalement, l'exécution des exemples nécessite un ou des lecteur(s) de cartes et des cartes à microprocesseur. De nombreux ordinateurs personnels viennent déjà pré-installés avec un lecteur de cartes, ou de nombreux sites offrent des lecteurs de cartes à la vente, tels que le magasin en ligne de Gemalto : <http://store.gemalto.com/is-bin/INTERSHOP.enfinity/eCS/Store>

En ce qui concerne les cartes, il vous sera possible d'expérimenter avec celles que vous avez déjà en votre possession : la carte SIM de votre téléphone mobile, votre carte bancaire EMV, votre carte d'identité Belpic, ou votre passeport électronique. Les spécifications techniques de ces cartes sont disponibles en ligne, et bien que n'ayant pas les clés émetteur de ces cartes, vous pourrez lire un certain nombre d'informations à partir de ces cartes. Le magasin en ligne de Gemalto offre aussi de nombreuses cartes que vous pourrez programmer vous-même, telles que les cartes JavaCard, pour lesquelles vous pourrez développer des applications sous forme d'applet JavaCard, un dérivé du langage Java pour les cartes à microprocesseur.

3

Téléchargement, construction et installation de pycard

Le framework **pycard** est disponible au téléchargement sur sourceforge. La page d'accueil du projet est <http://pycard.sourceforge.net>, et indique les liens vers les pages de téléchargement. Un nombre minimal de paquetages binaires est disponible, et il est fort probable que le paquetage binaire correspondant à votre distribution ne soit pas disponible au téléchargement. Il faudra donc télécharger le paquetage source et construire **pycard**.

Les paquetages de sources sont nommés **pycard-x.y.z.tar.gz**, où **x.y.z** est un numéro de version. A l'impression de ce numéro hors série, la version la plus récente de **pycard** est 1.6.7, et le paquetage source correspondant est **pycard-1.6.7.tar.gz**. Désarchivez le paquetage dans un répertoire de votre choix.

La construction et l'installation de **pycard** se font à l'aide du standard de distribution **distutils** de Python. Un script **distutils setup.py** est disponible dans le répertoire racine des sources. Pour construire et installer **pycard**, exécutez avec les priviléges **root** :

```
/usr/bin/python setup.py build_ext install
```

Cette commande compile l'extension native de **pycard** (**build_ext**) puis installe la distribution dans le répertoire **site-packages** de Python. Si votre distribution a une version 2.5 de Python, le paquetage **pycard** sera installé dans le répertoire **/usr/lib/python2.5/site-packages/smardcard**.

Ce répertoire peut cependant changer suivant les installations de Python. Pour vérifier que l'installation s'est bien déroulée, exécutez **/usr/bin/python**, et au prompt, entrez les commandes suivantes :

```
$ /usr/bin/python
Python 2.5.2 (r252:60911, Aug  5 2008, 16:17:06)
[GCC 4.2.2 20071128 (prerelease) (4.2.2-3.lmdv2008.0)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import smartcard
```

```
>>> print smartcard.System.readers()
['SCM SCR 3310 (21120432100028) 00 00', 'Axalto Reflex USB v3
(21120522106594 01 00']
>>> ^Z
```

Un certain nombre d'erreurs peut survenir lors de la construction de **pyscard**. En cas d'erreur d'inclusion de **winscard.h**, vérifiez que **libpcsc-lite-devel** est bien installé. En cas d'erreur d'inclusion de **python.h**, vérifiez que **python-devel** est bien installé.

4

Prise en main rapide de pyscard

Cette section présente les méthodes de base de **pyscard**, ainsi que certains concepts fondamentaux des cartes à microprocesseur. Les exemples donnés peuvent soit être entrés interactivement dans l'interpréteur Python, soit être assemblés dans un script Python, c'est-à-dire un fichier texte avec extension **.py**.

Pour les personnes non familières avec le langage Python, un excellent didacticiel est livré dans la documentation de Python et est disponible en ligne également. Dans le cas d'une exécution interactive, exécutez Python à travers la commande **/usr/bin/python** ou **/usr/local/bin/python**, en fonction du répertoire d'installation, et entrez les commandes une par une au prompt Python **>>>**. Les touches **Ctrl + D** permettent de sortir de l'interpréteur.

Dans le cas d'une exécution par script, assemblez les commandes dans un fichier d'extension **.py**, par exemple **monscript.py**, et exécutez le script en utilisant la commande **/usr/bin/python monscript.py**. Une alternative est d'inclure la ligne suivante au début de **monscript.py** :

```
#! /usr/bin/env python
```

et d'ajouter l'attribut exécutable au fichier **monscript.py**.

4.1

Les lecteurs de cartes

Une application a généralement besoin de se connecter à une carte afin d'envoyer des commandes APDU. Cette connexion se fait à travers un lecteur de cartes, qui est soit un lecteur conventionnel dans lequel la carte est insérée, soit un lecteur sans contact pour les cartes sans contact.

La liste des lecteurs de cartes disponibles peut être obtenue avec la méthode **readers()** du paquetage Python **smartcard.System**, par exemple :

```
>>> from smartcard.System import readers
>>> r=readers()
>>> print r
['SCM SCR 3310 (21120432100028) 00 00', 'Axalto Reflex USB v3
(21120522106594 01 00']
```

La méthode **readers()** retourne la liste des lecteurs présents, et la méthode **createConnection()** peut être appliquée à chaque lecteur pour se connecter à la carte insérée dans ce lecteur :

```
>>> connection = r[0].createConnection()
>>> connection.connect()
```

Une fois la connexion créée avec **createConnection()**, l'objet **Connection** peut être utilisé pour envoyer des commandes APDU à la carte :

```
>>> data, sw1, sw2 = connection.transmit([0xA0, 0xA4, 0x00, 0x00, 0x02, 0x7F, 0x10])
>>> print "%x %x" % (sw1, sw2)
9f 1a
```

Dans l'exemple précédent, la commande APDU transmise à la carte est **A0 A4 00 00 02 7F 10**, qui est une commande de sélection du répertoire **7F10** sur une carte SIM. La carte répond par une réponse APDU dont les codes de retour, appelés « mots d'état » ou *Status Words* en anglais, sont conventionnellement dénommés SW1 et SW2. Les mots d'état indiquent en particulier le succès ou l'erreur de l'exécution de chaque commande envoyée à la carte.

L'exemple précédent permet de se connecter à la carte à travers le lecteur de cartes d'indice 0, c'est-à-dire le premier lecteur de cartes du système. Cette approche centrée lecteur présente certains inconvénients.

Le premier inconvénient est que la carte avec laquelle l'application veut communiquer n'est peut-être pas insérée dans le premier lecteur, mais dans le deuxième. Il faudra alors modifier le script pour remplacer **r[0]** par **r[1]**, ce qui n'est pas désirable pour une application en général.

Le deuxième inconvénient est que la carte n'est peut-être pas insérée dans le lecteur, auquel cas l'exécution du script provoquera une exception **CardConnectionException** qui indique que le lecteur n'a aucune carte insérée.

Enfin, cette approche orientée lecteur ne vérifie pas que la carte insérée est du type attendu par l'application. Il se pourrait que la carte ne soit pas une carte SIM, et soit une carte EMV par exemple, auquel cas la sélection du répertoire **7F10**, et l'application carte en général, échouerait.

La plupart de ces restrictions est résolue par l'utilisation de techniques de détection de cartes, basées sur des critères de détection tels que l'ATR (*Answer To Reset*).

4.2 L'ATR

La première réponse d'une carte insérée dans un lecteur est appelée l'ATR (Answer To Reset). La raison principale de l'ATR est de décrire les paramètres de communication supportés pour établir un dialogue avec le lecteur de cartes. L'ATR est une série d'octets normalisés par la norme ISO7816-3. Les premiers octets décrivent en particulier les conventions de communication avec la carte, puis les interfaces disponibles et leurs paramètres. Ces octets sont suivis d'une série d'octets appelés « octets historiques », pour lesquels aucune norme n'existe, et qui sont utilisés pour transmettre des informations diverses telles que le type de carte, la version des logiciels embarqués, l'état de la carte. Finalement, ces octets historiques sont éventuellement suivis d'un octet de vérification, *checksum* des octets précédents.

L'ATR peut être utilisé pour détecter une carte soit en sa totalité, c'est-à-dire détecter une carte ayant un ATR donné, insérée dans n'importe quel lecteur, soit en partie, par exemple à partir des octets historiques, qui permettent de détecter une carte d'une version particulière ou dans un état particulier.

L'exemple suivant affiche l'ATR des cartes insérées dans les lecteurs :

```
#!/usr/bin/env python
from smartcard.Exceptions import NoCardException
from smartcard.System import readers
from smartcard.util import toHexString

for reader in readers():
    try:
        connection=reader.createConnection()
        connection.connect()
        print reader, '-', toHexString( connection.getATR() )
    except NoCardException:
        print reader, 'no card inserted'
```

L'exécution de ce script produit une impression du type :

```
SCM SCR 3310 (21120432100028) 00 00 - 3B E5 00 00 81 21 45 9C 10 01 00 80 0D
Axalto Reflex USB v3 (21120522106594 01 00 - no card inserted
```

où chaque nom de lecteur est suivi de l'ATR de la carte qu'il contient.

pyscard contient une classe ATR qui permet la manipulation de l'ATR, comme le montre l'exemple suivant d'analyse de l'ATR d'une carte USIM, c'est-à-dire 3G :

```
#!/usr/bin/env python
from smartcard.ATR import ATR
from smartcard.util import toHexString

atr = ATR([0x3B, 0x9E, 0x95, 0x80, 0x1F, 0xC3, 0x80, 0x31, 0xA0, 0x73,
           0xBE, 0x21, 0x13, 0x67, 0x29, 0x02, 0x01, 0x81, 0xCD, 0xB9] )

print atr
print 'octets historiques: ', toHexString( atr.getHistoricalBytes() )
print 'checksum: ', "%02X" % atr.getChecksum()
print 'checksum OK: ', atr.checksumOK
print 'support T0: ', atr.isT0Supported()
print 'support T1: ', atr.isT1Supported()
print 'support T15: ', atr.isT15Supported()
```

qui produit la sortie suivante :

```
3B E5 95 80 1F C3 80 31 A0 73 BE 21 13 67 29 02 01 01 81 CD B9
octets historiques:  80 31 A0 73 BE 21 13 67 29 02 01 01 81 CD
checksum:  0xB9
checksum OK:  True
support T0:  True
support T1:  False
support T15:  True
```

4.3

Les protocoles de communication T=0 et T=1

En général, les paramètres de communication sont surtout importants pour la négociation de protocoles avec le lecteur de cartes. Il est important de savoir néanmoins que les principaux protocoles sont les protocoles T=0 et T=1, respectivement pour la transmission des données par octet ou par bloc d'octets.

Le protocole de transmission peut-être spécifié soit lors de la connexion à la carte, soit lors de la transmission des commandes. L'exemple suivant permet de sélectionner le protocole T=1 à la connexion :

```
#!/usr/bin/env python
from smartcard.CardConnection import CardConnection
from smartcard.Exceptions import NoCardException, CardConnectionException
from smartcard.System import readers
from smartcard.util import toHexString

for reader in readers():
    try:
        connection=reader.createConnection()
        connection.connect( CardConnection.T1_protocol )
        print reader, '-', toHexString( connection.getATR() )
    except NoCardException:
        print reader, 'no card inserted'
    except CardConnectionException, e:
        print reader, e
```

Une exception est générée si le lecteur n'a pas de carte insérée, ou si le protocole T=1 n'est pas supporté :

```
SCM SCR 3310 (21120432100028) 00 00 - 3B E5 00 00 81 21 45 9C 10 01 00 80 0D
Axalto Reflex USB v3 (21120522106594 01 00 - 'Smartcard Exception: Unable
to connect with protocol: T1 Card protocol mismatch.'!
```

Une analyse de l'ATR permet d'ailleurs de vérifier que le protocole T=1 est supporté :

```
from smartcard.ATR import ATR
from smartcard.util import toBytes
atr = ATR( toBytes( "3B E5 00 00 81 21 45 9C 10 01 00 80 0D" ) )
print atr
print 'support T0: ', atr.isT0Supported()
print 'support T1: ', atr.isT1Supported()
```

Qui produit la sortie suivante :

```
3B E5 00 00 81 21 45 9C 10 01 00 80 0D
support T0:  False
support T1:  True
```

Le protocole de communication peut aussi être précisé lors de la transmission d'APDU, comme le montre le script suivant :

```
from smartcard.System import readers
from smartcard.CardConnection import *
r=readers()
connection = r[0].createConnection()
connection.connect()
data, sw1, sw2 = connection.transmit([0xA0, 0xA4, 0x00, 0x00, 0x02,
0x7F,0x10], CardConnection.T1_protocol )
print "%x %x" % (sw1, sw2)
```

Les protocoles sont définis dans la classe **CardConnection**, et les valeurs **T0_protocol**, **T1_protocol**, **T15_protocol**, **RAW_protocol** peuvent être utilisées.

4.4

Manipulation de listes d'octets

Python est particulièrement efficace pour la manipulation de listes d'octets telles que les APDU, comme le montrent les exemples de syntaxe suivante :

```
from smartcard.util import toBytes, toHexString
response = toBytes( "04 0F 00 01 7F 10 02 00 00 00 00 00 15 19 01 10 02 00
83 8A 83 8A 00 00 90 00" )
print response # response as list of bytes
print response[-2:] # deux derniers octets
print response[3:15] # liste des octets 4 à 16
print response[response[0]-1:response[1]] # liste des octets de rang
response[0] a response[1]
print response + (32-len(response))*[0xFF] # pad to 32 with FFs
print response[-2:]==[ 0x90, 0 ] # test for equality
```

5

Détection d'une carte à l'aide du CardType

Les objets **CardType** et **CardRequest** permettent d'éviter les problèmes liés aux scripts basés sur la connexion de la carte à partir d'un lecteur. Dans cette approche centrée sur la carte et non le lecteur, l'application fait une requête (**CardRequest**) pour une carte d'un type bien précis (**CardType**). Lors de l'insertion d'une carte correspondant au **CardType** demandé, une connexion vers cette carte est automatiquement créée et retournée à l'application.

5.1

Détection d'une carte par son ATR, l'ATRCardType

La détection d'une carte par son ATR est illustrée dans l'exemple suivant :

```
>>> from smartcard.CardType import ATRCardType
>>> from smartcard.CardRequest import CardRequest
>>> from smartcard.util import toHexString, toBytes
>>>
>>> cardtype = ATRCardType( toBytes( "3B 16 94 20 02 01 00 00 00" ) )
>>> cardrequest = CardRequest( timeout=10, cardType=cardtype )
>>> cardservice = cardrequest.waitforcard()
>>>
>>> cardservice.connection.connect()
>>> print toHexString( cardservice.connection.getATR() )
3B 16 94 20 02 01 00 00 00
>>>
>>> SELECT = [0xA0, 0xA4, 0x00, 0x00, 0x02]
>>> DF_TELECOM = [0x7F, 0x10]
>>> data, sw1, sw2 = cardservice.connection.transmit( SELECT + DF_TELECOM )
>>> print "%x %x" % (sw1, sw2)
9f 1a
>>>
```

Dans cet exemple, un objet **CardType** dédié à un ATR précis, **l'ATRCardType**, est créé avec les octets de l'ATR dans le constructeur. Une **CardRequest** est alors créée avec un *timeout* de 10 secondes et l'objet **ATRCardType**. Puis la

méthode **waitforcard()** de l'objet **CardRequest** est appelée. Cet appel est bloquant pendant la durée de la temporisation (**timeout**), et la temporisation par défaut est d'une seconde. Si une carte correspondant à la requête est introduite pendant la durée de la temporisation, **waitfocard()** retourne un objet **CardService** qui permet de transmettre des APDU à la carte. Sinon, une exception de type **smartcard.Exceptions.CardRequestTimeoutException** est lancée.

Cette approche permet de s'affranchir de la connaissance des lecteurs de cartes : la détection et la connexion à la carte se font à partir de l'ATR de la carte, quel que soit le lecteur utilisé. Si nécessaire, le lecteur de cartes utilisé peut être obtenu à partir de l'objet **CardService** retourné par **CardRequest** :

```
>>> print cardservice.connection.getReader()
Axalto Reflex USB v3 (21120522106594 01 00)
```

Un masque peut aussi être appliqué à l'ATR à détecter. Seuls les bits non masqués de l'ATR seront examinés pour la détection de la carte. Le masque optionnel est spécifié à la construction :

```
cardtype = ATRCardType( toBytes( "3B 15 94 20 02 01 00 00 0F" ), toBytes(
"00 00 FF FF FF FF FF FF 00" ) )
```

5.2

Détection de l'insertion de n'importe quelle carte

Dans certains cas, il est désirable d'attendre l'insertion d'une carte, quel que soit son type. Ceci est possible avec la classe **AnyCardType** :

```
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=None, cardType=cardtype )
cardservice = cardrequest.waitforcard()
```

Dans l'exemple précédent, `waitforcard()` est un appel bloquant jusqu'à l'insertion de n'importe quelle carte, car une température infinie a été donnée au constructeur de `CardRequest(timeout=None)`.

5.3

Création de types de cartes dédiés

La classe `CardType` peut être dérivée, afin de développer des `CardType` dédiés par exemple à la détection de cartes EMV ou passeport ou SIM,... La méthode `match()` de la classe `CardType` doit être surchargée pour implémenter la logique de reconnaissance d'une carte spécifique.

Pour définir un type de carte qui supporte la convention directe, c'est-à-dire un premier octet d'ATR à 0x3B, il suffit de surcharger la méthode `match()` pour examiner la valeur de ce premier octet :

```
>>> from smartcard.CardType import CardType
>>> from smartcard.CardRequest import CardRequest
>>>
>>> class DCCardType(CardType):
...     def matches( self, atr, reader=None ):
...         return atr[0]==0x3B
...
>>> cardtype = DCCardType()
>>> cardrequest = CardRequest( timeout=1, cardType=cardtype )
>>> cardservice = cardrequest.waitforcard()
```

La méthode `match()` reçoit également le lecteur dans lequel la carte à tester est insérée. Il est alors possible d'implémenter une logique de reconnaissance qui se connecte à la carte et vérifie la présence de données, par exemple en essayant de sélectionner une application. L'exemple suivant de `CardType`

reconnaît les cartes ayant un répertoire `DF_TELECOM`, c'est-à-dire une carte SIM :

```
class GSMCardType(CardType):
    def matches( self, atr, reader=None ):
        connection = reader.createConnection()
        connection.connect()
        SELECT = [0xA0, 0xA4, 0x00, 0x00, 0x02]
        DF_TELECOM = [0x7F, 0x10]
        data, sw1, sw2 = connection.transmit( SELECT + DF_TELECOM )
        return sw1==0x9F
```

Le constructeur du `CardType` peut aussi être surchargé pour dériver des `CardTypes` génériques. `HistoricalBytesCardType` dans l'exemple suivant permet de détecter une carte basée sur ses bits historiques :

```
from smartcard.ATR import ATR
from smartcard.CardType import CardType
from smartcard.CardRequest import CardRequest
from smartcard.util import toHexString, toBytes

class HistoricalBytesCardType(CardType):
    def __init__( self, historicalbytes ):
        """HistoricalBytesCardType constructor."""
        self.historicalbytes = historicalbytes

    def matches( self, atr, reader=None ):
        matched=False
        catr = ATR( atr )
        return self.historicalbytes==catr.getHistoricalBytes()

cardtype = HistoricalBytesCardType( toBytes("46 04 6C 90 00") )
cardrequest = CardRequest( timeout=10, cardType=cardtype )
cardservice = cardrequest.waitforcard()

cardservice.connection.connect()
print toHexString( cardservice.connection.getATR() )
```

6

Le traçage des commandes et réponses APDU

Il est souvent désirable de tracer les échanges d'APDU durant l'exécution d'une application. L'insertion des traces devient rapidement illisible, comme le montre l'exemple suivant qui enchaîne plusieurs commandes :

```
from smartcard.CardType import ATRCardType
from smartcard.CardRequest import CardRequest
from smartcard.util import toHexString, toBytes

cardtype = ATRCardType( toBytes( "3B 16 94 20 02 01 00 00 0D" ) )
cardrequest = CardRequest( timeout=1, cardType=cardtype )
cardservice = cardrequest.waitforcard()
cardservice.connection.connect()
```

```
SELECT = [0xA0, 0xA4, 0x00, 0x00, 0x02]
DF_TELECOM = [0x7F, 0x10]
apdu = SELECT+DF_TELECOM
print 'sending ' + toHexString(apdu)
response, sw1, sw2 = cardservice.connection.transmit( apdu )
print 'response: ', response, ' status words: ', "%x %x" % (sw1, sw2)

if sw1 == 0x9F:
    GET_RESPONSE = [0XA0, 0XC0, 00, 00 ]
    apdu = GET_RESPONSE + [sw2]
    print 'sending ' + toHexString(apdu)
    response, sw1, sw2 = cardservice.connection.transmit( apdu )
    print 'response: ', toHexString(response), ' status words: ', "%x %x" %
```

```
(sw1, sw2)
```

```
"
```

Cette solution de traçage est particulièrement difficile à lire, le code de traçage étant plus verbeux que la logique même de l'application. L'utilisation de fonctions **trace_command** et **trace_response** diminuerait la verbosité, mais triplerait quand même le nombre d'instructions :

```
apdu = SELECT+DF_TELECOM
trace_command(apdu)
response, sw1, sw2 = cardservice.connection.transmit( apdu )
trace_response( response, sw1, sw2 )
if sw1 == 0x9F:
    GET_RESPONSE = [0XA0, 0XC0, 00, 00 ]
    apdu = GET_RESPONSE + [sw2]
    trace_command(apdu)
    response, sw1, sw2 = cardservice.connection.transmit( apdu )
    trace_response( response, sw1, sw2 )
```

6.1

Le Design Pattern observer appliqué au traçage

pyscard implémente le *design pattern* observer pour le traçage des APDU. Ce pattern est décrit en détail dans le livre *Thinking in Python* de Bruce Eckel, ainsi que les design pattern présentés plus loin dans cet article. Le principe est qu'un objet de type **ConnectionObserver** est déclaré auprès de la connexion carte à observer. Cet objet est notifié de l'envoi ou de la réponse d'APDU sur sa méthode **update()**, comme le montre l'exemple suivant :

```
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnectionObserver import ConsoleCardConnectionObserver

GET_RESPONSE = [0XA0, 0XC0, 00, 00 ]
SELECT = [0XA0, 0XA4, 0x00, 0x00, 0x02]
DF_TELECOM = [0x7F, 0x10]
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=10, cardType=cardtype )
cardservice = cardrequest.waitforcard()

observer=ConsoleCardConnectionObserver()
cardservice.connection.addObserver( observer )
cardservice.connection.connect()

apdu = SELECT+DF_TELECOM
response, sw1, sw2 = cardservice.connection.transmit( apdu )
if sw1 == 0x9F:
    apdu = GET_RESPONSE + [sw2]
    response, sw1, sw2 = cardservice.connection.transmit( apdu )
else:
    print 'no DF_TELECOM'
```

Dans cet exemple, un objet **ConsoleCardConnectionObserver** est attaché au **CardService** retourné par **waitforcard()**. Cet observer affiche sur la console les événements de connexion, déconnexion et échange d'APDU :

```
connecting to Axalto Reflex USB v3 (21120522106594) 00 00
> A0 A4 00 00 02 7F 10
< [] 9F 22
> A0 C0 00 00 22
< 00 00 00 01 7F 10 02 00 00 00 00 00 15 19 01 10 02 00 83 8A 83 8A 00 00
00 00 00 00 00 00 00 00 00 00 00 00 90 0
disconnecting from Axalto Reflex USB v3 (21120522106594) 00 00
disconnecting from Axalto Reflex USB v3 (21120522106594) 00 00
```

6.2

Écriture d'un CardConnectionObserver spécifique

La classe **CardConnectionObserver** peut être dérivée pour implémenter différentes fonctionnalités liées au traçage des événements carte. Des exemples d'applications sont le traçage d'APDU dans une fenêtre graphique de type wxPython ou GTK, le traçage d'APDU dans un fichier texte, ou l'interprétation d'APDU.

Une classe dérivée de **CardConnectionObserver** doit surcharger la méthode **update()**, qui est notifiée avec des événements de type **CardConnectionEvent**. La classe **TracerAndSELECTInterpreter** suivante par exemple fait une interprétation simple des APDU **SELECT** et **GET_RESPONSE** avant d'imprimer les échanges à l'écran :

```
from smartcard.CardConnectionObserver import CardConnectionObserver
from smartcard.util import toHexString
from string import replace
class TracerAndSELECTInterpreter( CardConnectionObserver ):
    def update( self, cardconnection, ccevent ):
        if 'connect'==ccevent.type:
            print 'connecting to ' + cardconnection.getReader()
        elif 'disconnect'==ccevent.type:
            print 'disconnecting from ' + cardconnection.getReader()
        elif 'command'==ccevent.type:
            str=toHexString(ccevent.args[0])
            str = replace( str , "A0 A4 00 00 02", "SELECT" )
            str = replace( str , "A0 C0 00 00", "GET RESPONSE" )
            print '> ', str
        elif 'response'==ccevent.type:
            if []==ccevent.args[0]:
                print '< [] ', "%-2X %-2X" % tuple(ccevent.args[-2:])
            else:
                print '<',toHexString(ccevent.args[0]),"%-2X %-2X" %
                    tuple(ccevent.args[-2:])
```

7

La gestion des erreurs

Les exemples précédents réalisaient peu d'échanges d'APDU, et n'adressaient pas les problèmes liés aux états d'erreurs que peut retourner la carte durant les échanges d'APDU. Sur l'envoi d'une commande APDU, la carte peut retourner un certain nombre de mots d'état SW1-SW2 correspondant à plusieurs cas de succès ou d'erreur. Certains de ces cas d'erreur sont normalisés par les normes ISO7816-4, ISO7816-8 ou ISO7816-9 par exemple. D'autres cas d'erreur sont normalisés par les normes Open Platform (carte Javacard), 3GPP (carte SIM), EMV (carte bancaire), et finalement certains cas d'erreur ne sont pas normalisés et dépendent d'une carte applicative dédiée (par exemple l'applet MUSCLE), ou d'une applet Javacard que vous avez vous-même écrite.

Certains codes d'erreur sont uniques, mais certains codes ont une signification différente dans différentes normes ; par exemple la norme ISO7816-4 définit le code d'erreur 0x62 0x83 comme « *File invalidated* », alors que la norme *open platform* version 2.1 définit le même code d'erreur par « *Card life cycle is CARD_LOCKED* ». On voit donc que les codes d'erreur possibles et leur interprétation sont liés au type de carte utilisé et ne peuvent pas être généralisés.

7.1

Gestion des erreurs par chaîne de responsabilité

Pour la gestion des erreurs, `pyscard` utilise la notion de vérificateur d'erreur `ErrorChecker` et utilise le design pattern chaîne de responsabilité. Un objet `ErrorChecker` contient un certain nombre de mots d'état d'erreur qu'il est capable de gérer et interpréter, par exemple les mots d'état de la norme 7816-8 (`ISO7816_8ErrorChecker`) ou les mots d'état de la norme Open Platform 2.1 (`op21_ErrorChecker`). Plusieurs objets `ErrorChecker` peuvent être chaînés dans une chaîne de responsabilité adaptée à une carte, un objet `ErrorCheckingChain`, qui sera responsable de gérer les erreurs retournées par les mots d'état. Ceci est illustré dans l'exemple suivant :

```
from smartcard.sw.ISO7816_4ErrorChecker import ISO7816_4ErrorChecker
from smartcard.sw.ISO7816_8ErrorChecker import ISO7816_8ErrorChecker
from smartcard.sw.ISO7816_9ErrorChecker import ISO7816_9ErrorChecker

from smartcard.sw.ErrorCheckingChain import ErrorCheckingChain

errorchain = []
errorchain=[ ErrorCheckingChain( errorchain, ISO7816_9ErrorChecker() ),
            ErrorCheckingChain( errorchain, ISO7816_8ErrorChecker() ),
            ErrorCheckingChain( errorchain, ISO7816_4ErrorChecker() ) ]

errorchain[0]( [], 0x90, 0x00 )
errorchain[0]( [], 0x6A, 0x8a )
```

Dans cet exemple, trois objets `ErrorChecker` sont créés et chaînés dans une chaîne de vérification d'erreur `errorchain`. Chaque code d'erreur peut être vérifié en appelant le premier élément de la chaîne. Dans le premier appel, `errorchain[0]([], 0x90, 0x00)`, les mots d'état 90 00 ne correspondent pas à une erreur gérée par la chaîne. L'appel à la chaîne retourne sans lever d'exception.

Dans le deuxième appel, `errorchain[0]([], 0x6A, 0x8a)`, le code d'erreur est reconnu par la chaîne et une exception est levée pour reporter l'erreur. L'exécution de ce script a pour résultat :

```
Traceback (most recent call last):
  File "/tmp/test/t7.py", line 13, in <module>
    errorchain[0]( [], 0x6A, 0x8a )
  File "/usr/lib/python2.5/site-packages/smartcard/sw/ErrorCheckingChain.py", line 71, in __call__
    self.strategy( data, sw1, sw2 )
  File "/usr/lib/python2.5/site-packages/smartcard/sw/ISO7816_9ErrorChecker.py", line 85, in __call__
    raise exception( data, sw1, sw2, message )
smartcard.sw.SWExceptions.CheckingErrorException: 'Status word exception: checking error - DF name already exists!'
```

7.2

Attachement de la gestion des erreurs à une connexion carte

Une chaîne de vérification d'erreur peut être attachée à une connexion carte en utilisant la méthode `setErrorCheckingChain` de l'objet `CardConnection`:

```
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=10, cardType=cardtype )
cardservice = cardrequest.waitforcard()
errorchain=[]
errorchain=[ ErrorCheckingChain( errorchain, ISO7816_8ErrorChecker() ),
            ErrorCheckingChain( errorchain, ISO7816_4ErrorChecker() ) ]
cardservice.connection.setErrorCheckingChain( errorchain )
```

Lors de l'envoi d'APDU sur cette connexion, les mots d'état SW1-SW2 seront vérifiés sur la chaîne de gestion d'erreur, et une exception correspondant à une erreur sera éventuellement levée.

7.3

Filtrage des exceptions levées par la gestion des erreurs

Quatre types d'exceptions peuvent être levés : `WarningProcessingException`, `ExecutionErrorException`, `SecurityRe`

latedException, et **CheckingErrorException**. Le choix de l'exception levée en fonction du code d'erreur est implémenté dans l'objet **ErrorChecker**. Par exemple, le vérificateur d'erreur **IS07816_8ErrorChecker** implémente la résolution suivante pour les SW1 0x65, 0x66 et 0x67 :

```
0x65:( smartcard.sw.SWExceptions.ExecutionErrorException,
{ 0x81:"Memory failure (unsuccessful changing)" } ),
0x66:( smartcard.sw.SWExceptions.SecurityRelatedException,
{ 0x00:"The environment cannot be set or modified",
  0x87:"Expected SM data objects missing",
  0x88:"SM data objects incorrect" } ),
0x67:( smartcard.sw.SWExceptions.CheckingErrorException,
{ 0x00:"Wrong length (empty Lc field)" } ),
```

Certaines exceptions peuvent être filtrées, par exemple, une application peut décider de ne pas lever d'exception **WarningProcessingException** en cas d'erreur pour poursuivre l'exécution. Le filtrage d'exception s'effectue à l'aide de la méthode **addFilterException** de la chaîne d'erreur :

```
from smartcard.sw.SWExceptions import WarningProcessingException
errorchain[0].addFilterException( WarningProcessingException )
```

Dans ce cas particulier, l'exécution de la commande **errorchain[0]([], 0x62, 0x00)** ne causera pas de levée d'exception.

7.4

Conception d'une ErrorCheckingChain dédiée

Dans la majeure partie des cas, les **ErrorChecker** normalisés ne suffiront pas, et il faudra écrire un ou plusieurs **ErrorChecker** correspondant aux normes ou spécifications de l'application carte. Pour implémenter un vérificateur d'erreur, dériver la classe **ErrorChecker**, et surcharger la méthode **_call_**.

8

Le changement dynamique du comportement d'une connexion carte

L'objet **CardConnection** permet de transmettre des APDU à une carte. Dans certains cas, il peut être désirable de changer de façon transparente le comportement par défaut d'une connexion carte : pour établir de façon transparente un canal sécurisé par exemple, ou pour filtrer et changer à la volée certaines commandes APDU, ou même l'ATR reporté par la carte. Le design pattern Decorator a été implémenté dans **pyscard** pour changer le comportement dynamique d'une connexion à l'aide de l'objet **CardConnectionDecorator**. Cet objet apparaît comme un objet **CardConnection** lui-même,

L'exemple suivant, la classe **MyErrorChecker** lève une exception **SecurityRelatedException** pour les mots d'état 0x66 0x00 :

```
class MyErrorChecker( ErrorChecker ):
    def __call__( self, data, sw1, sw2 ):
        if 0x66==sw1 and 0x00==sw2:
            raise SecurityRelatedException( data, sw1, sw2 )
```

L'exemple suivant détecte quelques mots d'erreurs de l'applet **Muscle**:

```
from smartcard.sw.ErrorChecker import ErrorChecker
import smartcard.sw.SWExceptions

MuscleApplet_SW = {
    0x9C:( smartcard.sw.SWExceptions.ExecutionErrorException,
        { 0x01:"Insufficient memory onto the card to complete the operation",
          0x02:"Unsuccessful authentication. Multiple consecutive
failures cause the identity to block",
          0x03:"Operation not allowed because of the internal state of
the Applet",
          0x05:"The requested feature is not supported either by the
card or by the Applet" } ),
}

class MuscleApplet_ErrorChecker( ErrorChecker ):
    def __call__( self, data, sw1, sw2 ):
        """Called to test data, sw1 and sw2 for error.

        data:      apdu response data
        sw1, sw2:  apdu data status words

        Derived classes must raise a smartcard.sw.SWException upon error."""
        if MuscleApplet_SW.has_key( sw1 ):
            exception, sw2dir = MuscleApplet_SW[sw1]
            if type(sw2dir)==type({}):
                try:
                    message = sw2dir[sw2]
                    raise exception( data, sw1, sw2, message )
                except KeyError:
                    pass
```

mais s'attache au-dessus d'une **CardConnection** pour en changer son comportement en surchargeant la méthode dont le comportement est à modifier.

8.1

Changement dynamique de l'ATR d'une carte

Dans l'exemple suivant, la méthode **getATR()** du décorateur est surchargée, afin de retourner un ATR constant quelle que soit la carte :

```
class FakeATRConnection( CardConnectionDecorator ):
    '''This decorator changes the fist byte of the ATR.'''
    def __init__( self, cardconnection ):
        CardConnectionDecorator.__init__( self, cardconnection )

    def getATR( self ):
        """Replace first BYTE of ATR by 3F"""
        atr = CardConnectionDecorator.getATR( self )
        return toBytes( "3F 65 25 08 43 04 6C 90 00")
```

Pour appliquer le décorateur, il suffit d'appeler son constructeur avec l'objet connexion dont le comportement est à modifier, et d'utiliser le décorateur à la place de l'objet original :

```
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=1.5, cardType=cardtype )
cardservice = cardrequest.waitforcard()
# attach our decorator
cardservice.connection = FakeATRConnection( cardservice.connection )
cardservice.connection.connect()
print 'ATR', toHexString( cardservice.connection.getATR() )
```

8.2

Connexion exclusive à une carte

Par défaut, **pyscard** ouvre une connexion partagée avec la carte, c'est-à-dire que plusieurs applications peuvent accéder simultanément à la même carte. Deux décorateurs sont implémentés dans **pyscard** : **ExclusiveConnectCardConnection** qui permet de s'assurer qu'un process différent ne pourra pas se connecter à la carte, et **ExclusiveTransmitCardConnection** qui permet de s'assurer que la transmission d'une APDU est non interrompue par un autre thread. Ces décorateurs sont implémentés en surchargeant la méthode **connect** pour la connexion exclusive et **transmit** pour la transmission exclusive. L'exemple suivant montre comment chaîner les deux décorateurs :

```
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=5, cardType=cardtype )
cardservice = cardrequest.waitforcard()
```

9

Un mot sur la cryptographie

Les applications carte nécessitent en général des opérations cryptographiques, pour l'établissement d'un canal sécurisé avec la carte, ou pour l'authentification mutuelle application/carte par exemple. Une des principales bibliothèques cryptographiques en langage Python est **pycrypto**. Cette section montre

```
# attach our decorator
cardservice.connection = ExclusiveTransmitCardConnection( ExclusiveConnect
CardConnection( cardservice.connection ) )

# connect to the card
cardservice.connection.connect()
```

8.3

Implémentation d'un canal sécurisé

Le pattern décorateur est particulièrement adapté à la création d'un canal sécurisé transparent. Beaucoup de cartes établissent un canal sécurisé entre l'application hors carte et la carte, permettant d'assurer la confidentialité, l'intégrité et l'authenticité des APDU échangées. La cryptographie nécessaire à l'établissement de ce canal sécurisé dépend du type de carte et des normes ou spécifications supportées. Cependant, un canal sécurisé comprend en général une combinaison de hachage, signature et chiffrement avant l'envoi de l'APDU, et une combinaison de vérification de hachage et déchiffrement après réception de l'APDU. Le décorateur suivant est une ébauche de connexion sécurisée, dont les méthodes **cipher** et **uncipher** devront être adaptées à la cryptographie nécessaire à l'établissement du *secure channel*.

```
class SecureChannelConnection( CardConnectionDecorator ):
    def __init__( self, cardconnection ):
        CardConnectionDecorator.__init__( self, cardconnection )

    def cipher( self, bytes ):
        return bytes

    def uncipher( self, data ):
        return data

    def transmit( self, bytes, protocol=CardConnection.T0_protocol ):
        cypheredbytes = self.cipher( bytes )
        data, sw1, sw2 = CardConnectionDecorator.transmit( self,
        cypheredbytes, protocol )
        if []!=data:
            data = self.uncipher( data )
        return data, sw1, sw2
```

Dans cet exemple, l'initialisation du canal sécurisé doit être implémentée dans le constructeur, et le chiffrement/déchiffrement dans les méthodes **cipher/uncipher**, qui dans notre cas ne font que retourner les octets de l'APDU inchangés.

brièvement les principales opérations de **pycrypto** pour le développement d'applications carte. Ce didacticiel rapide vous permettra d'implémenter la cryptographie nécessaire à votre application carte.

9.1

Chaînes de caractères et listes d'octets

pycrypto manipule des chaînes de caractères, c'est-à-dire des objets Python de type **string** contenant des caractères, imprimables ou non, tels que '**\01\42\70\23**', alors que les applications carte sont orientées APDU et manipulent des listes d'octets, telles que [0x01, 0x42, 0x70, 0x23]. Les fonctions suivantes permettent de traduire une liste d'octets vers une chaîne de caractères et inversement.

```
def HexListToBinString( hexlist ):
    binstring=""
    for byte in hexlist:
        binstring= binstring + chr( eval( '0x%x' % byte ) )
    return binstring

def BinStringToHexList( binstring ):
    hexlist=[]
    for byte in binstring:
        hexlist= hexlist + [ ord(byte) ]
    return hexlist
```

Un exemple de conversion est :

```
>>> test_data = [ 0x01, 0x42, 0x70, 0x23 ]
>>> binstring = HexListToBinString( test_data )
>>> hexlist = BinStringToHexList( binstring )
>>> print binstring, hexlist
@Bp# [1, 66, 112, 35]
```

9.2

Hachage

pycrypto supporte les algorithmes de hachage SHA-1, MD2, MD4 et MD5. Un hash SHA de 16 octets est effectué de la manière suivante :

```
from Crypto.Hash import SHA
from smartcard.util import toHexString, PACK

test_data = [ 0x01, 0x42, 0x70, 0x23 ]
binstring = HexListToBinString( test_data )

zhash = SHA.new( binstring )
hash_as_string = zhash.digest()[:16]
hash_as_bytes = BinStringToHexList( hash_as_string )
print hash_as_string, ',', toHexString( hash_as_bytes, PACK )
```

De façon similaire, un hash MD5 est obtenu en remplaçant **SHA** par **MD5** dans le script précédent.

9.3

Algorithmes à clé secrète

pycrypto supporte de nombreux algorithmes à clé secrète tels que DES, triple DES, AES, blowfish, IDEA. Un chiffrement triple DES en mode ECB est effectué de la manière suivante :

```
from Crypto.Cipher import DES3
from smartcard.util import toBytes

key = "31323334353637383132333435363738"
key_as_binstring = HexListToBinString( toBytes( key ) )
zdes = DES3.new( key_as_binstring, DES3.MODE_ECB )

message = "71727374757677787172737475767778"
message_as_binstring = HexListToBinString( toBytes( message ) )

encrypted_as_string = zdes.encrypt( message_as_binstring )
decrypted_as_string = zdes.decrypt( encrypted_as_string )
print message_as_binstring, encrypted_as_string, decrypted_as_string
```

10

Interface graphique

Les exemples précédents démontraient l'utilisation de **pyscard** pour l'écriture d'applications en mode console. Il est aussi bien sûr possible de développer au-dessus de **pyscard** des applications graphiques pour les cartes à microprocesseur. wxPython est un framework graphique qui permet ce type de développement, et un certain nombre d'exemples et de classes sont livrés dans **pyscard** à cet effet.

10.1

Développement d'application simple wxPython

La classe **smartcard.wx.SimpleSCardApp** permet de développer rapidement une simple application wxPython, comme le montre l'exemple suivant :

```
import os.path
from smartcard.wx.SimpleSCardApp import *
from SamplePanel import SamplePanel
def main( argv ):
    app = SimpleSCardApp()
    appname = 'A tool to send apdu to a card',
    apppanel = SamplePanel,
    appstyle = TR_SMARTCARD | TR_READER | PANEL_APDUTRACER,
    appicon = os.path.join( os.path.dirname( __file__ ), 'images',
    'mysmartcard.ico' ),
    size = (800,600)
    app.MainLoop()

if __name__ == "__main__":
    import sys
    main( sys.argv )
```

Les paramètres du constructeur de SimpleSCardApp incluent le nom de l'application, la classe du Panel à afficher dans la fenêtre de l'application (ici **SamplePanel**), et le style d'application. L'option **TR_SMARTCARD** permet d'afficher un arbre des cartes insérées, l'option **TR_READER** permet d'afficher un arbre des lecteurs présents, et l'option **PANEL_APDUTRACER** permet d'afficher une fenêtre de traçage des APDU.

La logique de l'application carte est alors développée dans la classe **SamplePanel**, implémentant l'interface **SimpleSCardAppEventObserver** et dérivant la classe **wx.Panel** de wxPython

```
import os.path
from smartcard.wx.SimpleSCardApp import *
from smartcard.wx.SimpleSCardAppEventObserver import SimpleSCardAppEventObserver

ID_TEXT = 10000

class SamplePanel( wx.Panel, SimpleSCardAppEventObserver ):
    def __init__( self, parent ):
        wx.Panel.__init__( self, parent, -1 )

        sizer = wx.FlexGridSizer( 0, 3, 0, 0 )
        sizer.AddGrowableCol( 1 )
        sizer.AddGrowableRow( 1 )

        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )

        self.feedbacktext = wx.StaticText( self, ID_TEXT, "" ,
        wx.DefaultPosition, wx.DefaultSize, 0 )
        sizer.Add( self.feedbacktext, 0, wx.ALIGN_LEFT|wx.ALL, 5 )

        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )
        sizer.Add( [ 20, 20 ] , 0, wx.ALIGN_CENTER|wx.ALL, 5 )

        self.SetSizer(sizer)
        self.SetAutoLayout(True)
```

```
# callbacks from SimpleSCardAppEventObserver interface
def OnActivateCard( self, card ):
    SimpleSCardAppEventObserver.OnActivateCard( self, card )
    self.feedbacktext.SetLabel( 'Activated card: ' + `card` )

def OnActivateReader( self, reader ):
    SimpleSCardAppEventObserver.OnActivateReader( self, reader )
    self.feedbacktext.SetLabel( 'Activated reader: ' + `reader` )

def OnDeactivateCard( self, card ):
    SimpleSCardAppEventObserver.OnDeactivateCard( self, card )
    self.feedbacktext.SetLabel( 'Deactivated card: ' + `card` )

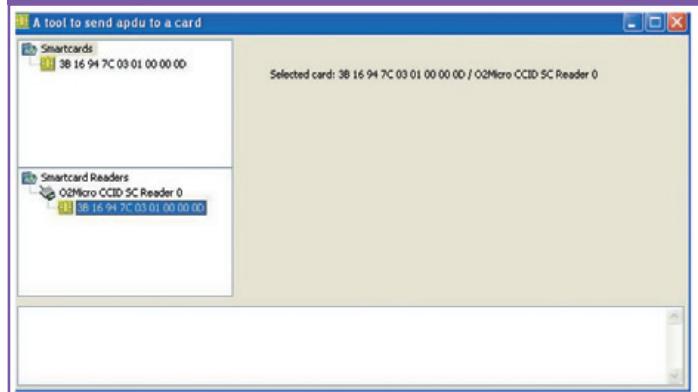
def OnSelectCard( self, card ):
    SimpleSCardAppEventObserver.OnSelectCard( self, card )
    self.feedbacktext.SetLabel( 'Selected card: ' + `card` )

def OnSelectReader( self, reader ):
    SimpleSCardAppEventObserver.OnSelectReader( self, reader )
    self.feedbacktext.SetLabel( 'Selected reader: ' + `reader` )
```

Les méthodes de l'interface **SimpleSCardObserver** sont appelées par le framework wxPython de **pyscard** lorsqu'une carte est activée ou sélectionnée dans les arbres de carte ou de lecteur. Le **wx.Panel** de cet exemple est un panel qui affiche dans un champ texte le nom de la carte activée.

Certains outils comme wxDesigner permettent de générer des panneaux de contrôle de façon graphique.

Figure 1 : Application wxPython basée sur smartcard.wx.SimpleSCardApp.



11

Conclusion

Cet article a présenté les fonctionnalités de base du framework Python **pyscard** pour les cartes à microprocesseur, ainsi que les fonctionnalités de base de la bibliothèque cryptographique **pycrypto**. Ces bibliothèques permettent de développer facilement des applications pour les cartes, et le langage Python contient un certain nombre de fonctionnalités qui facilitent la gestion des APDU. Pyscard est bien adapté à la génération de scripts pour la personnalisation de cartes après émission par exemple ou pour d'autres applications graphiques.

AUTEUR : JEAN-DANIEL AUSSÉL

LIENS

- Site du groupe de travail PC/SC : www.pcscworkgroup.com
- Site de PC/SC Lite : pcslite.alioth.debian.org
- Site du pilote CCID ou ICCD : pcslite.alioth.debian.org/ccid.html
- Site du framework Java Open Card Framework : www.openscdp.org/ocf
- Site du livre Thinking in Python de Bruce Eckel: www.mindview.net/Books/TIPython



Sous les systèmes de type UNIX, la gestion des cartes à puce n'est généralement pas intégrée nativement. Nous détaillons dans cet article comment rajouter la prise en charge des cartes à puce à l'aide des résultats de fichiers issus du projet MUSCLE.

Une fois cette gestion établie, nous donnons les étapes nécessaires à la construction d'un environnement complet de développement d'applications JavaCard permettant de tester les programmes développés et de les mettre en oeuvre de manière virtuelle avec un simulateur de carte à puce, ou bien directement sur de vraies cartes Java. Cet environnement est ensuite utilisé sur un exemple simple de type HelloWorld.

Mise en place d'un environnement complet de développement d'applications carte pour systèmes UNIX

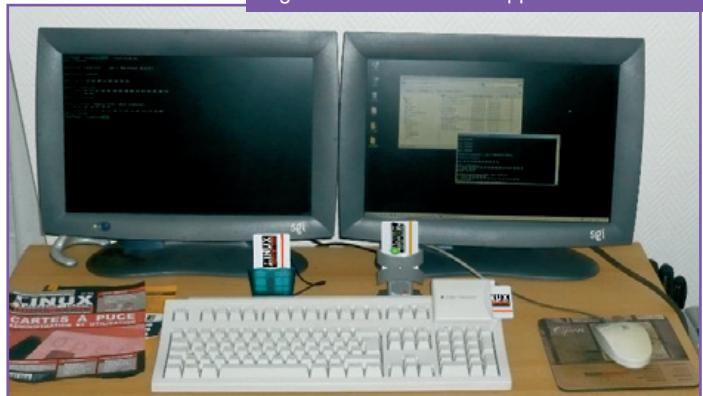
Auteur : Vincent Guyot

Une carte à puce peut être considérée comme étant un serveur et communique à travers le paradigme de communication client/serveur. En effet, la carte doit attendre d'être sollicitée pour fonctionner et émettre une réponse appropriée. Par conséquent, un programme carte à puce est une application répartie, avec une partie serveur s'exécutant dans la carte et une partie cliente s'exécutant sur le système où le lecteur de carte utilisé est branché. Ces deux parties peuvent chacune être écrites dans différents langages, en fonction de la carte utilisée et du middleware client choisi. Dans le cadre de cet article, s'exécutant dans une carte Java, le programme serveur est donc écrit en JavaCard, la partie cliente étant programmée en Java avec le middleware OCF (sur lequel nous reviendrons).

Ce SDK permet de programmer et de tester ensuite les applications carte développées, au choix, et ce, de manière transparente pour l'application cliente (c'est-à-dire sans modification), à travers un simulateur de carte ou bien une vraie carte Java. N'importe quelle carte Java peut être utilisée dans le cadre de ce SDK JavaCard, pourvu que l'on utilise un loader approprié pour installer/effacer dans la carte l'applet JavaCard que l'on a compilée. Les commandes dévolues à l'installation/effacement d'applets qui sont données dans cet article correspondent à la gestion de la carte Java Gemalto eGate CyberFlex 32K (actuellement disponible sur Internet).

Parmi les différents outils qui composent ce SDK, la plupart sont disponibles sous forme de sources à compiler ou bien de classes Java, ce qui leur assure une portabilité relativement bonne. Seul le simulateur du kit de développement JavaCard officiel n'est fourni par Sun Microsystems que sous forme exécutable. C'est par conséquent la seule partie du SDK à la portabilité toute relative, bien qu'il soit possible de l'utiliser à travers le réseau à partir d'un

Figure 1 : Le kit de développement en action



système distant. Les commandes données dans cet articles sont toutes destinées au système GNU/Linux.

Les scripts qui sont présentés dans cet article permettent l'utilisation du SDK JavaCard sous divers systèmes UNIX. Pour une utilisation autre [0]¹, se référer à mon article « Un SDK JavaCard générique », à paraître dans le numéro hors-série *Carte à puce* du magazine MISC en kiosque à la mi-novembre 2008.

La figure 1 montre le programme carte HelloWorld (explicite plus loin), qui s'exécute à la fois sur un pseudo-terminal GNU/Linux et en environnement graphique Windows Vista.

LA CARTE À PUCE À MICROPROCESSEUR JAVACARD

Roland Moreno, 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 « cartes ».

La programmation des cartes à puce est longtemps restée l'apanage des personnes 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^{1b}, 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^{1b} 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], annonçant des changements majeurs dans le langage, sans qu'il n'y ait encore de produits exploitant cette nouvelle mouture du langage.

Tout cela pour dire que c'est donc de cartes à puce à microprocesseur JavaCard dont il s'agit, dès lors que l'on parle abusivement de « cartes Java ».

1b Les sociétés leaders du monde de la carte à puce Axalto et Gemplus ont fusionné fin 2005 pour devenir Gemalto.

1

Yet another JavaCard SDK

Si vous avez une carte Java entre les mains, il y a 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 fabricant 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.

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 de la carte. 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 pourrait 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 *OpenCard Framework* [7] ou OCF, qui permet aux programmes Java de s'interfacer avec des cartes depuis Java 1.1, dont la portabilité et la robustesse sont

bien meilleures que celles du package récent de Java 1.6, mais surtout qui va nous permettre de tester notre application carte de manière transparente et indifférenciée avec un simulateur de carte ou bien un vraie carte Java.

2

Rajout au système de la gestion des cartes à puce

Plusieurs composants sont nécessaires à la gestion des cartes à puce et donc au bon fonctionnement du SDK JavaCard : le *middleware* système, les *drivers* des lecteurs de carte à puce que l'on veut utiliser, le middleware client nécessaire au programme s'adressant à la carte, et le *loader* d'applet pour installer/effacer des *applets* dans une carte Java.

Ces composants n'ont malheureusement pas évolué en même temps, et il est nécessaire d'en *patcher* certains pour les faire fonctionner ensemble. Les modifications nécessaires sont données sans justification, afin de ne pas sortir du cadre de l'article.

2.1

Installation du middleware système

Nous sommes ici entre nous, adorateurs de palmipèdes, amateurs de personnages fourchus, mangeurs de fugu, cultivateurs de pommes et autres randonneurs jardineux. Comme bien souvent, vous vous êtes rendu compte qu'il existait toujours plusieurs projets libres qui résolvent² un seul et même problème. C'est donc tout naturellement le cas également pour le problème de l'accès aux cartes à puce en environnement IEEE 1003/ISO-IEC 9945³.

Nous aurions pu choisir OpenCT [8] dont l'API très simple permet d'écrire rapidement des applications cartes en C. Si, historiquement, OpenCT n'implémentait que peu de drivers de lecteurs de cartes à puce, il permet d'utiliser aujourd'hui tous les lecteurs récents à la norme CCID.

Utilisé seul, il y a également OpenCard Framework qui permet de programmer des applications cartes en Java, pourvu que le lecteur de cartes utilisé possède un drivers OCF écrit en Java. Malheureusement, de tels drivers sont rares, et l'implémentation en Java interdit l'utilisation des lecteurs de cartes USB maintenant répandus. Toutefois, si l'on possède les lecteurs de cartes qui vont bien, c'est encore aujourd'hui la seule possibilité d'utiliser des cartes à puce sur certains systèmes aux architectures peu répandues, néanmoins pourvus de machine virtuelle Java.

Pour illustrer cet article, notre choix s'est porté sur l'utilisation de PC/SC Lite [9], fruit du projet MUSCLE [10], à l'API calquée sur celle de PC/SC [11] facilitant le portage d'applications cartes provenant de certains systèmes à fenêtres [0]. Ce choix permet de bénéficier des divers systèmes supportés par le projet MUSCLE, du grand nombre de drivers de lecteurs de cartes du projet MUSCLE et de programmer dans son langage de prédilection (voir article sur les wrappers PC/SC dans ce numéro).

Voici les différentes étapes nécessaires à l'installation de PC/SC Lite sur votre machine :

```
foo@bar:~$ wget https://alioth.debian.org/frs/download.php/2479/pcsc-lite-1.4.102.tar.bz2 --no-check-certificate
foo@bar:~$ tar xjvf pcsc-lite-1.4.102.tar.bz2
foo@bar:~$ cd pcsc-lite-1.4.102/
foo@bar:~/pcsc-lite-1.4.102$ ./configure -disable-libhal
```

REMARQUE

On désactive la prise en charge de la bibliothèque **libhal-dev** qui n'est pas souvent installée sur les systèmes et dont on peut se passer dans le cadre des manipulations de cet article.

Ensuite, à l'aide de votre éditeur de texte favori, modifiez le fichier **pcsc-lite-1.4.102/src/PCSC/ifdhandler.h** en remplaçant la ligne 19

```
#include <pcsclite.h>
```

par :

```
#include <PCSC/pcsclite.h>
```

Commentez ensuite les lignes 169 et 170 du même fichier :

```
/* RESPONSECODE IFDHControl(DWORD, DWORD, PUCHAR, DWORD, PUCHAR,
    DWORD, LPDWORD); */
```

Puis, modifiez le fichier **src/PCSC/pcsclite.h** en remplaçant la ligne 19

```
#include <wintypes.h>
```

par :

```
#include <PCSC/wintypes.h>
```

Enfin, on compile/installle de manière classique :

```
foo@bar:~/pcsc-lite-1.4.102$ make
foo@bar:~/pcsc-lite-1.4.102$ sudo make install
```

Si la commande **sudo** a du mal à passer sur votre configuration, exécutez ces commandes alternatives :

```
foo@bar:~/pcsc-lite-1.4.102$ su
root@bar:~/pcsc-lite-1.4.102# make install
```

Avant de pouvoir tester la bonne installation de PC/SC Lite, il reste encore à installer les drivers des lecteurs que vous allez utiliser (au moins un). Si le middleware permet le branchement à chaud des lecteurs USB et PCMCIA/PCCARD alors qu'il est actif, il faudra le relancer après avoir branché un lecteur PCMCIA/PCCARD, série ou PS/2 ces interfaces n'étant pas *hotplug-friendly*. Je n'ai malheureusement pas d'expérience à donner avec les lecteurs au nouveau format ExpressCard.

² Ceux qui « solutionnent » --> [1]

³ Merci à Richard Stallman pour avoir trouvé le bel acronyme POSIX qui remplace avantageusement ces barbarismes.

2.2

Installation des drivers des lecteurs de cartes à puce

On veut pouvoir utiliser plusieurs types de lecteurs de cartes à puce avec le SDK JavaCard : tous les lecteurs récents compatibles CCID, ainsi que les connecteurs qui ne permettent que d'utiliser (mais à quelle vitesse !) les cartes à puce à technologie eGate que l'on peut actuellement se procurer par Internet.

2.2.1

Installation du driver générique CCID

```
foo@bar:~$ wget https://alioth.debian.org/frs/download.php/2482/ccid-1.3.8.tar.bz2 --no-check-certificate
foo@bar:~$ tar xjvf ccid-1.3.8.tar.bz2
foo@bar:~$ cd ccid-1.3.8/
foo@bar:~/ccid-1.3.8$ ./configure
foo@bar:~/ccid-1.3.8$ make
foo@bar:~/ccid-1.3.8$ sudo make install
```

Si la commande **sudo** a du mal à passer sur votre configuration, exécutez ces commandes alternatives :

```
foo@bar:~/ccid-1.3.8$ su
root@bar:~/ccid-1.3.8# make install
```

Pour tester la bonne installation du driver CCID, exécutez en **root** le **daemon pcscd** (dans un autre terminal) avec :

```
root@bar:~# pcscd -f
```

Ensuite, le branchement d'un lecteur CCID et l'insertion d'une carte à puce sont des évènements qui doivent provoquer des *logs* à l'écran.

Si vous n'avez pas encore de lecteur CCID pour vos manipulations de cartes à puce, eBay est un bon endroit pour en acquérir contre quelques euros. Sinon, le TeoByXiring est sûrement le lecteur CCID neuf le plus abordable que l'on peut acheter à l'unité sur Internet.

2.2.2

Installation du driver eGate

```
foo@bar:~$ wget http://secure.netroedge.com/~phil/egate/ifd-egate-0.05.tar.gz
foo@bar:~$ tar xzvf ifd-egate-0.05.tar.gz
foo@bar:~$ cd ifd-egate-0.05/
```

Ensuite, à l'aide de l'éditeur de fichiers préféré, modifiez le fichier **ifdhandler.c** en remplaçant les trois premières lignes

```
#include <wintypes.h>
#include <pcsclite.h>
#include "ifdhandler.h"
```

par :

```
#include <PCSC/wintypes.h>
#include <PCSC/pcsclite.h>
#include <PCSC/ifdhandler.h>
```

Ensuite, on compile et installe par le classique **make/make install** :

```
foo@bar:~/ifd-egate-0.05$ make
foo@bar:~/ifd-egate-0.05$ sudo make install
```

Pour tester la bonne installation du driver eGate, exécutez sous le compte **root** le **daemon pcscd** (dans un autre terminal) avec :

```
root@bar:~# pcscd -f
```

Ensuite, l'insertion d'une carte à puce eGate dans son connecteur doit provoquer des logs à l'écran.

REMARQUE

L'insertion d'un connecteur seul n'affiche rien à l'écran, puisqu'il ne s'agit pas d'un vrai lecteur de cartes à puce, la gestion de la communication ayant été déportée dans les cartes à technologie eGate.

2.3

Installation du middleware client

Dans le SDK JavaCard, nous avons décidé d'écrire la partie cliente des applications cartes en Java. OpenCard Framework est un middleware qui permet d'écrire de tels programmes depuis le JDK 1.1, mais il ne propose que peu de drivers de lecteurs de cartes à puce. Heureusement, l'utilisation d'un bridge OCF - PC/SC Lite permet aux programmes OCF d'utiliser les nombreux lecteurs de cartes reconnus par PC/SC Lite.

2.3.1

Installation du JDK 1.6

À l'aide d'un *browser* web, récupérez le JDK 6 Update 7, sous la forme d'un fichier **.BIN** (74,85 Mo), à partir du site web de Java : <http://java.sun.com/javase/downloads/index.jsp>.

Ensuite, décompressez l'archive :

```
foo@bar:~$ sh jdk-6u7-linux-i586.bin
q
yes
foo@bar:~$
```

Le JDK 1.6 doit être installé à la racine du répertoire personnel pour que des commandes données plus loin puissent fonctionner.

Vérifiez la bonne installation du JDK 1.6 en exécutant la commande :

```
foo@bar:~$ ./jdk1.6.0_07/bin/java -version
java version '1.6.0_07'
Java(TM) SE Runtime Environment (build 1.6.0_07-b06)
Java HotSpot(TM) Client VM (build 10.0-b23, mixed mode, sharing)
foo@bar:~$
```

2.3.2

Installation du bridge OCF - PC/SC Lite

Le bridge OCF - PC/SC Lite doit être installé à la racine du répertoire personnel pour que des commandes données plus loin puissent fonctionner.

```
foo@bar:~$ wget http://www.musclecard.com/middleware/files/OCFPCSC1-0.0.1.tar.gz
foo@bar:~$ tar xzvf OCFPCSC1-0.0.1.tar.gz
foo@bar:~$ cd OCFPCSC1-0.0.1/
```

Il faut ensuite modifier le fichier **Makefile** :

ligne 2: remplacer **ld** par **g++**
ligne 4: **/home/foo/jdk1.6.0_07**
ligne 5: **PCSC_HDRS = -I/usr/local/include/PCSC**
ligne 6: **PCSC_LIBS = -L/usr/local/lib -lpcsc-lite**
ligne 7: remplacer **genunix** par **linux**
ligne 16: **cp -f *.so /usr/local/lib**
Puis, on compile le fichier **libOCFPCSC1.so** :

```
foo@bar:~/OCFPCSC1-0.0.1$ make
```

```
foo@bar:~$ tar xzvf globalplatform-5.0.0.tar.gz  
foo@bar:~$ cd globalplatform-5.0.0/  
foo@bar:~/globalplatform-5.0.0$ ./configure  
foo@bar:~/globalplatform-5.0.0$ make  
foo@bar:~/globalplatform-5.0.0$ sudo make install  
foo@bar:~/globalplatform-5.0.0$ sudo ldconfig
```

Si la commande **sudo** a du mal à passer sur votre configuration, exécutez ces commandes alternatives :

```
foo@bar:~/globalplatform-5.0.0$ su  
root@bar:~/globalplatform-5.0.0# make install  
root@bar:~/globalplatform-5.0.0# ldconfig
```

2.4 Installation du loader d'applets

Les mécanismes d'installation/effacement des applets dans les cartes Java sont régis par GlobalPlatform. Nous utilisons donc pour cela le programme GPShell, qui nécessite l'installation préalable de la bibliothèque GlobalPlatform sur laquelle il repose.

2.4.1 Installation de la bibliothèque GlobalPlatform

```
foo@bar:~$ wget http://downloads.sourceforge.net/globalplatform/  
globalplatform-5.0.0.tar.gz
```

2.4.2 Compilation de GPShell

GPShell doit être à la racine du répertoire personnel pour que des commandes données plus loin puissent fonctionner.

```
foo@bar:~$ wget http://downloads.sourceforge.net/globalplatform/gpshell-1.4.2.tar.gz  
foo@bar:~$ tar xzvf gpshell-1.4.2.tar.gz  
foo@bar:~$ cd gpshell-1.4.2  
foo@bar:~/gpshell-1.4.2$ ./configure  
foo@bar:~/gpshell-1.4.2$ make
```

3 Construction du SDK JavaCard

Plusieurs logiciels étant utilisés au sein du SDK JavaCard que nous construisons, il faut au préalable les récupérer. Fort heureusement, tous sont disponibles sur Internet, librement téléchargeables (ce qui ne veut pas forcément dire qu'ils sont libres).

Après avoir récupéré ces éléments constitutifs du SDK, nous donnons les étapes de sa construction, notamment les scripts appelant ces éléments de manière à pouvoir compiler et tester le plus simplement possible les applications que l'on va programmer. La hiérarchie des sous-répertoires donnée doit être respectée, au risque de devoir modifier les scripts fournis.

3.1 Récupération des différents morceaux

Pour que le SDK JavaCard soit pleinement opérationnel, nous avons besoin d'OpenCard Framework et de différentes versions du kit de développement JavaCard officiel de Sun Microsystems. Les possesseurs de cartes Java CyberFlex (dont la carte eGate CyberFlex 32K), au moins, auront également besoin d'une petite moulinette supplémentaire.

3.1.1 Téléchargement d'OpenCard Framework

L'utilisation de OCF nécessite à elle seule de récupérer quatre fichiers distincts :

- **BaseOCF.zip** [1467 Ko] ;
- **Reference_Impl.zip** [2369 Ko] ;

- **apduio-terminal-0.1.tar.gz** [333 Ko] ;
- **pcsc-wrapper-2.0.tar.gz** [411 Ko].

On les télécharge avec les commandes suivantes :

```
foo@bar:~$ wget http://www.openscdp.org/ocf/BaseOCF.zip  
foo@bar:~$ wget http://www.openscdp.org/ocf/Reference_Impl.zip  
foo@bar:~$ wget http://www.gemalto.com/techno/opencard/download/apduio-  
terminal-0.1.tar.gz  
foo@bar:~$ wget http://www.gemalto.com/techno/opencard/download/pcsc-  
wrapper-2.0.tar.gz
```

3.1.2 Téléchargement des JavaCard Development kits

Nous avons besoin de récupérer deux versions différentes du kit de développement JavaCard officiel, les versions 2.1.2 et 2.2.1 et, afin de pouvoir programmer des applications JavaCard pour les cartes Java munies de machines virtuelles, JavaCard 2.1 (très répandues) et 2.2 (pour les cartes Java plus récentes).

REMARQUE

On évite la version 2.2.2 du kit officiel qui ne permet pas d'utiliser le simulateur avec OCF aussi facilement que la version précédente, qu'on privilégie donc.

À l'aide de son navigateur web préféré, on se rend sur la page de téléchargement de JavaCard :

<http://java.sun.com/products/javacard/downloads/>

Deux fichiers sont donc à télécharger :

- **java_card_kit-2_1_2-solsparc.zip** [1,2 Mo] ;
- **java_card_kit-2_2_1-linux-dom.zip** [3,3 Mo].

REMARQUE

Au moment où le kit 2.1.2 a été développé, Sun Microsystems ne fournissait pas encore de logiciels pour les systèmes GNU/Linux. Ce n'est pas important pour ce qui nous intéresse. On préférera néanmoins la version 2.1.2 Solaris/SPARC pour rester dans le ton de l'article.

3.1.3 La moulinette facultative

Les possesseurs de cartes CyberFlex ont besoin de télécharger ce fichier :

- **captransf-1.5.zip** [113 Ko].

Malheureusement, il n'est pas téléchargeable directement. Il faudra en passer par un formulaire web et laisser une adresse email valide pour recevoir un lien temporaire où finalement le récupérer.

Le formulaire d'inscription est accessible à partir de la page <http://www.trusted-logic.com/down.php> où il faut choisir « CAP File Transformer V1.5 ».

4

Construction du SDK JavaCard

Une fois ces différents fichiers téléchargés, il faut les installer au sein d'une arborescence qui sera utilisée par un certain nombre de scripts que nous présenterons plus loin.

4.1 L'arborescence

4.1.1 Les répertoires

Nous allons construire le SDK de sorte qu'il soit indépendant de son emplacement dans l'arborescence du système, dans un répertoire que l'on nomme arbitrairement **JavaCardSDK**.

```
foo@bar:~$ mkdir JavaCardSDK
```

Dans ce répertoire **JavaCardSDK/**, on crée ces trois sous-répertoires :

- **JavaCardSDK/misc/** (les logiciels installés) ;
- **JavaCardSDK/out/** (les fichiers compilés) ;
- **JavaCardSDK/src/** (les sources des différents projets développés).

```
foo@bar:~$ cd JavaCardSDK/
foo@bar:~/JavaCardSDK$ mkdir misc out src
```

On commence par déplacer dans **JavaCardSDK/misc/** les répertoires du JDK 1.6 et de GPSHELL (précédemment compilé). Ensuite, on installe les archives en suivant (de manière arbitraire) l'ordre dans lequel elles ont été téléchargées :

```
foo@bar:~/JavaCardSDK$ cd misc/
foo@bar:~/JavaCardSDK/misc$ mv ~/jdk1.6.0_07/ .
foo@bar:~/JavaCardSDK/misc$ mv ~/gpshell-1.4.2/
foo@bar:~/JavaCardSDK/misc$ unzip ~/BaseOCF.zip
foo@bar:~/JavaCardSDK/misc$ unzip -o ~/Reference_Impl.zip
foo@bar:~/JavaCardSDK/misc$ mkdir apduio-terminal-0.1/
foo@bar:~/JavaCardSDK/misc$ cd apduio-terminal-0.1/
foo@bar:~/JavaCardSDK/misc/apduio-terminal-0.1$ tar xzvf ~/apduio-terminal-0.1.tar.gz
foo@bar:~/JavaCardSDK/misc/apduio-terminal-0.1$ cd ..
foo@bar:~/JavaCardSDK/misc$ mkdir pcsc-wrapper-2.0
foo@bar:~/JavaCardSDK/misc$ cd pcsc-wrapper-2.0/
foo@bar:~/JavaCardSDK/misc/pcsc-wrapper-2.0$ tar xzvf ~/pcsc-wrapper-2.0.tar.gz
```

```
foo@bar:~/JavaCardSDK/misc/pcsc-wrapper-2.0$ cd ..
foo@bar:~/JavaCardSDK/misc$ unzip ~/java_card_kit-2_1_2-solsparc.zip
foo@bar:~/JavaCardSDK/misc$ unzip ~/java_card_kit-2_2_1-linux-dom.zip
foo@bar:~/JavaCardSDK/misc$ mkdir captransf-1.5
foo@bar:~/JavaCardSDK/misc$ cd captransf-1.5/
foo@bar:~/JavaCardSDK/misc/captransf-1.5$ unzip ~/captransf-1.5.zip
foo@bar:~/JavaCardSDK/misc/captransf-1.5$ cd ..
```

On doit finalement obtenir cette arborescence :

- **JavaCardSDK/misc/apduio-terminal-0.1/**
- **JavaCardSDK/misc/captransf-1.5/**
- **JavaCardSDK/misc/gpshell-1.4.2/**
- **JavaCardSDK/misc/java_card_kit-2_1_2/**
- **JavaCardSDK/misc/java_card_kit-2_2_1/**
- **JavaCardSDK/misc/jdk1.6.0_07/**
- **JavaCardSDK/misc/OCF1.2/**
- **JavaCardSDK/misc/pcsc-wrapper-2.0/**

4.1.2 Différents fichiers nécessaires

On crée trois fichiers texte, servant au simulateur de carte Java, dans **JavaCardSDK/misc/** :

Header.scr contient les premières instructions servant à initialiser le simulateur de carte :

```
// Start the simulator
powerup;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 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 0x0B 0x0a 0xa0 0x0 0x0 0x0 0x62 0x03 0x1 0xc 0x6 0x1 0x7F;
```

Footer.scr contient la dernière instruction de l'initialisation du simulateur :

```
// Shut down the simulator
powerdown;
```

On copiera ensuite le bridge OCF - PC/SC Lite (qui a été précédemment compilé) à la racine du SDK :

```
foo@bar:~/JavaCardSDK$ cp ~/OCFPCSC1-0.0.1/libOCFPCSC1.so .
```

Pour finir, on va créer le fichier de configuration du programme client OCF, long de deux lignes uniquement. Dans **JavaCardSDK/**, on édite le fichier **opencard.properties** différemment en fonction de l'utilisation ou non d'un lecteur de carte à puce. Si aucun lecteur n'est utilisé, seul le simulateur sera utilisable. Fichier **opencard.properties** dans le cas d'un lecteur de carte à puce :

```
OpenCard.services = opencard.opt.util.PassThruCardServiceFactory
OpenCard.terminals = com.ibm.opencard.terminal.pcsc10.Pcsc10CardTerminalFactory com.gemplus.opencard.terminal.apduio.ApduIOCardTerminalFactory|mySim|Socket|localhost:9025
```

Fichier **opencard.properties** sans lecteur de carte à puce :

```
OpenCard.services = opencard.opt.util.PassThruCardServiceFactory
OpenCard.terminals = com.gemplus.opencard.terminal.apduio.ApduIOCardTerminalFactory|mySim|Socket|localhost:9025
```

Dans les deux cas, on note la possibilité de préciser au programme client OCF sur quelle machine dans le réseau est installé le simulateur de carte. Par défaut, il est sur la machine courante (**localhost**) au port TCP 9025. Pensez à configurer votre firewall local le cas échéant.

4.2 Les scripts

Tous se trouvent dans **JavaCardSDK/**. Ces scripts permettent d'interagir avec le SDK JavaCard. Une fois ces fichiers installés, la racine du SDK doit être :

```
JavaCardSDK/card-deleteApplet.sh
JavaCardSDK/card-installApplet.sh
JavaCardSDK/cleanAll.sh
JavaCardSDK/libOCFPCSC1.so
JavaCardSDK/makeApplet.sh
JavaCardSDK/makeClient.sh
JavaCardSDK/misc/
JavaCardSDK/opencard.properties
JavaCardSDK/out/
JavaCardSDK/runClient.sh
JavaCardSDK/runSimulator.sh
JavaCardSDK/setenv.sh
JavaCardSDK/simulator-init.sh
JavaCardSDK/simulator-installApplet.sh
JavaCardSDK/src/
```

Afin d'éviter une saisie fastidieuse, le code des scripts est disponible sur le site web du magazine.

On lancera les différents scripts par :

```
foo@bar:~/JavaCardSDK$ sh fooScript.sh
```

Ou bien directement, après les avoir rendus exécutables :

```
foo@bar:~/JavaCardSDK$ chmod +x *.sh
foo@bar:~/JavaCardSDK$ ./foo.script.sh
```

4.2.1 setenv.sh

Ce script contient les données de configuration d'un projet, à savoir l'emplacement des applications nécessaires, le nom des sous-répertoires utilisés, les numéros AID des applets JavaCard ou encore les données techniques ayant rapport au type de la carte Java utilisée. Il est exécuté par tous les autres scripts pour les configurer automatiquement. On notera la variable **JC_EXP** qui permet 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). On remarque également que les numéros AID des applets et packages JavaCard apparaissent deux fois, écrits différemment, en fonction des outils qui les utilisent. 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/**.

```
#!/bin/sh
export PROJECT=mini
export DIR=.
export OUT=$DIR/out
export SRC=$DIR/src
export MISC=$DIR/misc
export CAPTRANSF=$MISC/captransf-1.5
export GPSHELL=$MISC/gpshell-1.4.2
export JC21_HOME=$MISC/java_card_kit-2_1_2
export JC22_HOME=$MISC/java_card_kit-2_2_1
export JAVA_HOME=$MISC/jdk1.6.0_07
export OCF_HOME=$MISC/OCF1.2
export PCSC_WRAPPER=$MISC/pcsc-wrapper-2.0
export APDUIO_TERM=$MISC/apduio-terminal-0.1
export JC21_API=$JC21_HOME/lib/api21.jar
export JC22_API=$JC22_HOME/lib/api.jar
export JC21_EXP=$JC21_HOME/api21_export_files
export JC22_EXP=$JC22_HOME/api_export_files
# uncomment the right line to compile JC2.1 or JC2.2 applet:
export JC_EXP=$JC21_EXP
#export JC_EXP=$JC22_EXP
export PKGCLIENT=client
export CLIENT=TheClient
export PKGAPPLET=applet
export APPLET=TheApplet
export AIDPACKAGE=0xA0:0x00:0x00:0x00:0x62:0x03:0x01:0x0C:0x06
export PACKAGEAID=A00000006203010C06
export AIDAPPLET=0xA0:0x00:0x00:0x00:0x62:0x03:0x01:0x0C:0x06:0x01
export APPLETAID=A00000006203010C0601
# section specific to the card eGate CyberFlex 32K
export CARDSECURITYDOMAIN=A00000003000000
export CARDSECURITYDOMAIN2=A000000030000
export CARDKEY=404142434445464748494A4B4C4D4E4F
export PKGVERSION=1.0
export SIMUSCRIPT=script
```

4.2.2 cleanAll.sh

Comme son nom l'indique, ce script permet d'effacer tous les fichiers qui ont été compilés.

```
#!/bin/sh
source ./setenv.sh
# delete temporary files
rm -f $DIR/*
rm -f $SRC/$PROJECT/$PKGAPPLET/*
rm -f $SRC/$PROJECT/$PKGCLIENT/*
```

```
# delete client
rm -rf $OUT/$PROJECT/$PKGCLIENT

# delete applet
rm -rf $OUT/$PROJECT/$PKGAPPLET
rm -rf $OUT/$PROJECT/$PKGAPPLET.cap

# delete simulator data
rm $OUT/$PROJECT/$APPLET.scr $OUT/$PROJECT/script.scr $OUT/$PROJECT/eeprom
```

4.2.3 makeApplet.sh

Ce fichier lance la compilation de l'applet et met en place les fichiers nécessaires à son utilisation par le simulateur de carte. Si l'on veut l'utiliser dans une vraie carte Java, il suffit d'installer dans la carte le fichier **.CAP** compilé (en fonction du loader utilisé, il peut être nécessaire de renommer le fichier **.CAP** en **.JAR**).

```
#!/bin/sh
source ./setenv.sh
echo Compilation...
echo $JC22_API
$JAVA_HOME/bin/javac -source 1.2 -target 1.2 -g -classpath $JC22_API -d $OUT/$PROJECT/ $SRC/$PROJECT/$PKGAPPLET/$APPLET.java
echo $APPLET.class compiled: OK
echo .

echo Converting...
$JAVA_HOME/bin/java -classpath $JC22_HOME/lib/converter.jar:$JC22_HOME/lib/offcardVerifier.jar com.sun.javacard.converter.Converter -nobanner -classdir $OUT/$PROJECT/ -exportpath $JC_EXP -d $OUT/$PROJECT/ -out EXP
JCA CAP -applet $AIDAPPLET $PKGAPPLET.$APPLET $PKGAPPLET $AIDPACKAGE
$PKGVERSION
echo $APPLET.cap converted: OK
echo .
cp $OUT/$PROJECT/$PKGAPPLET/javocard/$PKGAPPLET.cap $OUT/$PROJECT/
echo Scripting...
$JAVA_HOME/bin/java -classpath $JC22_HOME/lib/scriptgen.jar com.sun.javacard.scriptgen.Main -nobanner -o $OUT/$PROJECT/$APPLET.scr $OUT/$PROJECT/$PKGAPPLET/javocard/$PKGAPPLET.cap
echo $APPLET.scr created: OK
echo .

echo Completing script...
cat $MISC/Header.scr $OUT/$PROJECT/$APPLET.scr $MISC/Install.scr $MISC/Footer.scr > $OUT/$PROJECT/$SIMUSCRIPT.scr
rm -f $OUT/$PROJECT/$APPLET.scr
echo $SIMUSCRIPT.scr created: OK
echo .
```

4.2.4 makeClient.sh

Ce script lance la compilation du programme client (Java/OCF) qui va échanger des APDU avec le programme serveur (JavaCard) situé dans la carte, qu'elle soit réelle ou simulée.

```
#!/bin/sh
source ./setenv.sh
# opencard.core.*
export CLASSES=$CLASSES:$OCF_HOME/lib/base-core.jar
# opencard.opt.util
export CLASSES=$CLASSES:$OCF_HOME/lib/base-opt.jar
echo Compilation...
$JAVA_HOME/bin/javac -classpath $CLASSES -g -d $OUT/$PROJECT $SRC/$PROJECT/$PKGCLIENT/$CLIENT.java
echo $CLIENT.class compiled: OK
```

4.2.5 runClient.sh

Ce script lance l'exécution du client compilé et le place en attente, soit de l'insertion d'une vraie carte Java, soit du lancement du simulateur de carte Java.

```
#!/bin/sh
source ./setenv.sh
export CLASSPATH=$JC22_HOME/lib/apduio.jar
export CLASSPATH=$CLASSPATH:$OCF_HOME/lib/base-core.jar
export CLASSPATH=$CLASSPATH:$OCF_HOME/lib/base-opt.jar
export CLASSPATH=$CLASSPATH:$PCSC_WRAPPER/lib/pcsc-wrapper-2.0.jar
export CLASSPATH=$CLASSPATH:$APDUIO_TERM/lib/apduio-terminal-0.1.jar
$JAVA_HOME/bin/java -Djava.library.path=. -classpath $OUT/$PROJECT/:$CLASSPATH $PKGCLIENT.$CLIENT
```

4.2.6 runSimulator.sh

Ce script lance l'exécution du simulateur de carte, préalablement initialisé (voir scripts **simulator-init.sh** et **simulator-installApplet.sh**).

```
#!/bin/sh
source ./setenv.sh
$JC22_HOME/bin/cref -nobanner -nomeminfo -i $OUT/$PROJECT/eeprom
```

4.2.7 simulator-init.sh

Ce script place le simulateur en attente d'initialisation. Il faut alors lancer **simulator-installApplet.sh** pour réellement l'initialiser.

```
#!/bin/sh
source ./setenv.sh
$JC22_HOME/bin/cref -nomeminfo -nobanner -o $OUT/$PROJECT/eeprom
```

4.2.8 simulator-installApplet.sh

Ce script installe l'applet compilée dans le simulateur, placé au préalable dans un état d'attente suite à l'exécution de **simulator-init.sh**.

```
#!/bin/sh
source ./setenv.sh
# apdutool
$JAVA_HOME/bin/java -noverify -classpath $JC22_HOME/lib/apdutool.jar:$JC22_HOME/lib/apduio.jar com.sun.javacard.apdutool.Main -nobanner $OUT/$PROJECT/$SIMUSCRIPT.scr
```

4.2.9 card-installApplet.sh

Ce script installe dans la carte (définie dans **setenv.sh**) le fichier **.CAP** compilé qui contient l'applet.

Remarque : 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.sh**. 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.sh**.

ATTENTION

Ce script ne fonctionne en l'état qu'avec la carte eGate CyberFlex 32K. Ne l'utilisez pas avec d'autres cartes sous peine de les bloquer. Les possesseurs d'autres cartes Java n'ont qu'à installer dans leur carte le fichier **.CAP** compilé (après l'avoir éventuellement renommé en **.JAR**, en fonction du loader utilisé).

```
#!/bin/sh

# this file is specific to the card eGate CyberFlex 32K
# do not use it with another Java card, you would block it!
source ./setenv.sh

$JAVA_HOME/bin/java -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
LD_LIBRARY_PATH=. && $GPSHELL/gpshell < $OUT/config.txt
rm -f $OUT/config.txt
```

4.2.10 card-deleteApplet.sh

Ce script permet d'effacer l'applet précédemment chargé dans la carte, tout du moins l'applet dont le numéro AID est dans le fichier **setenv.sh** courant. Pour peu que les cartes aient les mêmes clefs, ce script est utilisable sur les cartes Java compatibles OpenPlatform 2.0.1, et pas seulement avec l'eGate CyberFlex 32K comme le script **card-install.sh**.

```
#!/bin/sh

source ./setenv.sh

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

LD_LIBRARY_PATH=. && $GPSHELL/gpshell < $OUT/config.txt
rm -f $OUT/config.txt
```

5

Déroulement du projet HelloWorld

Pour tester ce SDK JavaCard, nous allons créer un projet nommé « mini », qui comprendra les deux sources de l'applet et du client d'une application minimale mettant en œuvre une carte Java (simulée ou réelle).

On crée un sous-répertoire **JavaCardSDK/src/mini/**, deux sous-répertoires **JavaCardSDK/src/mini/applet/** et **JavaCardSDK/src/mini/client/**, qui contiendront respectivement la source de l'applet (**TheApplet.java**, en JavaCard) et du client (**TheClient.java**, en Java/OCF), et enfin un sous-répertoire **JavaCardSDK/out/mini/** qui contiendra les programmes compilés du projet « mini ».

```
foo@bar:~/JavaCardSDK$ mkdir src/mini src/mini/applet src/mini/client out/mini
```

5.1

TheApplet.java

Cette applet est minimalistique. 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 le programme client. 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]);
}
```

5.2

TheClient.java

Ce programme Java/OCF est très simple. Il attend l'événement 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)0xC0, (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.hexify( card.getCardID().getATR() ) + "\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();
}
}

```

5.3

Compilation et test du projet « mini »

Afin de s'assurer de l'état initial du SDK, on commence par lancer **cleanAll.sh** qui réinitialise le SDK en effaçant les fichiers déjà compilés.

Avant toute chose, l'utilisation ultérieure d'une vraie carte impose de lancer le daemon **pcscd** sous le compte root après

avoir branché le lecteur de carte à puce. On peut sauter cette étape si seul le simulateur est utilisé.

```
root@bar:~# pcscd -f
```

Ensuite, on lance **makeApplet.sh** et **makeClient.sh**.

À ce moment, le fichier applet au format **.CAP** est prêt à être chargé dans une carte Java ou dans le simulateur de carte.

REMARQUE

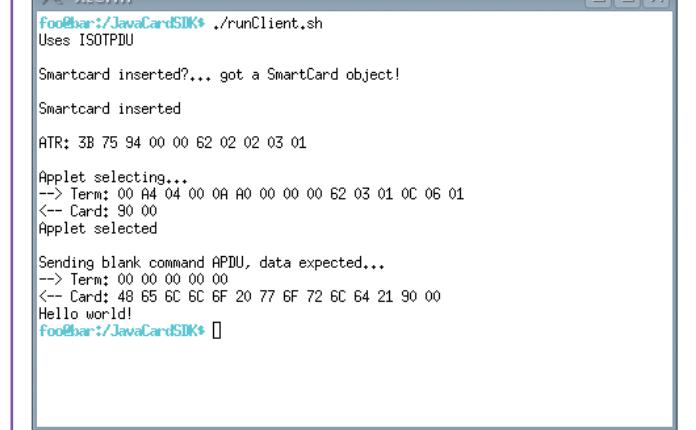
De manière étrange, certains loaders de cartes Java n'accepteront d'installer le fichier CAP qu'une fois renommé en JAR.

Pour utiliser le simulateur de carte, il faut l'initialiser en exécutant **simulator-init.sh**, puis **simulator-installApplet.sh**.

Si l'on souhaite utiliser une vraie carte Java, alors il faut exécuter **card-installApplet.sh** ou bien utiliser un loader particulier pour charger dans la carte le fichier **.CAP**, copié à la racine du projet dans **JavaCardSDK/out/\$PROJECT/**.

Enfin, on insère la carte Java réelle dans le lecteur ou bien on lance **runSimulator.sh**, et, enfin, on lance **runClient.sh**.

On retrouve bien sur la figure 2 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 comme prévu.

REMARQUE

Lors du développement d'une application carte, si vous souhaitez la tester avec le simulateur, veillez à ce qu'il n'y ait aucune carte à puce d'insérée dans un lecteur. En effet, si un même programme client peut s'adresser de manière indifférenciée à une eGate Cyberflex ou à un simulateur de carte, la vraie carte a la priorité sur le simulateur.

6

Conclusion

À partir d'un système sans aucune prise en charge des cartes à puce, nous avons installé tous les composants système nécessaires à l'utilisation de cartes.

Ce SDK est construit à partir de différents logiciels librement téléchargeables. Avec lui, nous sommes en mesure de prototyper une application carte Java avec la possibilité de tester le programme serveur sur simulateur, en cours de développement, et de tester « en vrai » sur une carte Java réelle sans avoir à modifier le code du programme 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 transférer ce répertoire sur un ordinateur de même type (OS/archi) pour pouvoir ensuite utiliser le SDK immédiatement.

Remarques : Pour peu que l'on fonctionne sous un système où on puisse exécuter le simulateur de carte intégré à cet environnement de développement, il n'est pas nécessaire de posséder une vraie carte Java pour pouvoir réaliser de vraies applications carte.

Pour réduire drastiquement la taille du répertoire contenant l'ensemble du SDK, on peut utiliser une version antérieure du JDK, toujours plus compacte.

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.

AUTEUR : VINCENT GUYOT



Professeur ingénieur à l'ESIEA, tout particulièrement au sein des mastères spécialisés « Sécurité de l'Information et des systèmes » et « Network and information security », et chercheur collaborant avec le Laboratoire d'informatique de Paris 6. Randonneur depuis 1994, mais pas au sens où sa femme l'entend...

LIENS

- [0] <http://www.microsoft.com/>
- [1] <http://www.basiccard.com/>
- [2] <http://java.sun.com/>
- [3] <http://www.sun.com/software/opensource/java/>
- [4] <http://java.sun.com/products/javacard/>
- [5] <http://java.sun.com/products/javacard/3.0/>
- [6] http://en.wikipedia.org/wiki/ISO_7816
- [7] <http://www.openscdp.org/ocf/>
- [8] <http://www.opensc-project.org/openct/>
- [9] <http://pcsclite.alioth.debian.org/>
- [10] <http://www.musclecard.com/>
- [11] <http://www.pcscworkgroup.com/>



PC/SC (ou WinScard) est une API en langage C. Il existe un grand nombre de wrappers pour utiliser l'API PC/SC depuis son langage de prédilection. Cet article présente le wrapper pour Perl et jette un œil à ceux pour Python, Caml, Prolog, Ruby et Java.

Wrappers PC/SC ou comment se passer du langage C pour accéder aux cartes à puce

Auteur : Ludovic Rousseau

1

PC/SC en Perl

1.1

Historique

Le projet PCSC/Perl [[pcsc-perl](#)] a été commencé en 2001 par Lionel Victor. D'ailleurs, donnons la parole à Lionel pour qu'il présente lui-même le projet :

« Le besoin initial était de dialoguer avec des cartes à puce pour faire des évaluations sécuritaires. Ce que nous cherchions à faire était très varié et changeait d'une carte à l'autre.

À cette époque, nous écrivions souvent de petits programmes en C pour scanner les fichiers d'une carte SIM, faire des recherches exhaustives, des cryptanalyses... Certaines personnes de notre groupe utilisaient des pilotes ou des API propriétaires, puis PCSC-lite a débarqué sous Linux.

On a rapidement trouvé que la possibilité d'avoir un code source compatible entre Linux et Windows était très intéressante, mais nous étions vite bloqués pour faire des choses complexes. En outre, le C n'était pas le langage le plus adapté à notre besoin, car il faut gérer la mémoire et beaucoup d'autres choses. Nous ne cherchions pas la robustesse ou la performance, mais au contraire la souplesse et la richesse d'un langage de script. En bref, nous ne voulions pas passer notre temps à faire des `malloc()` et des `free()`, mais plutôt nous concentrer sur le code métier : les attaques !

Perl possédait déjà une bonne API cryptographique, il était simple et riche. Nous nous sommes lancés dans l'élaboration de ce wrapper qui a été notre dernier code en C !

Le premier programme a été un scanner de fichiers pour cartes SIM. On a aussi fait des attaques sur les algorithmes GSM COMP128 grâce à l'API cryptographique de Perl.

C'était très souple, le code reflétait exactement l'esprit de chaque attaque et rien d'autre : exactement ce qu'on voulait.

Un autre avantage de Perl est de permettre l'utilisation d'une interface graphique Gtk/Perl, (presque) compatible avec Windows... C'est d'ailleurs comme cela qu'est né gscriptor (un petit utilitaire graphique souvent fourni avec PCSC/Perl).

PCSC/Perl n'était pas un projet ambitieux techniquement et je n'avais jamais prévu d'en faire quelque chose de vraiment utile. On l'avait simplement mis sur CPAN pour une communauté qu'on imaginait assez restreinte : en effet, qui pouvait bien vouloir parler à une carte à puce ? Un jour, j'ai vu qu'un paquet était disponible pour la distribution Mandriva chez moi... sous Debian aussi... quelle surprise !

C'est d'autant plus enthousiasmant que la plupart des projets plus pointus techniquement que j'ai réalisés pour mon travail ont fini au placard... Avec les logiciels libres, au contraire, rien ne se perd : il y a toujours quelqu'un à qui notre travail peut servir...

C'est peut-être ça le développement durable ! »

Le logiciel PCSC/Perl est référencé sur le site CPAN [[CPAN](#)]. Les tests automatiques effectués par les testeurs CPAN sont tous en échec. C'est simplement dû au fait qu'il faut installer pcsc-lite avant de compiler PCSC/Perl. Je suis preneur d'une solution à ce problème si quelqu'un a une idée.

1.2 Installation

Pour une distribution Debian, Ubuntu ou autre distribution dérivée, vous pouvez installer le paquet **libpcsc-perl**, ce qui va automatiquement installer **libpcslite1** et autres dépendances. Si ce n'est pas déjà fait, vous devez aussi installer le paquet **pcscd** qui va alors installer le paquet **libccid** qui est le pilote pour les lecteurs USB respectant la spécification CCID. Si votre lecteur nécessite un autre pilote il faudra l'installer.

Si votre distribution Linux ne contient pas déjà un paquet pour PCSC/Perl, vous pouvez récupérer l'archive **pcsc-perl-1.4.7.tar.gz** sur [[pcsc-perl](#)] et l'installer avec les commandes suivantes.

```
pcsc-perl-1.4.7$ perl Makefile.PL
osname: linux
LDFLAGS:
INC: `pkg-config --cflags libpcslite'
Checking if your kit is complete...
Looks good
Writing Makefile for Chipcard::PCSC::Card
Writing Makefile for Chipcard::PCSC
pcsc-perl-1.4.7$ make
[...]
pcsc-perl-1.4.7$ make test
[...]
pcsc-perl-1.4.7$ make install
[...]
```

1.3 Description de l'API

Les appels PC/SC sont disponibles directement en Perl sans grandes modifications. L'API propose deux classes **Chipcard::**:

PCSC et **Chipcard::PCSC::Card**. Le préfixe **Chipcard::** vient du fait qu'il existe aussi un module **Chipcard::CTAPI** permettant d'utiliser l'API CTAPI plutôt que PC/SC pour interagir avec des lecteurs et des cartes à puce. Le module **Chipcard::CTAPI** utilise directement **libtowitoko** qui est un pilote pour les (vieux) lecteurs Towitoko. N'ayant pas de lecteurs Towitoko, je ne peux pas utiliser **Chipcard::CTAPI**.

1.3.1 Module Chipcard::PCSC

Cette classe permet d'obtenir la liste des lecteurs et d'attendre un événement carte. Je ne vais pas décrire les méthodes une par une : elles sont directement liées à leurs versions équivalentes l'API WinSCard en langage C. Une différence notable est que les méthodes Perl utilisent des paramètres optionnels ayant une valeur par défaut raisonnable. La documentation complète est disponible sur la page du projet sur le site CPAN [[CPAN](#)].

1.3.1.1 Exemple

```
#!/usr/bin/perl -w

use Chipcard::PCSC;

# crée un nouvel objet
$hContext = new Chipcard::PCSC();
die ("Can't create the PCSC object: $Chipcard::PCSC::errno\n")
unless defined $hContext;

# récupère la liste des lecteurs
@ReadersList = $hContext->ListReaders();
die ("Can't get readers' list: $Chipcard::PCSC::errno\n")
unless defined $ReadersList[0];

# affiche la liste des lecteurs
print "$_\n" foreach @ReadersList;

# crée une liste pour GetStatusChange avec uniquement le premier lecteur
@readers_states = ({'reader_name'=>ReadersList[0]});

# initialise l'état courant
>StatusResult = $hContext->GetStatusChange(\@readers_states);
$readers_states[0]->{current_state} = $readers_states[0]->{event_state};

# attend un changement : insertion ou retrait carte
>StatusResult = $hContext->GetStatusChange(\@readers_states);
```

La sortie est alors :

```
Gemplus GemPC Twin 00 00
```

Notez que le programme est (ou devrait être) bloqué sur le deuxième appel à **GetStatusChange** jusqu'à ce qu'un mouvement carte (insertion ou retrait) soit détecté. C'est donc tout à fait normal que le programme ne rende pas la main avant d'avoir inséré une carte ou enlevé la carte présente suivant le cas.

La classe **Chipcard::PCSC** fournit aussi deux méthodes utilitaires pour convertir une APDU d'un codage texte dans le format interne utilisé par **Chipcard::PCSC::Card** et réciproquement.

1.3.2 Module Chipcard::PCSC::Card

Cette classe permet de manipuler les cartes et principalement de se connecter à une carte et d'échanger des APDU. Sont aussi présentes les méthodes **BeginTransaction()**, **EndTransaction()** ainsi que **Control()**.

1.3.2.1 Exemple

```
#!/usr/bin/perl -w

use Chipcard::PCSC;

# crée un nouvel objet
$hContext = new Chipcard::PCSC();
die ("Can't create the PCSC object: $Chipcard::PCSC::errno\n")
    unless defined $hContext;

# récupère la liste des lecteurs
@ReadersList = $hContext->ListReaders();
die ("Can't get readers' list: $Chipcard::PCSC::errno\n")
    unless defined $ReadersList[0];

# se connecte au premier lecteur
$hCard = new Chipcard::PCSC::Card($hContext, $ReadersList[0]);
die ("Can't connect: $Chipcard::PCSC::errno\n")
    unless defined $hCard;

# envoie l'APDU Select Applet
$cmd = Chipcard::PCSC::ascii_to_array("00 A4 04 00 0A A0 00 00 00 62 03 01
0C 06 01");
$RecvData = $hCard->Transmit($cmd);
die ("Can't transmit data: $Chipcard::PCSC::errno") unless defined
$RecvData;
print Chipcard::PCSC::array_to_ascii($RecvData)."\n";

# envoie l'APDU de test
$cmd = Chipcard::PCSC::ascii_to_array("00 00 00 00");
$RecvData = $hCard->Transmit($cmd);
die ("Can't transmit data: $Chipcard::PCSC::errno") unless defined
$RecvData;
print Chipcard::PCSC::array_to_ascii($RecvData)."\n";

$hCard->Disconnect();
```

La sortie est :

```
90 00
48 65 6C 6C 6F 20 77 6F 72 6C 64 21 90 00
```

Tous les programmes d'exemples de cet article utilisent l'applet « *Hello world!* » présentée par Vincent Guyot dans ce magazine. On peut vérifier que la sortie est correcte et correspond au résultat attendu.

```
$ echo -n 'Hello world!' | xxd -g 1
000000: 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 Hello world!
```

Remarquez que la sortie contient les 12 octets envoyés par la carte, ainsi que les 2 octets correspondant au mot d'état (*Status Word*). La valeur **0x90 0x00** signifie : exécution normale. La commande s'est correctement déroulée, sans erreur.

1.3.2.2 TransmitWithCheck

La méthode **TransmitWithCheck()** permet d'avoir une méthode de plus haut niveau que **Transmit()** pour envoyer une APDU et récupérer les résultats. Dans le code précédent, remplacez les deux derniers blocs de code par :

```
# envoie l'APDU Select Applet
($sw, $RecvData) = $hCard->TransmitWithCheck("00 A4 04 00 0A A0 00 00 00
62 03 01 0C 06 01", "90 00");
die ("Can't transmit data: $Chipcard::PCSC::errno") unless defined $sw;
print $RecvData."\n";
print Chipcard::PCSC::Card::ISO7816Error($sw) . " ($sw)\n";

# envoie l'APDU de test
($sw, $RecvData) = $hCard->TransmitWithCheck("00 00 00 00", "90 00");
die ("Can't transmit data: $Chipcard::PCSC::errno") unless defined $sw;
print $RecvData."\n";
print map { chr hex $_ } split ' ', $RecvData;
print "\n";
print Chipcard::PCSC::Card::ISO7816Error($sw) . " ($sw)\n";
```

La méthode **TransmitWithCheck()** renvoie une paire de paramètres. Elle fait elle-même le découpage entre données renvoyées et mot d'état. Elle prend en plus un paramètre qui est le mot d'état attendu. Si le mot d'état de retour n'est pas celui indiqué, la méthode retourne en erreur et il est facile de quitter le programme avec un message d'erreur approprié.

Nous découvrons aussi la méthode **Chipcard::PCSC::Card::ISO7816Error()** qui permet de convertir un mot d'état en format brut (numérique) en format texte plus lisible par un humain.

La sortie est (avec une première ligne vide, puisque la commande Select Applet ne renvoie pas de données mais juste un code de retour) :

```
Normal processing. (90 00)
48 65 6C 6C 6F 20 77 6F 72 6C 64 21
Hello world!
Normal processing. (90 00)
```

1.4 Exemples d'utilisation en vrai

1.4.1 scriptor

scriptor est un outil inclus dans pcsc-tools [**pcsc-tools**] (paquet Debian **pcsc-tools**) qui permet d'envoyer, très simplement, des APDU en ligne de commande. **scriptor** a été développé à l'origine comme un exemple d'utilisation de PCSC/Perl. **scriptor** peut être utilisé par n'importe qui sans connaissance de Perl. Il suffit de connaître les commandes à envoyer à la carte.

L'outil peut être utilisé de façon interactive ou en script.

En ligne de commande interactive (utilisez [Ctrl-d] pour quitter) :

```
$ scriptor
No reader given: using Gemplus GemPC Twin 00 00
Using T=1 protocol
Reading commands from STDIN
00 A4 04 00 0A A0 00 00 00 62 03 01 0C 06 01
> 00 A4 04 00 0A A0 00 00 00 62 03 01 0C 06 01
< 90 00 : Normal processing.
00 00 00 00
> 00 00 00 00
< 48 65 6C 6C 6F 20 77 6F 72 6C 64 21 90 00 : Normal processing.
```

En fichier script :

```
$ cat essai.scriptor
#!/usr/bin/env scriptor
reset
# Select applet
00 A4 04 00 0A A0 00 00 00 62 03 01 0C 06 01
# APDU de test
00 00 00 00

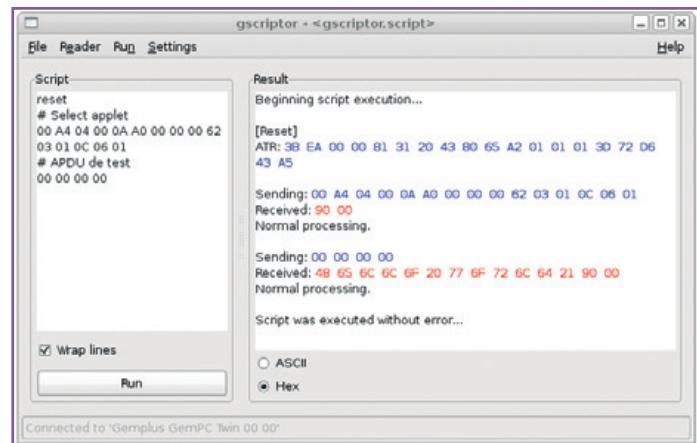
$ ./essai.scriptor
No reader given: using Gemplus GemPC Twin 00 00
Using T=1 protocol
Using given file: ./essai.scriptor
#!/usr/bin/env scriptor
reset
> RESET
< OK: 3B EA 00 00 81 31 20 43 80 65 A2 01 01 01 3D 72 D6 43 A5
# Select applet
00 A4 04 00 0A A0 00 00 00 62 03 01 0C 06 01
> 00 A4 04 00 0A A0 00 00 00 62 03 01 0C 06 01
< 90 00 : Normal processing.
# APDU de test
00 00 00 00
> 00 00 00 00
< 48 65 6C 6C 6F 20 77 6F 72 6C 64 21 90 00 : Normal processing.
```

Les lignes vides ou commençant par # sont ignorées. **reset** est un mot clé spécial faisant un reset de la carte. C'est très pratique pour toujours partir avec une carte dans un état connu. Les autres lignes sont considérées comme des commandes APDU à envoyer à la carte.

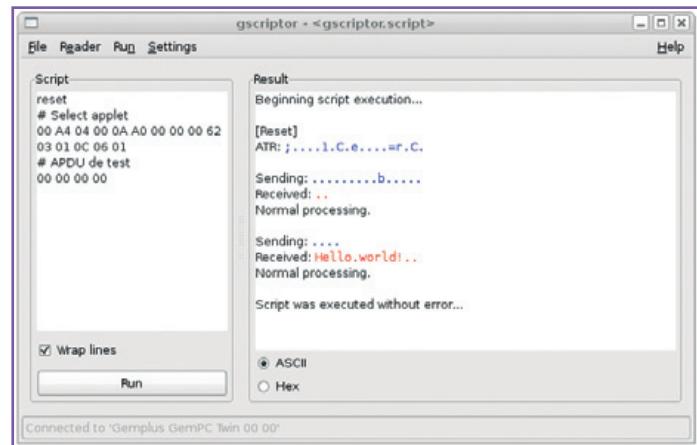
scriptor est assez rudimentaire, puisqu'il n'est pas possible de faire de traitement sur les résultats renvoyés par la carte. J'utilise cet outil pour effectuer des tests unitaires sur mon pilote CCID décrit dans l'article sur le projet MUSCLE dans ce même magazine. J'ai une carte de test que j'utilise pour générer tous les types de commandes et de toutes les longueurs possibles. Je peux ainsi très facilement utiliser individuellement la commande APDU qui fait planter le pilote ou le lecteur et corriger le problème.

1.4.2 gscriptor

gscriptor est équivalent à **scriptor**, mais avec une interface graphique utilisant Gtk+2 pour Perl.



gscriptor permet d'afficher les résultats en hexadécimal ou en ASCII. La représentation ASCII est particulièrement utile dans le cas présent puisque les données renvoyées par l'APDU de test sont du texte.



1.4.3 Amusez-vous avec une carte SIM

Pour mettre en œuvre de façon utile le wrapper PCSC/Perl, j'ai écrit un programme qui affiche les noms et numéros de téléphones correspondants présents dans l'annuaire de la carte SIM. Pour des problèmes de place, le code source n'est pas inclus ici, mais vous pouvez le récupérer sur mon site web [\[SIM-3.0\]](#).

Un gros avantage de la carte SIM par rapport à d'autres cartes courantes (carte Vitale ou carte bancaire par exemple) est que les spécifications sont publiques et gratuites. À l'époque de l'écriture du programme (2001), je crois me souvenir que les spécifications n'étaient pas publiques. Il fallait trouver une copie de la spécification GSM 11.11 aussi appelée *Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (GSM 11.11)*. Cette spécification (de 1995) se trouve maintenant facilement sur Internet.

Les cartes utilisées dans un téléphone UTMS ou 3G sont décrites par un document beaucoup plus récent : *3rd Generation Partnership Project; Technical Specification Group Core*

Network and Terminals; Characteristics of the USIM application (Release 1999) disponible en ligne sous la référence 3GPP TS 31.102 V3.18.0 (2005-06) [31.102]. La carte est appelée USIM (*Universal Subscriber Identity Module*) et non plus SIM qui est une carte utilisable uniquement sur un réseau GSM.

Une carte USIM devant être capable de fonctionner dans un téléphone GSM (c'est-à-dire non-3G) le programme devrait fonctionner sur une USIM même si ces cartes n'existaient pas encore lorsqu'il a été écrit. Je laisse le soin aux lecteurs motivés d'écrire une version améliorée pour pleinement supporter les cartes USIM. Pour plus d'informations sur les cartes SIM et

USIM je vous renvoie au hors-série carte à puce de MISC à paraître mi-novembre 2008.

1.5 Portabilité

PCSC/Perl fonctionne sur n'importe quel système GNU/Linux. Un paquet binaire existe déjà pour les principales distributions Linux. PCSC/Perl fonctionne également sur Mac OS X (Darwin) et sur Windows. Il existe, de plus, des ports pour NetBSD et FreeBSD.

2

PC/SC en Python

2.1

pycsc

Pycsc [pycsc] est un wrapper PC/SC pour Python écrit par Jean-Luc Giraud en 2002. Le projet ne fournit qu'un sous-ensemble des fonctions PC/SC. Par exemple, les transactions PC/SC et la fonction **SCardControl()** ne sont pas disponibles.

Le projet n'est plus actif depuis 2004 et peut avantageusement être remplacé par pyscard.

2.2

pyscard

pyscard [pyscard] est un projet de Jean-Daniel Aussel (voir article dans ce magazine). Un paquet Debian [python-pyscard](#), devrait être très prochainement disponible. Pour l'instant, le paquet est en attente dans la queue des nouveaux paquets (NEW queue).

2.2.1

Exemple

```
#!/usr/bin/env python
from smartcard.System import *
from smartcard.CardConnection import *
```

```
r = readers()
connection = r[0].createConnection()
connection.connect()

SELECT = [0x00, 0xA4, 0x04, 0x00, 0x0A, 0xA0, 0x00, 0x00, 0x00, 0x62,
0x03, 0x01, 0x0C, 0x06, 0x01]
data, sw1, sw2 = connection.transmit(SELECT, protocol=CardConnection.T1_protocol)
print data
print "%02x %02x" % (sw1, sw2)

data, sw1, sw2 = connection.transmit([0, 0, 0, 0],
protocol=CardConnection.T1_protocol)
print data
print "".join(map(chr, data))
print "%02x %02x" % (sw1, sw2)

connection.disconnect()
```

Ce qui me donne en sortie :

```
[]
90 00
[72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 33]
Hello world!
90 00
```

3

PC/SC en Caml

Manuel Preliteiro a écrit OCaml PC/SC [ocamlpcsc] commencé en 2003 et toujours actif en 2007. La dernière version du projet est 0.6 et est toujours en bêta, mais fonctionne très bien d'après mes tests.

3.1

Installation

Il faut d'abord installer OCaml. Un simple [apt-get install ocaml](#) suffit sur un système Debian (ou Debian-like). Ensuite,

récupérez l'archive [ocamlpcsc-0.6.tar.gz](#) sur le site [\[ocamlpcsc\]](#).

```
ocamlpcsc-0.6$ make
cc `pkg-config libpcsc-lite --cflags` -c convert.c
ocamlc -c pcscmacros.ml
ocamlc -c pcscmacros.ml
cc -I /usr/lib/ocaml/3.10.2 `pkg-config libpcsc-lite --cflags` -c pcscC.c
ocamlc -pp camlp4o -c pcscML.ml
ocamlc -custom -pp camlp4o -c pcscML.ml
```

3.2

Exemple

N'étant pas expert en Objective Caml, le code qui suit est une modification de l'exemple de code de Manuel pour utiliser notre applet d'exemple.

```
open Pcsccmacros;;
open Printf;;
open PcsccML;;
open Str;;
let func () =
  let (rvEstablishContext, hContext) = sCardEstablishContext scard_scope_
  system () () in
  if (rvEstablishContext != scard_s_success) then (
    printf "SCardEstablishContext: Cannot Connect to Resource Manager
%d\n" rvEstablishContext;
    flush(stdout);
    ())
  );
  let (rvListReaders, readers) = sCardListReaders () in
  printf "Readers List: \n";
  Array.iter (fun z -> print_string (z ^ " : ")) readers;
  printf "\n";
  flush(stdout);
  let reader1 = readers.(0) in
  let (rvSCardConnect, hCard, dwActiveProtocol) = sCardConnect reader1
  scard_share_shared scard_protocol_t1 in
  if (rvSCardConnect != scard_s_success) then (
    printf "SCardConnect: Cannot connect to card %d\n" rvSCardConnect;
    flush(stdout);
    ())
  );
  let pioSendPci = (scard_protocol_t1, 0) in
  let commande1 = [|0x00; 0xA4; 0x04; 0x00; 0x0A; 0xA0; 0x00; 0x00;
  0x00; 0x62; 0x03; 0x01; 0x0C; 0x06; 0x01|] in
  let dwSendLength = Array.length commande1 in
  let (rvSCardTransmit, pioRecvPci, pcRecvBuffer) = sCardTransmit
  hCard pioSendPci commande1 dwSendLength in
  if (rvSCardTransmit != scard_s_success) then (
    printf "SCardTransmit: Cannot transmit to card %d\n" rvSCardTransmit;
    flush(stdout);
    ())
  )
  else (
    let (_, _) = pioRecvPci in
    printf "Data transmitted with success: ";
    Array.iter (fun z -> print_string ((string_of_int z) ^ " :
")) pcRecvBuffer;
    print_string "\n";
    Array.iter (fun z -> print_char (char_of_int z)) pcRecvBuffer;
    print_string "\n";
    flush(stdout)
  );
  let rvSCardDisconnect = sCardDisconnect hCard scard_leave_card in
  if (rvSCardDisconnect != scard_s_success) then (
    printf "SCardDisconnect: Cannot disconnect card %d\n" rvSCardDisconnect;
    flush(stdout);
    ())
  );
;;
let _ = func ();;
(** ocamlopt -c demo.ml **)
(** ocamlopt -pp camlp4o str.cmxa -cclib '-lpcsclite convert.o pcscC.o' -o
demo pcscML.cmux pcscmacros.cmux demo.cmux **)
```

```
print_string "\n";
flush(stdout)
);

let commande2 = [|0x00; 0x00; 0x00; 0x00|] in
let dwSendLength = Array.length commande2 in
let (rvSCardTransmit, pioRecvPci, pcRecvBuffer) = sCardTransmit
hCard pioSendPci commande2 dwSendLength in
if (rvSCardTransmit != scard_s_success) then (
  printf "SCardTransmit: Cannot transmit to card %d\n" rvSCardTransmit;
  flush(stdout);
  ())
else (
  let (_, _) = pioRecvPci in
  printf "Data transmitted with success: ";
  Array.iter (fun z -> print_string ((string_of_int z) ^ " :
")) pcRecvBuffer;
  print_string "\n";
  Array.iter (fun z -> print_char (char_of_int z)) pcRecvBuffer;
  print_string "\n";
  flush(stdout)
);
let rvSCardDisconnect = sCardDisconnect hCard scard_leave_card in
if (rvSCardDisconnect != scard_s_success) then (
  printf "SCardDisconnect: Cannot disconnect card %d\n" rvSCardDisconnect;
  flush(stdout);
  ())
);
;;
let _ = func ();;
(** ocamlopt -c demo.ml **)
(** ocamlopt -pp camlp4o str.cmxa -cclib '-lpcsclite convert.o pcscC.o' -o
demo pcscML.cmux pcscmacros.cmux demo.cmux **)
```

Les deux dernières lignes du fichier source sont les commandes à exécuter pour compiler le programme. L'exécution du programme donne :

```
Readers List:
Gemplus GemPC Twin 00 00 :
Data transmitted with success: 144 : 0 :
Data transmitted with success: 72 : 101 : 108 : 108 : 111 : 32 : 119 : 111
: 114 : 108 : 100 : 33 : 144 : 0 :
Hello world!?
```

On peut remarquer un caractère « ? » après le texte « *Hello world!* ». Il s'agit de la conversion en ASCII du *status word* 0x90 0x00. Il faudrait en fait n'afficher que les n-2 premiers caractères de la liste. Je laisse la réalisation de cet exercice au lecteur.

4

PC/SC en Prolog

Renaud Mariana a écrit un wrapper PC/SC pour GNU Prolog. La dernière version 1.01 date de 2003 et est disponible dans les contributions à GNU Prolog [[gplpcsc](#)].

4.1

Installation

Il vous faut d'abord installer GNU Prolog : [apt-get install gprolog](#). Ensuite, il faut appliquer quelques modifications au projet, voir mon patch sur liste [users-prolog](#) [[gplpcsc-patch](#)]. La compilation se passe ensuite sans problème.

```
gplpcsc/contribs/gplpcsc$ make
gplc -c -C "-Wall -g `pkg-config --cflags libpcsclite`" pcsc/muscle_c.c
```

```
gplc -o testapdu pcsc/muscle_c.o -L "`pkg-config --libs libpcsc-lite` -lcrypto -lpthread" testapdu.p1 util_pcsc/ut il.p1
```

4.2 Exemple

```
:- include('pcsc/muscle').

start :- 
    scard_establish_context,
    scard_connect,
    % print ATR data
    scard_status(_, Atr),
    print('ATR: '), print_hex_list(Atr), nl,
    % enable trace
    scard_enable_verbose(true),
    Commande = [0,0xA4,4,0,0x0A,0xA0,0,0,0,0x62,3,1,0x0C,6,1],
    scard_transmit(Commande, _ApduOut1, _Sw1, _Time1),
    print_hex_list(_ApduOut1),
    print_hex_list(_Sw1), nl,
    scard_transmit([0,0,0,0], _ApduOut2, _Sw2, _Time2),
    print_hex_list(_ApduOut2),
```

```
print_hex_list(_Sw2), nl,
print_ASCII_list(_ApduOut2), nl.

print_ASCII_list(L) :-
    ( list(L) ->
        print('['), '$print_ASCII_list'(L,[]), print(']')
    ;
        print('print_ASCII_list error, argument is not a list')
    ).

'$print_ASCII_list' -> [], !.
'$print_ASCII_list' -> [A], { format('%c',[A]) },
    '$print_ASCII_list'.
:- initialization(start).
```

En sortie, j'obtiens :

```
ATR: [3B FA 94 00 00 81 31 20 43 80 65 A2 01 01 01 3D 72 D6 43 21 ]
tr: 00a40400aa0000006203010c0601 - 9000, t= 15ms.
[] [90 00]
tr: 00000000 - 48656c6c6f20776f726c64219000, t= 23ms.
[48 65 6C 6C 6F 20 77 6F 72 6C 64 21 ] [90 00]
[Hello world!]
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?-
```

5 PC/SC en Ruby

Il existe aussi un wrapper pour le langage Ruby [\[smartcard-ruby\]](#).

5.1 Installation

Il faut d'abord installer Ruby si vous ne l'avez pas déjà (c'était mon cas), ainsi que les autres paquets nécessaires à la compilation de smartcard pour Ruby : **apt-get install ruby rake rubygems libopenssl-ruby ruby1.8-dev...** puis, récupérer l'archive **smartcard.zip** [\[smartcard-ruby\]](#).

```
$ sudo gem install echoe
[...]
$ unzip smartcard.zip
$ cd smartcard
$ rake manifest
[...]
$ rake package
(in /home/rousseau/HSLM/smartcard)
  Successfully built RubyGem
  Name: smartcard
  Version: 0.3.1
  File: smartcard-0.3.1.gem
Private key not found; gem will not be signed.
Targeting "ruby" platform.
$ rake test
(in /home/rousseau/HSLM/smartcard)
/usr/bin/ruby1.8 extconf.rb
checking for main() in -lpcsc-lite... yes
checking for wintypes.h... yes
checking for reader.h... yes
checking for winscard.h... yes
checking for pcsc-lite.h... yes
creating Makefile
[...]
```

```
/usr/bin/ruby1.8 -Ilib:ext:bin:test "/var/lib/gems/1.8/gems/rake-0.8.3/lib/
rake/rake_test_loader.rb" "test/test_containers.rb" "test/test_smoke.rb"
Loaded suite /var/lib/gems/1.8/gems/rake-0.8.3/lib/rake/rake_test_loader
Started
...
Finished in 0.046223 seconds.
3 tests, 14 assertions, 0 failures, 0 errors
$ rake docs
[...]
```

5.2 Exemple

```
require 'smartcard'

context = Smartcard::PCSC::Context.new(Smartcard::PCSC::SCOPE_SYSTEM)
readers = context.list_readers nil
context.cancel

# Utilise le premier lecteur
reader = readers.first

# Connexion à la carte
card = Smartcard::PCSC::Card.new(context, reader, Smartcard::PCSC::SHARE_
SHARED, Smartcard::PCSC::PROTOCOL_ANY)

# Récupère le protocole à utiliser pour transmit
card_status = card.status

# Selection de l'applet
aid = [0xA0, 0x00, 0x00, 0x00, 0x62, 0x03, 0x01, 0x0C, 0x06, 0x01]
select_apdu = [0x00, 0xA4, 0x04, 0x00, aid.length, aid].flatten
send_ioreq = {Smartcard::PCSC::PROTOCOL_T0 => Smartcard::PCSC::IOREQUEST_T0,
              Smartcard::PCSC::PROTOCOL_T1 => Smartcard::PCSC::IOREQUEST_T1}[card_status[:protocol]]
recv_ioreq = Smartcard::PCSC::IoRequest.new
response = card.transmit(select_apdu.map {|bytel| bytel.tochr}.join(''))
```

```
send_ioreq, recv_ioreq)
response_str = (0..response.length).map { |i| ' %02x' % response[i].to_i }.join('')
puts "Réponse :#{response_str}\n"

# APDU de test
test_apdu = [0, 0, 0, 0]
response = card.transmit(test_apdu.map{|byte| byte.chr}.join(''), send_ioreq, recv_ioreq)
response_str = (0..response.length).map { |i| ' %02x' % response[i].to_i }.join('')
puts "Réponse :#{response_str}\n"
response_str = (0..response.length-2).map { |i| '%c' % response[i].to_i }.join('')
```

```
puts "Réponse : #{response_str}\n"
# Déconnexion et nettoyage
card.disconnect Smartcard::PCSC::DISPOSITION_LEAVE
context.release
```

En sortie, j'obtiens :

```
Réponse : 90 00
Réponse : 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 90 00
Réponse : Hello world!
```

6

PC/SC en Java

L'utilisation de cartes à puce depuis Java est un terrain mouvant. Ceci est peut-être dû au fait que Java est un langage choisi par des décideurs et non par des ingénieurs. Lorsque le décideur change de poste le projet d'interface Java/carte à puce disparaît avec lui.

6.1 OCF

OCF (*OpenCard Framework*) [[ocf](#)] est un projet d'un consortium d'industriels, dont IBM et Gemplus, commencé en 1998. À l'époque PC/SC n'était pas très développé et pcsc-lite était juste naissant. OCF inclut une couche équivalente à PC/SC, ainsi que des pilotes pour les lecteurs de carte et des pilotes pour les cartes à puce. Tout est écrit en Java (y compris les pilotes de lecteurs en s'appuyant sur une API d'accès direct au port série).

Le projet est très complet, mais n'a pas évolué avec les lecteurs actuels USB, faute de prise en charge normalisée de l'USB par Java. Heureusement, on peut utiliser ces lecteurs à l'aide d'un bridge PCSC/OCF, comme d'ailleurs tous les lecteurs (fonctionnant à l'aide de pilotes) PCSC. OCF permet toujours néanmoins la programmation bas niveau générique de n'importe quelle carte à puce, en échangeant des APDU. Un avantage indéniable est également de pouvoir interfaçer de manière transparente une vraie carte ou un simulateur comme celui du SDK Javacard officiel de Sun (approche d'ailleurs choisie par Vincent Guyot pour créer le SDK générique qu'il présente dans ce magazine). Le projet OCF a récemment été repris et est désormais hébergé sur [[openscdp](#)].

6.2 jpcsc

Marcus Oestreicher (d'IBM encore) a écrit jpcsc [[jpcsc](#)] en 2004. L'ambition est beaucoup plus réduite puisqu'il s'agit d'un wrapper au-dessus de PC/SC en utilisant le mécanisme JNI (*Java Native Interface*). On retrouve les appels PC/SC directement en Java. jpcsc a été développé dans le cadre du projet JCOP (*Java Card OpenPlatform*) d'IBM Zürich Research Laboratory.

La dernière version, 0.8.0, est suffisamment stable et complète pour déjà dater de 2004.

6.3 JSR 268

Le JSR (*Java Specification Request*) 268 aussi connu sous le nom *Java Smart Card I/O API* réutilise quelques idées d'OCF et utilise le process standard pour une évolution de Java. Le

travail a commencé en 2005 et la spécification finale est sortie en décembre 2006. Les travaux de ce JSR ont été inclus dans Java 6 (ou 1.6). N'importe quelle machine virtuelle Java (JRE ou *Java Runtime Environment*) version 6 ou supérieure devrait inclure le support du JSR 268.

En regardant dans le code source du JDK (*Java SE Development Kit*) disponible sur [[jdk6](#)] on apprend que le support carte à puce s'appuie sur PC/SC.

Le JRE version 6 est disponible pour Debian dans le paquet [openjdk-6-jre](#). L'accès aux fonctions carte à puce est fonctionnel si vous installez aussi le paquet [libpcslite-dev](#). En effet, la JVM cherche à charger la bibliothèque [libpcslite.so](#) alors qu'elle devrait utiliser [libpcslite.so.1](#). Notez le .1 à la fin identifiant la version de l'API.

Java 1.6.0 existe aussi pour Mac OS X, mais je n'ai pas réussi à utiliser mon programme d'exemple. La liste des lecteurs est vide alors qu'il y a bien un lecteur connecté et qu'il est vu par l'outil [pcstest](#). Il semble que cette nouvelle technologie n'est pas encore mature et n'atteint notamment pas encore le même degré de portabilité que OpenCard Framework.

6.3.1 Exemple

```
import java.util.List;
import javax.smartcardio.*;
public class LinuxMag {
    public static void main(String[] args) {
        try {
            // Affiche les terminaux (lecteurs) disponibles
            TerminalFactory factory = TerminalFactory.getDefault();
            List<CardTerminal> terminals = factory.terminals().list();
            System.out.println("Terminaux : " + terminals);

            // Utilise le premier terminal
            CardTerminal terminal = terminals.get(0);

            // Etablit une connexion avec la carte
            Card card = terminal.connect("T=1");
            System.out.println("carte : " + card);
            CardChannel channel = card.getBasicChannel();

            // Envoie la commande Select Applet
            byte[] aid = {(byte)0xA0, 0x00, 0x00, 0x00, 0x62, 0x03, 0x01,
                0x0C, 0x06, 0x01};
            ResponseAPDU réponse1 = channel.transmit(new CommandAPDU(0x00,
                0xA4, 0x04, 0x00, aid));
            System.out.println("réponse : " + réponse1.toString());
            // Envoie la commande de test
            ResponseAPDU réponse2 = channel.transmit(new CommandAPDU(0x00,
                0x00, 0x00, 0x00)));
```

```

        System.out.println("réponse : " + réponse2.toString());
        byte r[] = réponse2.getData();
        for (int i=0; i<r.length; i++)
            System.out.print((char)r[i]);
        System.out.println();
        // Déconnecte la carte
        card.disconnect(false);
    } catch (Exception e) {
        System.out.println("Aïe : " + e.toString());
    }
}

```

```

    }
}
```

Sortie :

```

Terminaux : [PC/SC terminal Gemplus GemPC Twin 00 00]
carte : PC/SC card in Gemplus GemPC Twin 00 00, protocol T=1, state OK
réponse : ResponseAPDU: 2 bytes, SW=9000
réponse : ResponseAPDU: 14 bytes, SW=9000
Hello world!

```

7

C#

J'ai trouvé un wrapper pour C#, mais c'est un logiciel propriétaire (*proprietary*). Je ne vais donc pas en faire la publicité ici (ni ailleurs).

Le logiciel C# Scriptor [**springcard**], qui est une version de gscriptor en C#, utilise PC/SC directement en chargeant la bibliothèque avec des **[DllImport("WinScard.dll", EntryPoint**

= "SCardEstablishContext")]. Je pense que cela ne fonctionne que sur Windows, mais qu'en utilisant **libpcsc-lite.so.1** à la place de **WinScard.dll** le même mécanisme devrait fonctionner sur Unix avec l'implémentation libre Mono de .NET.

La licence du logiciel C# Scriptor n'est pas libre, même si le code source est disponible.

8

Conclusion

Tous les projets décrits dans cet article (hormis ceux pour C#) sont, sauf erreur, des logiciels libres. Dans la majorité des cas, les wrappers sont inclus dans la distribution officielle (cas de Java 1.6), inclus sur le dépôt du projet (cas de Prolog) ou référencés sur les sites officiels des contributions au langage (cas de Python avec <http://pypy.python.org/>, Perl avec <http://www.cpan.org/> et OCaml avec <http://caml.inria.fr/>). Les projets sont donc très faciles à trouver.

Il existe sûrement des wrappers PC/SC pour d'autres langages. Par exemple, je viens de découvrir un projet (commencé en juin 2008) de wrapper pour ADA [**PCSC/Ada**] réalisé par Reto Buerki. Des projets naissent et d'autres disparaissent. Il existait un wrapper pour TCL en 1998 appelé TclSC (*The Tcl Smart Card Extension*), mais le site web n'existe plus que dans le cache de Google ou dans la *wayback machine* [**tclsc**].

REMERCIEMENTS

Merci à Lionel pour la présentation du projet PCSC/Perl. Bonne chance à lui dans son nouveau travail. Merci à Vincent Guyot de m'avoir suggérer d'écrire cet article. Merci à Marlène pour ses encouragements et ses relectures. Ce tour d'horizon des wrappers PC/SC a été assez amusant même si réalisé suivant la méthodologie de La Rache [**la rache**]. Et surtout merci à tous les auteurs des projets décrits dans l'article.

BIBLIOGRAPHIE

- [**31.102**] http://www.3gpp.org/ftp/Specs/archive/31_series/31.102/31102-310.zip – 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Characteristics of the USIM application (Release 1999).
- [**CPAN**] <http://search.cpan.org/~whom/pcsc-perl-1.4.7/> – PCSC/Perl sur CPAN.
- [**gplpcsc**] <http://gprolog CVS.sourceforge.net/gprolog/contribs/gplpcsc/> – PC/SC Lite interface for GNU Prolog.

- [**gplpcsc-patch**] <http://lists.gnu.org/archive/html/users-prolog/2008-10/msg00001.html> – patch for gppcsc (PC/SC interface).
- [**jdk6**] <http://download.java.net/jdk6/6u3/promoted/b05/index.html> – Java™ Platform, Standard Edition 6u3 Source Snapshot Releases. Fichier **jdk-6u3-fcs-src-b05-jrl-24_sep_2007.jar**
- [**jpcsc**] <http://www.linuxnet.com/middle.html> – JPC/SC Java API
- [**la rache**] <http://www.la-rache.com/> – site of the International Institute of La RACHE.
- [**ocamlpcsc**] <http://www.di.ubi.pt/~desousa/ocamlpesc/> – OCaml PC/SC Binding to the PCSC standard.
- [**ocf**] <http://www.gemalto.com/techno/opencard/> – OCF, the OpenCard Framework is a standard Java framework for working with Smart Cards.
- [**openscdp**] <http://www.openscdp.org/ocf/> – OpenCard Framework.
- [**PCSC/Ada**] <http://www.nongnu.org/pcscada/> – PCSC/Ada.
- [**pcsc-perl**] <http://ludovic.rousseau.free.fr/softwares/pcsc-perl/> – pcsc-perl.
- [**pcsc-tools**] <http://ludovic.rousseau.free.fr/softwares/pcsc-tools/> – pcsc-tools.
- [**pycsc**] <http://homepage.mac.com/jlgiraud/pycsc/Pycsc.html> – Pycsc: a pcsc wrapper for Python.
- [**pyscard**] <http://pyscard.sourceforge.net/> – pyscard – python smart card library.
- [**SIM-3.0**] <http://ludovic.rousseau.free.fr/softwares/SIM-3.0.tar.gz> – SIM.pl : print the phone book of a GSM SIM card.
- [**smartcard-ruby**] <http://rubyforge.org/projects/smartsard/> – Interface for ISO 7816 Smart Cards.
- [**springcard**] <http://springcard.com/fi/solutions/pcsc.html> – C# Scriptor.
- [**tclsc**] http://web.archive.org/web/*http://www.scdk.com/tclsc/default.htm – TclSC – The Tcl Smart Card Extension.



L'utilisation de smartcards ou cartes à puce intelligentes se généralise. Ceci forme une réponse à une demande toujours grandissante de sécurité pour l'utilisateur.

La popularisation et l'aspect mafieux du vol d'identité et du phishing créent un contexte où une authentification plus forte des utilisateurs devient indispensable. Les smartcards représentent une réponse idéale et GNU/Linux n'est pas en reste face aux très nombreuses solutions existantes sous Windows.

Introduction à l'utilisation de smartcards sous GNU/Linux

Auteur : Denis Bodor

Le support des *smartcards* ou *tokens*, périphériques USB généralement constitués d'un lecteur et d'une puce de carte *packagé* sous la forme d'une clef, prend généralement deux formes sous GNU/Linux. Généralement de type USB, le périphérique de lecture est pris en charge par un *middleware*. Ce dernier permet de faire office de passerelle pour les applications souhaitant utiliser le périphérique.

Deux projets se partagent la vedette sous GNU/Linux. Nous avons d'une part le couple OpenCT/OpenSC et de l'autre PC/SC Lite (voir article de L. Rousseau et D. Corcoran dans le présent magazine). Parlons tout d'abord d'OpenSC. Ce projet produit plusieurs bibliothèques, applications et outils en rapport avec les smartcards, leur support et leur gestion. Les deux éléments importants sont OpenCT et OpenSC. Le premier est le middleware à proprement parler permettant l'accès aux lecteurs de cartes (*Card Terminal*). D'après le créateur même d'OpenCT, le principal intérêt est la prise en charge des périphériques non supportés par PC/SC Lite, à quoi il faut ajouter l'éventuelle préférence personnelle du sysadmin en termes de configuration.

Le couple OpenCT/OpenCS intègre son propre support pour certains lecteurs, mais peut également accéder aux services fournis par PC/SC Lite et accéder aux lecteurs uniquement supportés via CT-API. CT signifie *Chipcard Terminal*. Il s'agit d'une API concurrente à PC/SC. Les deux projets s'entendent, semble-t-il, pour pousser les utilisateurs vers PC/SC lite. L'idée est donc d'utiliser le middleware PC/SC lite avec les outils PKCS#15 d'OpenSC.

PC/SC justement est un standard défini par le groupe de travail du même nom. Ce standard définit une API permettant d'accéder de manière unifiée à un lecteur de carte et une carte quels que soient le modèle et le fabricant, pourvu qu'il existe un pilote. Une implémentation libre du standard PC/SC est PC/SC Lite. Selon votre distribution, l'organisation et la répartition des éléments logiciels en paquets peuvent varier, mais PC/SC Lite se compose :

- D'un démon **pcscd** fonctionnant comme gestionnaire de ressources et partageant l'accès aux périphériques pour les applications.
- De pilotes prenant la forme de bibliothèques dynamiques appelées « *ifdhandlers* ». Ces pilotes permettent d'accéder aux lecteurs via

la libUSB (entre autres). Ces ifdhandlers sont normalement placés dans le répertoire **/usr/lib/pcsc/drivers** selon une arborescence précise. Un fichier XML Property list accompagne le pilote et décrit, entre autres, à quels lecteurs il est destiné (couple VendorID/ProductID).

- D'une bibliothèque **libpcsclite.so** permettant la création d'applications voulant accéder à la carte. C'est l'API unifiée fournie par PC/SC Lite (voir article sur le projet MUSCLE).

La configuration de PC/SC Lite se résume donc à l'installation du démon et d'un ou plusieurs ifdhandlers. Ceci prend

généralement la forme d'une installation d'un paquet **pcscd** et d'un ou plusieurs paquets qu'on listera facilement sous Debian et dérivées avec **apt-cache search PC/SC driver**.

Là où tout cela devient intéressant, c'est dans la polyvalence des solutions. En effet, OpenSC peut utiliser le démon PC/SC Lite. Inversement, le démon PC/SC Lite est en mesure d'accéder aux périphériques pris en charge par OpenCT. Il est donc possible virtuellement d'accéder à n'importe quel lecteur, du moment qu'il dispose d'un support dans l'un ou l'autre projet.

1

Les lecteurs

Afin de bien comprendre l'intérêt de l'utilisation d'OpenCT et/ou de PC/SC Lite, nous allons mettre en œuvre deux lecteurs USB. Les modèles de lecteurs se répartissent sommairement en deux catégories. Nous avons d'une part les lecteurs compatibles CCID qui fonctionneront avec un pilote générique aussi bien avec OpenCT qu'avec PC/SC Lite et d'autre part... les autres lecteurs. Vous vous doutez bien que, lors de l'achat, il est fortement recommandé de choisir un lecteur compatible CCID. Malheureusement, selon le cas, il n'est pas toujours possible de choisir (lecteur déjà en place, achat sur Ebay, etc.).

1.1

GemPC430

Notre premier exemple est un lecteur de cartes GemPlus GemPC430 non compatible CCID (**08e6:0430**). Sa mise en œuvre nécessite l'utilisation de PC/SC Lite, car il n'existe pas de support pour OpenCT (ce qui est généralement le cas pour les anciens lecteurs). PC/SC Lite, en revanche, dispose d'un ifdhandler approprié qu'on installera via le paquet Debian **libgempc430**. Celui-ci prend en charge les lecteurs GemPC 430, 432 et 435 sous la forme d'une bibliothèque **libGemPC430.so.1.0.3** et d'un fichier **ifd-GemPC430.bundle/Contents/Info.plist**. Une fois le paquet installé et après un redémarrage du démon **pcscd**, on constatera la bonne prise en charge via la commande **pcsc_scan** :

```
% pcsc_scan
Scanning present readers
0: GemPC430 00 00

Tue Sep 30 07:39:47 2008
Reader 0: GemPC430 00 00
Card state: Status unavailable,
```

1.2

eToken Pro 32k

Ce périphérique est un token USB, un lecteur couplé à une smartcard. Le support PC/SC Lite n'existe pas à ma connaissance.

Il est donc nécessaire d'utiliser le middleware OpenCT. Après installation, on parcourra le fichier **/etc/openct.conf** regroupant les couples VendorID/ProductID des lecteurs et/ou tokens supportés. On y retrouve ainsi les valeurs de notre périphérique produit par Aladdin Knowledge Systems :

```
driver etoken {
    ids = {
        usb:0529/050c,
        usb:0529/0514,
    };
};
```

On vérifiera ensuite la bonne prise en charge par OpenCT avec :

```
% openct-tool list
Ø Aladdin eToken PRO
```

Nous venons de mettre en œuvre deux middlewares différents pour deux types de lecteurs différents. Nous verrons plus loin comment ensuite, rendre tout cela interopérable. Il nous reste cependant un type de lecteur à détailler : le lecteur compatible CCID.

1.3

SCM SCR335

Ce lecteur fabriqué par SCM Microsystems est un excellent choix du fait de sa compatibilité. Ainsi, nous avons le choix entre PC/SC Lite et OpenCT. Les deux middlewares reconnaîtront le périphérique sans le moindre problème :

```
% pcsc_scan
Scanning present readers
Ø: SCM SCR 335 00 00

Wed Oct 1 16:39:32 2008
Reader 0: SCM SCR 335 00 00
Card state: Card removed,

% openct-tool list
Ø CCID Compatible
```

2

Outils et interaction de middlewares

Il n'est pas très courant de disposer de plusieurs lecteurs ou tokens différents sur un seul système. Cependant, les cas particuliers existent et nous allons en profiter pour détailler l'interaction entre OpenCT, PC/SC Lite et les outils OpenSC. Pour ce faire, nous allons envisager un cas précis où nous allons faire usage du eToken d'Aladdin Knowledge Systems, ainsi que du lecteur GemPC430 avec une carte OpenPGP.

Les deux smartcards ont des usages différents. La carte OpenPGP, comme son nom l'indique, est conçue pour une utilisation avec GnuPG (voir article sur le sujet dans le présent magazine). D'autre part, le token est destiné à une utilisation avec des outils reposant sur SSL/TLS comme OpenSSH, OpenVPN, etc.

Comment faire en sorte d'utiliser les deux middlewares de concert avec les mêmes outils ? Ceci ne présente pas de difficultés particulières dans le sens où chaque middleware dispose d'un support pour un périphérique précis. Nous n'avons donc qu'à installer OpenCT et PC/SC Lite et lancer les deux services associés. OpenCT « voit » le token et **pcsc_scan** le lecteur GemPC430 et la carte OpenPGP :

```
% pcsc_scan
PC/SC device scanner
V 1.4.14 (c) 2001-2008,
  Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.4.101
Scanning present readers
0: GemPC430 00 00

Wed Oct 1 16:50:54 2008
Reader 0: GemPC430 00 00
Card state: Card inserted,
ATR: 3B FA 13 00 FF 81 31 80 45 00 31 C1 73 C0 01 00 00 90 00 B1

ATR: 3B FA 13 00 FF 81 31 80 45 00 31 C1 73 C0 01 00 00 90 00 B1
+ TS = 3B --> Direct Convention
+ T0 = FA, Y(1): 1111, K: 10 (historical bytes)
TA(1) = 13 --> Fi=372, Di=4, 93 cycles/ETU
          (38400 bits/s at 3.57 MHz)
TB(1) = 00 --> VPP is not electrically connected
TC(1) = FF --> Extra guard time: 255 (special value)
TD(1) = 81 --> Y(i+1) = 1000, Protocol T = 1
-----
TD(2) = 31 --> Y(i+1) = 0011, Protocol T = 1
-----
TA(3) = 80 --> IFSC: 128
TB(3) = 45 --> Block Waiting Integer: 4 -
          Character Waiting Integer: 5
+ Historical bytes: 00 31 C1 73 C0 01 00 00 90 00
Category indicator byte: 00 (compact TLV data object)
Tag: 3, len: 1 (card service data byte)
Card service data byte: C1
  - Application selection: by full DF name
  - Application selection: by partial DF name
  - EF.DIR and EF.ATR access services:
    by GET RECORD(s) command
  - Card without MF
Tag: 7, len: 3 (card capabilities)
Selection methods: C0
```

```
- DF selection by full DF name
- DF selection by partial DF name
Data coding byte: 01
- Behaviour of write functions: one-time write
- Value 'FF' for the first byte of BER-TLV
tag fields: invalid
- Data unit in quartets: 2
Command chaining, length fields and logical channels: 00
- Logical channel number assignment: No logical channel
- Maximum number of logical channels: 1
Mandatory status indicator (3 last bytes)
LCS (life card cycle): 00 (No information given)
SW: 9000 (Normal processing.)
+ TCK = B1 (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B FA 13 00 FF 81 31 80 45 00 31 C1 73 C0 01 00 00 90 00 B1
OpenPGP
```

Les outils de gestion PKCS#15 (voir introduction en début de magazine) fournis par OpenSC nous montrent les deux lecteurs après configuration du fichier **/etc/opensc/opensc.conf** contenant :

```
reader_drivers = pcsc,openct;
```

La directive **reader_drivers** permet de préciser à OpenSC la manière et l'ordre dans lequel tester les middlewares qui sont utilisés pour accéder aux lecteurs. Ici, nous spécifions PC/SC Lite, puis OpenCT. Ainsi, via **opensc-tool**, nous pouvons lister les lecteurs disponibles et tester la responsivité des smartcards :

```
% opensc-tool -l
Readers known about:
Nr.   Driver      Name
0     pcsc        GemPC430 00 00
1     openct       Aladdin eToken PRO

% opensc-tool -r 0 -an
3b:fa:13:00:ff:81:31:80:45:00:31:c1:73:c0:01:00:00:90:00:b1
OpenPGP

% opensc-tool -r 1 -an
3b:f2:98:00:ff:c1:10:31:fe:55:c8:03:15
CardOS M4
```

L'option **-r** nous permet de spécifier le lecteur à utiliser, **-a** retourne l'ATR de la carte (*answer to reset*) et **-n** permet d'identifier la carte et d'afficher son nom. On voit clairement que, pour un outil « utilisateur », le middleware utilisé est totalement invisible (c'est le but d'un middleware me direz-vous).

Mais, nous pouvons aller plus loin dans l'intégration. En effet, dans la situation actuelle, c'est OpenSC qui est configuré de manière à « jongler » avec les middlewares disponibles. Nous pouvons cependant configurer PC/SC Lite de manière à utiliser OpenCT. Nous commençons donc par modifier la configuration d'OpenSC de manière à restreindre son support à PC/SC en

précisant : **reader_drivers = pcsc;** dans **opensc.conf**. À ce stade, nous ne voyons plus que le lecteur GemPC430 avec **opensc-tool -l**.

Nous pouvons ensuite nous pencher sur la prise en charge de l'ifdhandler OpenCT pour le démon **pcscd**. Le paquet Debian **openct** a installé un pilote PC/SC Lite pour OpenCT dans **/usr/lib/pcsc/drivers/openct-ifd.bundle**. Dans le fichier XML **Info.plist** associé (un lien symbolique vers **/etc/openct-Info.plist**) se trouvent les couples VendorID/ProductID des périphériques concernés. Une petite modification s'impose pour notre token. En effet, le couple spécifié est **0x0529/0x0600**. Nous modifions donc le ProductID en **0x0514**.

Dernière étape pour la prise en charge, nous devons modifier le contenu de **/etc/reader.conf.d/openct** et décommenter les lignes :

```
FRIENDLYNAME      "OpenCT"
DEVICE_NAME       "/dev/null"
LIBPATH          "/usr/lib/openct-ifd.so"
CHANNELID        0
```

Enfin, nous appelons **update-reader.conf** pour générer un **/etc/reader.conf** adéquat. Ceci est spécifique aux distributions Debian et dérivées qui aiment à diviser les fichiers de configuration de la sorte. Avec une autre distribution, la simple modification du fichier **reader.conf** fera l'affaire.

Après redémarrage du démon **pcscd**, nous pouvons utiliser **opensc-tool** à nouveau pour constater les changements :

```
% opensc-tool -l
Readers known about:
Nr.   Driver     Name
0     pcsc      OpenCT 00 00
1     pcsc      GemPC430 00 00
```

Nous voyons clairement que le premier lecteur (**0**) utilise un pilote **pcsc** ayant pour nom **OpenCT**. C'est notre lien vers le token d'Aladdin :

```
% opensc-tool -r 0 -an
3b:f2:98:00:ff:c1:10:31:fe:55:c8:03:15
CardOS M4
```

À présent, **pcsc_scan** peut nous retourner les informations sur le token (dérivées tout simplement de l'ATR de la smartcard) :

```
Scanning present readers
0: OpenCT 00 00
1: GemPC430 00 00

Wed Oct 1 17:56:24 2008
Reader 0: OpenCT 00 00
Card state: Card inserted,
ATR: 3B F2 98 00 FF C1 10 31 FE 55 C8 03 15

ATR: 3B F2 98 00 FF C1 10 31 FE 55 C8 03 15
+ TS = 3B --> Direct Convention
+ T0 = F2, Y(1): 1111, K: 2 (historical bytes)
TA(1) = 98 --> Fi=512, Di=12, 42.6667 cycles/ETU
(83700 bits/s at 3.57 MHz)
TB(1) = 00 --> VPP is not electrically connected
TC(1) = FF --> Extra guard time: 255 (special value)
```

```
TD(1) = C1 --> Y(i+1) = 1100, Protocol T = 1
-----
TC(2) = 10 --> Work waiting time: 960 x 16 x (Fi/F)
TD(2) = 31 --> Y(i+1) = 0011, Protocol T = 1
-----
TA(3) = FE --> IFSC: 254
TB(3) = 55 --> Block Waiting Integer: 5 -
Character Waiting Integer: 5
+ Historical bytes: C8 03
Category indicator byte: C8 (proprietary format)
+ TCK = 15 (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B F2 98 00 FF C1 10 31 FE 55 C8 03 15
Siemens CardOS M 4.01 (SLE66CX320P)
```

Le token comme la carte OpenPGP utilisent un système de fichiers PKCS#15. Nous pouvons donc tester le bon fonctionnement de l'ensemble avec **opensc-explorer** :

```
% opensc-explorer -r 0
OpenSC Explorer version 0.11.4
OpenSC [3F00]> ls
FileID Type Size
[6666] DF 18184 Name: AKS
[5015] DF 18184
[2F00] WEF 128
OpenSC [3F00]>
```

```
% opensc-explorer -r 1
OpenSC Explorer version 0.11.4
OpenSC [3F00]> ls
FileID Type Size
004F WEF 16
[005E] DF 5
[0065] DF 23
[006E] DF 199
[0073] DF 0
[007A] DF 5
5F50 WEF 0
[B600] DF 141
[B800] DF 141
[A400] DF 141
B601 WEF 142
B801 WEF 142
A401 WEF 142
OpenSC [3F00]>
```

NOTE

Problème de lecteur CCID

Ceci peut paraître très surprenant, mais les lecteurs CCID du fait de leur prise en charge à la fois par OpenCT et PC/SC Lite provoquent un problème. En effet, le lancement d'OpenCT prend en charge automatiquement le périphérique **04e6:5115**. Malheureusement, au lancement du démon **pcscd**, nous obtenons une erreur :

```
00006423 ccid_usb.c:402:OpenUSBBByName()
Can't claim interface 002/020: Device or resource busy
0000010 ifdhandler.c:99:IFDHCreateChannelByName() failed
0000006 readerfactory.c:1121:RFInitializeReader() Open Port 20000 Failed
```

```
(usb:04e6/5115:libhal:/org/freedesktop/Hal/devices/
usb_device_4e6_5115_21120617208509_4_if0)
```

Et pour cause ! Le périphérique **04e6/5115** est déjà pris en charge et contrôlé par une application. Vous me direz, il suffit de laisser toute la prise en charge (eToken et lecteur CCID) se faire via OpenCT. Certes, cela fonctionne, mais des applications comme GnuPG utilisant le démon **pcscd** ou leur support interne CCID rencontreront un problème.

Après avoir testé bon nombre de possibilités, la seule solution semble être le fait de jongler avec les middlewares. En effet, il n'a pas été possible de désactiver le support pour un lecteur donné dans **openct.conf**. D'autre part, la solution consistant à retirer le support CCID de PC/SC Lite (**apt-get remove libccid**) aurait pu fonctionner, mais le démon **pcscd** ne semble voir que le premier lecteur géré par OpenCT.

3

Mise en application

À présent, vous devez être en mesure d'installer un lecteur de carte ou un token et d'y accéder via un ou plusieurs middlewares. Nous pouvons donc passer à la partie la plus intéressante, la mise en œuvre avec des applications courantes. Le support choisi pour cette partie de l'article est l'eToken 32k d'Aladdin. Celui-ci est supporté par OpenCT et est compatible PKCS#11. De plus, les informations contenues dans la mémoire sont au format PKCS#15, ce qui permet quelques expérimentations avec des outils comme **opensc-explorer**. L'eToken est destiné à la génération, à la gestion et au stockage de données SSL/x509. Le périphérique prend en charge un certain nombre d'algorithmes de chiffrement et de fonctions de hachage cryptographique (RSA 1024-bit/2048-bit, DES, 3DES, SHA1). L'utilisation principale de ce type de matériel est donc directement en rapport avec les utilitaires et applications SSL/TLS.

3.1

Préparation & initialisation

Lorsque vous recevez l'eToken, celui-ci est vierge. La mémoire Flash permettant de stocker les informations SSL, ainsi que les éléments de gestion (identités, code PIN, etc.) doit être initialisée. On utilisera pour ce faire les outils mis à disposition par le projet OpenSC permettant le formatage de la mémoire pour une utilisation avec PKCS#15. Remarquez que l'initialisation est indispensable pour ce type d'usage. En effet, les outils propriétaires du constructeur n'utilisent pas de standard. Oubliez donc la compatibilité entre ces outils et ce dont nous allons parler maintenant. Si vous recherchez une compatibilité avec des plateformes propriétaires (MS/Windows ou Mac OS X), il est toutefois possible d'utiliser des versions d'OpenSC spécifiques à ces plateformes. Initialiser la clef est d'une simplicité enfantine :

```
% pkcs15-init -ECT
New Security Officer PIN (Optional - press return for no PIN).
Please enter Security Officer PIN:
```

Les trois options servent respectivement :

- à effacer le contenu de la mémoire de la carte ;
- à créer une arborescence PKCS#15 dans cette mémoire ;
- à forcer l'utilisation de la clef de transport de la carte.

Cette clef spécifique est connue, en fonction de la carte en présence, par les outils OpenSC. Elle permet d'effacer la carte et de réinitialiser le SO-PIN (voir ci-après). L'option T permet de forcer l'utilisation de cette clef lorsqu'elle est nécessaire, et ce, même si le pilote ou le lecteur pense l'avoir en cache (le cache, c'est mal). L'accès aux données d'une smartcard est protégé par un code PIN (*Personal Identity/Identification Number*). Le principe est exactement le même que pour la carte SIM de votre mobile (qui est une smartcard). La différence tient dans le fait que le code PIN d'un eToken (ou d'une OpenPGP Card par exemple) peut être non numérique et faire bien plus de 4 caractères. Le Security Officer PIN (ou SO-PIN) qui vous est demandé est optionnel. La notion d'« officier de sécurité » est liée à celle de « PKI ». L'officier de sécurité est la personne habilitée à ajouter et supprimer des couples certificats/clef privée de la smartcard, mais pas à les utiliser. Il est donc le responsable de la PKI, mais ses pouvoirs sont cependant limités. À présent que la carte est effacée et formatée, nous allons ajouter un minimum de sécurité pour l'utilisateur et définir immédiatement son code PIN :

```
pkcs15-init -PT -a 1 -l denis -v
Connecting to card in reader Aladdin eToken PRO...
Using card driver Siemens CardOS.
Found OpenSC Card
About to store PIN.
New User PIN.
Please enter User PIN: ****
Please type again to verify: ****
Unblock Code for New User PIN (Optional - press return for no PIN).
Please enter User unblocking PIN (PUK): ****
Please type again to verify: ****
```

L'option **P** permet de créer un nouveau code PIN. **-a** permet de préciser l'ID du code PIN à créer et **-l** un label pour cet ID. Classiquement, **-v** rend l'outil plus bavard. L'objet est créé sur le token dans le système de fichier PKCS#15. Nous pouvons d'ailleurs vérifier cet état de fait avec :

```
% pkcs15-tool --list-pins
PIN [denis]
  Com. Flags: 0x3
  ID      : 01
  Flags   : [0x32], local, initialized, needs-padding
  Length  : min_len:4, max_len:8, stored_len:8
  Pad char: 0x00
  Reference: 1
  Type    : ascii-numeric
  Path    : 3f005015
```

ATTENTION

Certaines cartes ou tokens sont livrées avec un code PIN ou un PUK prédéfini. Lisez la documentation de votre produit ! Dans bien des cas, trois codes PIN erronés entraînent un blocage. Il faut alors utiliser le code PUK pour débloquer et créer un nouveau code PIN. Trois tentatives d'utilisation du code PUK menant à un échec bloquent définitivement la quasi-totalité des cartes. Je dis bien DÉFINITIVEMENT ! Pour débloquer un code PIN, utilisez **pkcs15-tool -u** et suivez les indications.

Notez qu'il est possible de créer plusieurs identités sur le token avec des ID et des labels différents.

3.2 Installation des clefs et certificats

L'étape suivante dans l'initialisation du token consiste à placer les certificats x509 et les clef privées associées. Deux stratégies sont applicables. Nous avons d'une part la génération par le token lui-même et, d'autre part, la génération via **openssl** et la copie sur le périphérique.

La génération par la carte se fera via :

```
% pkcs15-init -G RSA -a 1 -v -u sign,decrypt --split-key
Connecting to card in reader Aladdin eToken PRO...
Using card driver Siemens CardOS.
Found OpenSC Card
About to generate key.
User PIN required.
Please enter User PIN: ****
```

Nous utilisons l'option **-G (--generate-key)** suivie du type de clef demandée (RSA). On associe cette paire de clefs à l'ID d'un PIN nouvellement créé avec **-a**. Notez l'utilisation de l'option **--split-key**. Ceci répond à une fonctionnalité souvent intégrée aux smartcards qui prévoit une sécurité renforcée en n'autorisant pas une même clef privée à être utilisée pour le chiffrement et la signature. La clef privée sera présente deux fois sur la carte avec un label d'utilisation différent et un numéro d'objet unique. Le label d'utilisation est spécifié avec l'option **-u**. Nous pouvons lister le contenu avec **pkcs15-tool** :

```
% pkcs15-tool --list-keys | egrep "Usage|ID"
  Usage      : [0x22], decrypt, unwrap
  Auth ID   : 01
  ID        : 45

  Usage      : [0x20C], sign, signRecover, nonRepudiation
  Auth ID   : 01
  ID        : 45

  Usage      : [0x22], decrypt, unwrap
  Auth ID   : 02
  ID        : 46

  Usage      : [0x20C], sign, signRecover, nonRepudiation
  Auth ID   : 02
  ID        : 46
```

et :

```
% pkcs15-tool --list-public-keys | egrep "Usage|ID"
  Usage      : [0x4], sign
  Auth ID   :
  ID        : 45

  Usage      : [0x4], sign
  Auth ID   :
  ID        : 46
```

NOTE

Certaines options permettent de définir des comportements spécifiques. C'est le cas, par exemple, de l'option **--extractable** permettant de spécifier que la clef privée peut être extraite de la smartcard. **--insecure** porte bien son nom, car elle retire l'obligation de spécifier le code PIN protégeant la clef.

Nous allons maintenant créer un certificat intégrant la clef publique précédemment générée. Les outils d'OpenSC ne le permettent pas directement, il faut donc utiliser la commande **openssl** et un module spécifique permettant à la commande de dialoguer avec la smartcard. Ce module est un Engine OpenSSL, sorte de greffon chargeable dynamiquement permettant d'ajouter des fonctionnalités cryptographiques supplémentaires. Le module qui nous intéresse ici est une implémentation du PKCS#11. Ce module peut être installé via le paquet **libengine-pkcs11-openssl** sur distribution Debian.

Après lancement de la commande **openssl**, il convient de charger le module :

```
% openssl
OpenSSL> engine dynamic -pre \
SO_PATH:/usr/lib/engines/engine_pkcs11.so \
-pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD \
-pre MODULE_PATH:opensc-pkcs11.so
(dynamic) Dynamic engine loading support
[Success]: SO_PATH:/usr/lib/engines/engine_pkcs11.so
[Success]: ID:pkcs11
[Success]: LIST_ADD:1
[Success]: LOAD
[Success]: MODULE_PATH:opensc-pkcs11.so
Loaded: (pkcs11) pkcs11 engine
OpenSSL>
```

On peut ensuite accéder à la clef privée de la smartcard et générer un certificat auto-signé (ou une demande de certificat en omettant **-X509**) via : **req -engine pkcs11 -new -key id_45 -keyform engine -out cert.pem -text -X509**. Ceci fait et après éventuelle signature sur certificat par une autorité de certification, il suffit d'importer le contenu du fichier **cert.pem** dans le token avec **pkcs15-init -X cert.pem -v -a 1**.

On peut ensuite s'assurer de la réussite de l'opération avec :

```
% pkcs15-tool -c
X.509 Certificate [Certificate]
Flags    : 2
```

```
Authority: no
Path     : 3f0050153149
ID      : 45
```

Dans cette manipulation, vous aurez sans doute remarqué que la clef privée n'existe que sur le token. En aucune manière, celle-ci n'aura transité par la mémoire du PC ou d'un disque. C'est la méthode généralement utilisée. On peut cependant préférer générer une paire de clefs sur le PC pour ensuite charger le token des informations utiles.

On commence alors par produire la paire de clefs ou, plus exactement, la clef privée et la demande de certificat avec **openssl -newkey rsa:2048 -keyout moi.key -out moi.crt -text**. Nous obtenons ainsi **moi.key** contenant la clef privée et **moi.crt**, la demande de certificat.

C'est ensuite **pkcs15-init** qui entre en action pour charger le token :

```
% pkcs15-init -S moi.key -a 1 -u sign,decrypt --split-key
Please enter passphrase to unlock secret key: *****
User PIN required.
Please enter User PIN: ****

% pkcs15-init -X moi.crt -v -a 1
Using card driver Siemens CardOS.
Found OpenSC Card
About to store certificate.
User PIN required.
Please enter User PIN: ****
```

Remarquez que la phrase de passe protégeant la clef privée (sur le disque dur) est demandée juste avant le code PIN ouvrant l'accès à la smartcard. Le fichier **moi.key** DOIT être mis en sécurité. Vous pouvez le stocker sur une clef USB ou tout simplement l'imprimer avant de le supprimer de la machine en utilisant un outil comme Wipe. Placez le tout dans un coffre à l'abri des regards indiscrets.

À ce stade, notre token est prêt à être utilisé.

3.3 OpenSSH

C'est sans doute là la première application qui vient à l'esprit. Pouvoir se connecter à distance sur une machine et pouvoir avoir toujours sur soi la clef privée prouvant son identité. Le support des smartcards n'est pas complet dans la version officielle d'OpenSSH. Il existe un moyen de contourner le problème et également plusieurs patchs apportant des corrections ou des fonctionnalités supplémentaires.

La version d'OpenSSH telle que livrée par Debian n'est pas compilée avec le support des smartcards. Il vous faudra donc, dans un premier temps, recompiler un client OpenSSH afin d'activer une option de compilation spécifique. Vous pouvez vérifier l'absence (ou la présence) du support smartcard en essayant l'option **-I 0**. Si le message **no support for smartcards** s'affiche, point de support.

Pour garder une cohérence dans votre distribution, il est fortement recommandé d'éviter soigneusement les choses barbares comme **./configure && make && make install**. Mieux vaut donc récupérer les sources du paquet pour votre distribution et modifier les règles de construction. Sur Debian, ceci revient à utiliser les commandes suivantes :

```
% apt-get build-dep openssh-client
% apt-get install libopensc-dev libopenct1-dev
% apt-get source openssh-client
cd openssh-5.1p1
```

Vous modifierez ensuite le fichier **debian/rules** de manière à ajouter, dans les options de configuration des sources, la mention **--with-opensc**. Il suffit, ensuite, de reconstruire un paquet binaire avec **dpkg-buildpackage -b -rfakeroott**, puis de l'installer avec **dpkg -i**. À ce moment, le comportement de l'outil avec l'option **-I 0** doit être différent.

Nous préparons ensuite le serveur pour une première tentative de connexion en commençant par extraire la clef publique au format SSH avec :

```
% pkcs15-tool --read-ssh-key 45 | \
grep ssh-rsa >> ~/.ssh/authorized_keys
```

45 est ici l'ID de la clef qui apparaît lorsqu'on liste les clefs publiques avec **pkcs15-tool --list-public-keys**. Nous partons du principe que la première tentative de connexion se fera sur l'hôte local, d'où la redirection vers le fichier **authorized_keys**. Cette première tentative est normalement, invariablement, un échec. En effet, aucun système n'est intégré dans le client OpenSSH permettant de demander le code PIN pour accéder à la clef privée sur le token.

Deux solutions s'offrent à vous : soit passer par un agent SSH, soit modifier sensiblement les sources d'OpenSSH afin de demander le code PIN. Dans cette seconde éventualité, les sources du paquet **opensc** contiennent un patch (**opensc-0.11.4/src/openssh/ask-for-pin.diff**) qu'il faut appliquer avant reconstruction du paquet client OpenSSH.

Après réinstallation, vous devez être en mesure de vous connecter sans le moindre problème :

```
% ssh -I 0 localhost
Enter PIN for Private Key: ****
```

3.4 Firefox

La gestion des certificats placés sur une smartcard n'est pas réservée à OpenSSH. Toutes les applications capables de supporter SSL/TLS qui sont relativement bien conçues peuvent profiter du support smartcard. C'est le cas, par exemple, de votre navigateur Firefox. Attention, nous ne parlons pas ici de chiffrement simple, comme il est relativement facile d'en mettre en place, mais d'authentification du client auprès du serveur. Nous détaillerons ici la procédure complète comprenant, bien sûr, la mise en œuvre du protocole HTTPS.

À des fins d'expérimentation, nous commençons donc par créer une autorité de certification. Il est également possible de reposer sur une autorité externe. Générons donc, tout d'abord, un certificat et une clef privée pour l'AC :

```
% openssl req -x509 -newkey rsa:1024 \
-days 3650 -keyout ca.pem -out ca.crt
```

Nous obtenons ainsi les fichiers **ca.pem** et **ca.crt** qui sont respectivement la clef privée de l'AC et son certificat auto-signé. On s'attache ensuite à créer une clef privée et une demande de certificat pour notre serveur Web :

```
% openssl req -newkey rsa:1024 -days 3650 \
-keyout serveur.pem -out serveur.req
```

Attention, le Common Name doit être le FQDN (*Full Qualified Domain Name*) du serveur Web. Ceci n'est en rien spécifique à l'utilisation d'une smartcard, mais nécessaire à la mise en œuvre d'HTTPS. L'étape suivante, comme vous vous en doutez, est la signature de la demande de certificat par l'AC :

```
% openssl x509 -req -in serveur.req \
-out serveur.crt -sha1 -CA ca.crt \
-CAkey ca.pem -CAcreateserial -days 3650
```

Nous obtenons ainsi le fichier **serveur.crt** qui est le certificat du serveur Web. Nous copions ensuite les fichiers nécessaires dans un emplacement accessible par le serveur Web (**/etc/apache2/ssl**).

Puis, activez **mod_ssl**. N'oubliez pas d'ajouter ensuite **Listen 443** dans **ports.conf**, afin que le serveur soit à l'écoute du port traditionnellement associé au protocole HTTPS. Enfin, créez un nouveau fichier **sites-available/ssl**. Ce fichier contiendra quelque chose comme :

```
NameVirtualHost *:443
<VirtualHost *:443>
<IfModule mod_ssl.c>
    SSLEngine on
    SSLCACertificateFile /etc/apache2/ssl/ca.crt
    SSLCertificateFile /etc/apache2/ssl/serveur.crt
    SSLCertificateKeyFile /etc/apache2/ssl/serveur.pem
</IfModule>
DocumentRoot /var/www/ssl
<Directory />
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
ErrorLog /var/log/apache2/SSLError.log
LogLevel warn
CustomLog /var/log/apache2/SSLaccess.log combined
ServerSignature On
</VirtualHost>
```

Les directives importantes sont :

- **SSLEngine on** permettant d'activer le moteur SSL pour cet hôte virtuel associé au port 443 ;
- **SSLCACertificateFile** qui désigne le fichier contenant le certificat de l'AC ;
- **SSLCertificateFile** qui précise le certificat du serveur ;
- **SSLCertificateKeyFile** qui renseigne sur le fichier contenant la clef privée.

La configuration présentée jusqu'ici est celle classiquement utilisée pour créer un serveur HTTPS. Vous pouvez d'ores et déjà vous connecter au serveur Apache et vérifier le bon fonctionnement de l'ensemble. Un message d'avertissement vous signale que l'autorité de certification est inconnue et que le certificat serveur n'est peut-être pas de confiance. Réglez le problème tout simplement en important le certificat de l'AC dans le navigateur. Vous pouvez importer le certificat au format PEM tel que produit par défaut par OpenSSL via les préférences du navigateur.

Penchons-nous maintenant sur le côté client. Encore une fois, générerons une demande de certificat et sa clef privée :

```
% openssl req -newkey rsa:1024 -days 3650 \
-keyout client.pem -out client.req
```

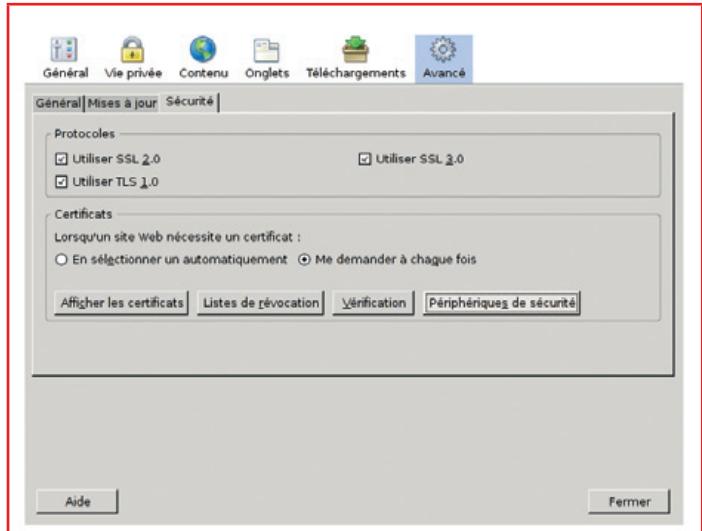
Empressons-nous ensuite de charger la clef privée dans la smartcard comme nous l'avons déjà fait pour OpenSSH :

```
% pkcs15-init -S client.pem -a 1 \
-u sign,decrypt --split-key
```

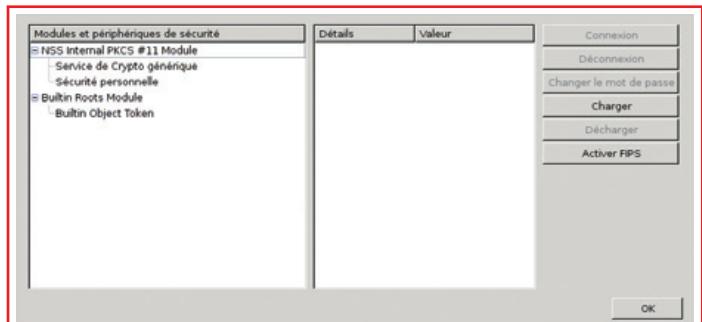
L'autorité de certification signe ensuite la demande :

```
% openssl x509 -req -in client.req \
-out client.crt -sha1 -CA ca.crt \
-CAkey ca.pem -CAcreateserial -days 3650
```

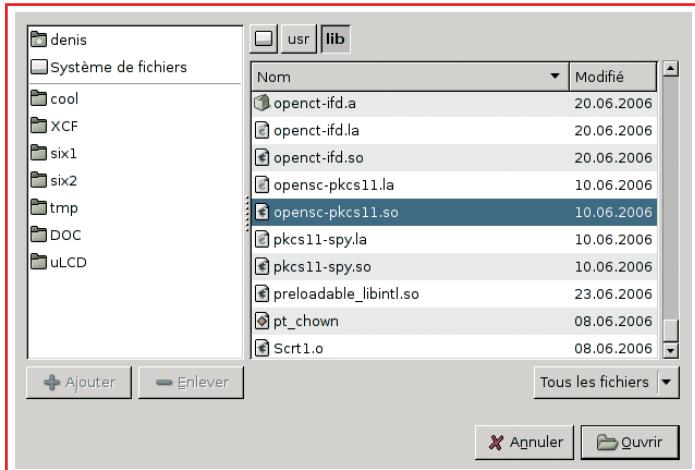
Enfin, chargeons le certificat signé dans la smartcard avec **pkcs15-init -X client.crt -v -a 1**. Notre smartcard est maintenant prête. Dernière opération pour le client et son navigateur, renseigner Firefox sur la façon d'accéder aux informations. Il nous suffit d'utiliser le pilote PKCS#11 fourni par OpenSC tout comme nous l'avons fait pour OpenSSL précédemment. C'est dans les préférences que nous trouverons un bouton « Périphériques de sécurité ». Là également, nous pouvons décider si Firefox décidera seul du certificat à utiliser ou s'il nous demandera notre avis à chaque fois.



Après avoir cliqué sur le bouton, on remarquera qu'il existe déjà un certain nombre de modules chargés. Ce sont les modules internes de Firefox permettant d'utiliser un certificat stocké localement et non sur une smartcard.



Pour prendre en charge le middleware et donc l'accès à la smartcard, il nous suffira de charger la bibliothèque partagée **opensc-pkcs11.so** qui n'est autre que notre pilote PKCS#11.



Dès chargement, on voit immédiatement apparaître les informations concernant notre eToken utilisé pour les essais. À ce stade, notre navigateur est prêt et nous changeons alors la configuration du serveur. Il nous suffit d'ajouter une simple directive pour demander un certificat signé de notre AC de la part du client et accepter ou non la connexion en fonction de sa vérification avec **SSLVerifyClient require**. Dès que le navigateur tentera de se connecter au serveur, la demande de certificat sera faite et Firefox tentera d'accéder à la smartcard. De ce fait, une boîte de dialogue nous demandera le code PIN. Si nous avons demandé une sélection manuelle du certificat, Firefox nous présentera une nouvelle fenêtre. Dès le choix validé, nous obtenons la page Web souhaitée. Client et serveur sont sûrs de leur identité respective et la communication sera chiffrée avec une clef de session partagée négociée. Le tour est joué.

3.5 OpenVPN

Seules les versions récentes d'OpenVPN supportent l'utilisation d'une smartcard ou d'un token. L'actuelle version stable, 2.0.9, ne fonctionnera pas avec les explications qui suivent. Les essais ont été réalisés avec une version 2.1rc9 telle qu'empaquetée pour Debian Testing.

Nous passerons ici sur la mise en œuvre à proprement parler d'OpenVPN. La gestion des clefs et des certificats est similaire à celle utilisée à la fois pour OpenSSH et Firefox. Sachez simplement que tout repose sur l'autorité de certification. C'est en effet sa signature qui détermine la validité d'un certificat et donc l'autorisation de connexion au VPN de la part d'un client. Vous devez donc produire un certificat auto-signé pour l'AC, puis des paires clef privée/demande de certificat pour chaque client. Côté serveur, la configuration ressemblera à ceci (classique) :

```
local 9.851.6.814
port 1818
proto udp
dev tap
mode server
ifconfig 192.168.25.1 255.255.255.0
keepalive 10 60
```

```
client-config-dir /etc/openvpn/ccd
ccd-exclusive
client-to-client
push "route-gateway 192.168.25.1"
tls-server
dh /etc/openvpn/dh1024.pem
ca /etc/openvpn/LefinnoisCA-cacert.pem
cert /etc/openvpn/serveur.crt
key /etc/openvpn/serveur.pem
crl-verify /etc/openvpn/LefinnoisCA-crl.pem
reneg-sec 21600
comp-lzo
```

client-config-dir précise un répertoire dans lequel nous trouvons des fichiers nommés d'après le champ CN des certificats qui vont être présentés pour l'authentification (les espaces sont remplacés par des _). Dans les fichiers, on trouve une simple ligne permettant d'attribuer une adresse au client, comme **ifconfig-push 192.168.25.12 255.255.255.0**.

Un fichier de configuration classique pour un client du VPN ressemblera à :

```
client
remote 9.851.6.814 1818
dev tap
tls-client
tls-remote Lefinnois_VPN
ca /etc/openvpn/LefinnoisCA-cacert.pem
cert /etc/openvpn/home.crt
key /etc/openvpn/home.pem
reneg-sec 21600
comp-lzo
```

Lorsque nous nous penchons sur la smartcard (token), nous procédons exactement comme en début d'article en important les fichiers dans le token avec **pkcs15-init -S token.pem -a 1 -u sign,decrypt --split-key**, puis **pkcs15-init -X token.crt -v -a 1**.

OpenVPN n'intègre pas directement les éléments nécessaires au dialogue avec un token ou une smartcard. À l'instar de la commande **openssl** (voir plus haut), il peut charger un provider qui fournit une interface PKCS#11. Le provider PKCS#11 est installé en même temps qu'OpenSC. C'est **/usr/lib/opensc-pkcs11.so** (oui, comme pour Firefox). Ainsi, la première chose à faire est de s'assurer que tout cela fonctionne bien :

```
% openvpn --show-pkcs11-slots /usr/lib/opensc-pkcs11.so
Provider Information:
cryptokiVersion: 2.11
manufacturerID: OpenSC Project
flags: 0

The following slots are available for use with this provider.
Slots: (id - name)
0 - Aladdin eToken PRO
1 - Aladdin eToken PRO
2 - Aladdin eToken PRO
3 - Aladdin eToken PRO
[...]
```

Seul le premier slot nous intéresse. Nous avons besoin des informations sur le token, telles que vues par OpenSSL et le provider PKCS#11. Voyons les objets disponibles dans ce premier slot :

```
% openvpn --show-pkcs11-objects /usr/lib/opensc-pkcs11.so 0
PIN: ****
Token Information:
label: OpenSC Card (denis)
manufacturerID: OpenSC Project
model: PKCS #15 SCard
```

```
serialNumber: 2221AA071518  
flags: 0000040c
```

```
You can access this token using  
--pkcs11-slot-type "label" \  
--pkcs11-slot "OpenSC Card (denis)" options.
```

openvpn est fort sympathique, il nous explique comment, en ligne de commande, accéder directement à ce slot/token et liste ensuite les objets contenus à cet emplacement. Nous obtenons ainsi toutes les informations utiles pour construire notre fichier de configuration :

```
client  
remote www.lefinnois.net 4242  
dev tap
```

```
tls-client  
tls-remote Lefinnois_VPN  
ca /etc/openvpn/LefinnoisCA-cacert.pem  
reneg-sec 21600  
comp-lzo  
verb 2  
pkcs11-cert-private 1  
pkcs11-providers /usr/lib/openssl-pkcs11.so  
pkcs11-id "OpenSC Project/PKCS \\x2315  
SCard/2221AA071518/OpenSC Card (denis)/45"  
reneg-sec 20
```

Dès le lancement du démon **openvpn** en ligne de commande **openvpn --config token.conf** ou via le service d'Init Sys V, on nous demande le PIN pour accéder à la clef privée et la connexion se fait comme avec des certificats sous forme de fichiers.

4

Conclusion

Comme nous venons de le voir, la mise en œuvre des lecteurs et l'exploitation des smartcards sont des choses relativement aisées sous GNU/Linux. On peut ainsi augmenter sensiblement le niveau de sécurité de son infrastructure sans difficultés particulières. Malheureusement, pour une petite structure, il est encore relativement difficile d'acquérir des smartcards et des tokens sans passer par un revendeur qui n'hésitera pas à insister lourdement pour vous vendre sa solution logicielle hors de prix et habituellement pour un système propriétaire. C'est vraiment dommage, car les réseaux de distribution directe (boutiques en ligne, etc.) représenteraient une véritable avancée pour la popularisation de ces solutions, et ce, quel que soit le système d'exploitation utilisé. Alors, la conclusion se résumera à un message dans un patois que j'espère intelligible

pour les fabricants et les vendeurs de smartcards et tokens : « Messieurs, il y a un gros potentiel de marché B to C dans ce secteur. Réagissez ! ».

AUTEUR : DENIS BODOR



Rédacteur en chef de GLMF. Utilisateur GNU/Linux depuis 1994. Randonneur du jardin magique.



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 œuvre 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 ITBPC07491YCW



*Qu'est-ce qu'une smartcard ?
Un périphérique contenant un processeur cryptographique et un peu de mémoire, le tout répondant à un ensemble de standards.*

*Est-il possible d'obtenir un résultat similaire avec une simple clef USB ?
Presque.*

DÉFINITION : ONE TIME PAD

Pour renforcer la sécurité et ne pas rendre l'authentification uniquement dépendante du numéro de série de la clef, un système OTP (*One Time Pad*) est utilisé par défaut. Un pad est un secret partagé entre la machine hôte et le périphérique USB. Cela prend la forme d'un fichier contenant une série de valeurs aléatoires. Le pad est présent des deux côtés et est comparé à intervalle régulier. Voilà pourquoi le module PAM utilise **pmount**.

Sur la clef USB, le fichier est stocké dans un répertoire **.pamusb** et sur le système hôte dans **~/.pamusb**. Ainsi, en cas de vol de la clef, vous pouvez la révoquer en supprimant tout simplement le fichier de votre système ou en y plaçant une autre valeur. Le voleur ne pourra pas s'authentifier sur votre machine, même en disposant de la clef avec le bon numéro de série.

Si vous vous mélangez les crayons dans les identités, ou en cas d'erreur de manipulation, supprimez simplement les deux répertoires (hôte et clef) et relancez **pamusb-check**.

PAM + USB, l'authentification matérielle du pauvre

Auteur : Denis Bodor

Bon nombre d'utilisateurs stockent leurs paires de clefs (**.gnupg** ou **.ssh**) sur une clef USB (un memory stick). Ainsi, quel que soit l'endroit où l'on se trouve, nos données cryptographiques sont à portée de main. La clef peut être perdue ou volée, mais, surtout, elle peut être copiée. Même si les clefs privées sont protégées par une phrase de passe, l'absence d'authentification matérielle et l'accès direct aux données rendent ce système fragile.

Cependant, une clef de stockage USB peut faire office de méthode d'authentification renforcée sous la forme d'un module PAM. Rappelons que PAM ou *Pluggable Authentication Module*, est un système modulaire permettant de choisir pour chaque application et service compatible (**Login**, **xlock**, GDM, etc.) la ou les méthode(s) d'authentification de son choix. Le tout via la simple édition de fichiers dans **/etc/pam.d/**.

Ce module propose un système d'authentification reposant principalement sur le numéro de série des clefs USB. En effet, chaque périphérique de ce type intègre un certain nombre de valeurs qui le définissent. Un simple **lsusb -v** vous en donne la preuve :

```
idVendor      0x08ec M-Systems Flash Disk Pioneers
idProduct     0x0008
bcdDevice     1.00
iManufacturer 1 Intuix
iProduct      2 DiskOnKey
iSerial       3 0DB15550F3830A3A
```

Nous avons là une clef USB de 128 Mo de marque Intuix identifiée par le couple **0x08ec/0x0008** et ayant pour numéro de série **0DB15550F3830A3A**. Cette dernière valeur est unique. Attention cependant, ne confondez pas difficulté et impossibilité. Le numéro de série d'une clef peut être falsifié. Cette opération n'est certes pas aisée, mais elle est possible pour n'importe quelle personne maîtrisant le fonctionnement des protocoles USB et ayant un certain niveau en électronique. Nous sommes loin, très loin, de la sécurité offerte par une smartcard cryptographique.

La mise en œuvre de ce module est simple. Après installation des paquets **libpam-usb** et **pamusb-tools**, vous pouvez, en tant qu'administrateur, utiliser la commande **pamusb-conf** pour prendre en charge le périphérique connecté :

```
% pamusb-conf --add-device intuix
Please select the device you wish to add.
* Using "Intuix DiskOnKey
          (Intuix_DiskOnKey_0DB15550F3830A3A)" (only option)

Which volume would you like to use for storing data ?
* Using "/dev/sdb1
          (UUID: b3d3bf7e-08dd-434e-9eb7-9150388e33c6)" (only option)
```

```
Name      : intuix
Vendor   : Intuix
Model    : DiskOnKey
Serial   : Intuix_DiskOnKey_0DB15550F3830A3A
UUID     : b3d3bf7e-08dd-434e-9eb7-9150388e33c6

Save to /etc/pamusb.conf ?
[Y/n]
Done.
```

Un fichier XML **/etc/pamusb.conf** sera créé, contenant les informations sur la clef USB. Si plusieurs périphériques utilisables sont connectés, le choix vous sera permis. Le nom de périphérique utilisé en argument sur la ligne de commande peut être choisi arbitrairement. C'est l'identifiant qui sera utilisé dans le fichier de configuration.

Il faut ensuite passer à la configuration des utilisateurs. L'ajout se fera également via **pamusb-conf** :

```
% pamusb-conf --add-user denis
Which device would you like to use for authentication ?
* Using "intuix" (only option)

User      : denis
Device    : intuix

Save to /etc/pamusb.conf ?
[Y/n] Y
Done.
```

Vous pouvez ajouter autant d'utilisateurs que vous le souhaitez pour un périphérique. Notez qu'ici il n'est plus question d'accès à la clef USB elle-même. Les utilisateurs sont associés à l'identifiant précédemment spécifié.

ATTENTION

La commande **pamusb-check** utilise **pmount** pour monter le système de fichiers de la clef. Avec Debian, **pmount** ne peut être utilisé que par les membres du groupe **plugdev**. Il vous faudra ajouter les utilisateurs à ce groupe avec **usermod -a -G plugdev [utilisateur]** pour que le module fonctionne correctement.

Un petit utilitaire permet d'immédiatement vérifier le fonctionnement (en tant qu'utilisateur **denis**) :

```
% pamusb-check denis
* Authentication request for user "denis" (pamusb-check)
* Device "intuix" is connected (good).
* Performing one time pad verification...
* Regenerating new pads...
* Access granted.
```

L'absence du périphérique provoque une erreur (comme attendu) :

```
% pamusb-check denis
* Authentication request for user "denis" (pamusb-check)
* Device "intuix" is not connected.
* Access denied.
```

Nous pouvons essayer, avec un cas concret d'utilisation, la commande **sudo**. Pour cela, éditez le fichier **/etc/pam.d/sudo** et ajoutez une ligne pour le nouveau module, ce qui nous donne :

```
#%PAM-1.0
@include common-auth
@include common-account
auth required pam_usb.so
```

Editez également le fichier **/etc/sudoers** pour ajouter l'utilisateur **denis** et changer le délai d'expiration à **0** pour forcer l'authentification systématique :

```
Defaults timestamp_timeout=0
root  ALL=(ALL) ALL
denis ALL=(ALL) ALL
```

Passez ensuite au test :

```
% sudo -s
Password:
* pam_usb v0.4.2
* Authentication request for user "denis" (sudo)
* Device "intuix" is connected (good).
* Performing one time pad verification...
* Access granted.
```

Le paquet **pamusb-tools** contient également l'utilitaire **pamusb-agent**. Celui-ci permet de déclencher des actions dépendantes des événements de verrouillage et déverrouillage de la clef (insertions et retraits vérifiés). Pour profiter de ces fonctionnalités, vous devrez éditer le fichier XML de configuration et, dans la section **users**, pour un utilisateur donné, ajouter une balise **agent**. Exemple :

```
<users>
  <user id="denis">
    <device>intuix</device>
    <agent event="lock">xlock</agent>
  </user>
</users>
```

Il suffit ensuite de lancer **pamusb-agent** pour obtenir le comportement attendu :

```
% pamusb-agent
pamusb-agent[13379]: pamusb-agent up and runn'unng.
pamusb-agent[13379]: Watching device "intuix" for user "denis"
pamusb-agent[13379]: Device "intuix" has been removed,
  locking down user "denis"...
pamusb-agent[13379]: Running "xlock"
pamusb-agent[13379]: Locked.
pamusb-agent[13379]: Device "intuix" has been inserted.
  Performing verification...
pamusb-agent[13379]: Executing "/usr/bin/pamusb-check --quiet
  --config=/etc/pamusb.conf --service=pamusb-agent denis"
pamusb-agent[13379]: Authentication succeeded. Unlocking user "denis"...
pamusb-agent[13379]: Unlocked.
```

L'option **--daemon** vous permettra de lancer le programme en mode démon et donc de le détacher de la console active.

Comme vous venez de le voir, n'importe quelle clef USB pourra faire l'affaire et celles de petite taille sont légion pour quelques sous (voire gratuites). Notez qu'il existe un autre projet du même type (<http://usbauth.delta-xi.net/>) qui semble apporter son lot de fonctionnalités supplémentaires. Une autre voie à suivre si le module présenté ici ne vous apporte pas toute satisfaction.

AUTEUR : DENIS BODOR



Les étiquettes interrogables par radiofréquence (RFID, RadioFrequency IDentifier) sont utilisées, souvent à notre insu, dans de nombreuses activités quotidiennes : antivol, suivi et identification de marchandises, contrôle d'accès, bientôt les passeports... autant d'applications qui justifient de comprendre la technologie sous-jacente, ses domaines d'applications et les éventuels risques associés.

Nous allons mettre en pratique les connaissances acquises sur les puces servant au tatouage électronique des animaux de compagnie, en nous fixant pour objectif de réaliser l'ensemble de la chaîne de lecture du code identifiant un chien.

Analyse des étiquettes d'identification par radiofréquence (RFID)

Auteur : J.-M. Friedt

1

Introduction

Nous proposons d'étudier l'ensemble de la chaîne d'interrogation d'étiquettes d'identification [1, 2] interrogables par radiofréquences (RFID), du circuit électronique d'acquisition des données au traitement du signal pour finalement interpréter les données résultantes. Notre objectif est de lire le code (tatouage) contenu dans les étiquettes d'identification d'animaux de compagnie. Ces puces ont pour propriété de ne pas embarquer de source d'énergie, mais d'« absorber » une partie de l'énergie fournie par l'interrogateur pour alimenter leur circuit électronique. Il s'agit donc de dispositifs totalement passifs, de durée de vie a priori illimitée, fournissant des fonctionnalités simples telles que le stockage et la restitution d'un code d'identification. La tendance actuelle vise à compléter l'identification avec la mesure de quantités physiques [3].

Ce projet est relativement simple puisqu'il concerne les dispositifs les plus accessibles – ne contenant aucune sécurité pour restituer les informations contenues dans l'étiquette – et la principale difficulté consiste à trouver les acronymes pertinents pour focaliser les recherches sur le web sur les aspects techniques noyés dans les publicités, légendes urbaines et autres débats stériles, sans intérêt pour le domaine technique qui nous intéresse.

Parmi les trois grandes gammes de dispositifs que nous allons brièvement répertorier ici, nous ne nous intéresserons qu'à ceux fonctionnant aux fréquences les plus basses : ces dispositifs sont les plus simples à mettre en œuvre (les fréquences mises en jeu ne nécessitent aucune précaution de fabrication du circuit électronique d'interrogation) et celles utilisées dans les tatouages électroniques des animaux de compagnie. Les trois grandes classes de RFID – définies par leur gamme de fréquence de fonctionnement et donc leur domaine d'application (différentes portées, efficacité de l'antenne et pénétration de l'onde électromagnétique dans l'objet porteur du RFID) sont :

- Les dispositifs fonctionnant en basse fréquence (en dessous de 150 kHz, et en particulier 125 et 134,2 kHz). Compte tenu des longueurs d'ondes (plusieurs kilomètres), le couplage entre la puce et l'interrogateur ne peut être que par induction magnétique entre bobines. Ce couplage décroît rapidement (d^{-3}) avec la distance d , et la portée de ces dispositifs est nécessairement réduite.

- Les dispositifs fonctionnant aux fréquences dans la gamme 10-1000 MHz, avec en particulier les plages normalisées de 13,56 MHz [4, 5] et 868 MHz. L'efficacité de couplage des ondes électromagnétiques dans l'antenne est accrue, mais l'atténuation par des objets diélectriques augmente.
- Les dispositifs fonctionnant aux très hautes fréquences (au-delà du GHz, en particulier 2,45 GHz, soit des longueurs

d'onde de l'ordre de la dizaine de centimètres). L'antenne de l'interrogateur peut être très efficace – voire directive – mais la pénétration de l'onde dans les objets est pratiquement nulle : la puce interrogée doit se trouver en surface, mais le débit est d'autant plus élevé que la fréquence de porteuse et la bande passante associée sont élevées.

2

Circuit électronique d'interrogation

La réalisation du circuit va se résumer en l'exploitation d'un composant dédié à l'interrogation des RFID basses fréquences : l'Atmel U2270B [6, 7]. Ce composant, disponible pour 3,30 euros chez Farnell (référence 1095806), se charge de la partie analogique de l'interrogation : oscillateur générant la porteuse d'interrogation du RFID, amplificateur pour alimenter la bobine faisant office d'antenne, circuit de mesure de l'intensité du couplage de la bobine d'interrogation avec le RFID, signal de sortie exploitable par un microcontrôleur (Fig. 1). Nous ne nous servirons pas de la fonctionnalité de modulation de la porteuse afin d'envoyer des commandes au RFID. Dans le cas qui va nous intéresser, la porteuse est émise en continue et la modulation d'impédance vue par l'antenne est induite par le RFID lorsque celui-ci a accumulé assez d'énergie pour entrer en fonctionnement.

Les deux subtilités notables pour exploiter ce circuit sont :

- Prevoir une résistance variable pour régler la fréquence de l'oscillateur et l'ajuster à la fréquence à laquelle répond le RFID (125 ou 134,2 kHz, à ajuster en fonction de l'inductance de la bobine faisant office d'antenne).
- La sortie est en collecteur ouvert, ce qui signifie qu'en l'absence de résistance de tirage vers l'alimentation du microcontrôleur, aucun signal n'apparaît sur la broche 2 de l'U2270B. Ce point crucial n'est pas détaillé dans la datasheet, mais a été identifié en consultant <http://kudelsko.free.fr/transpondeur/presentation2.htm>.

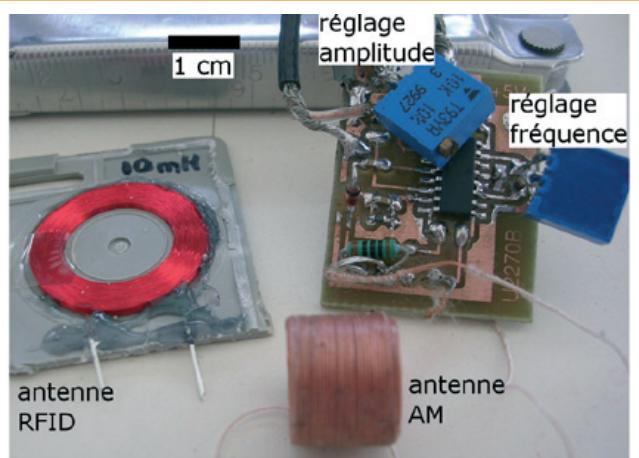
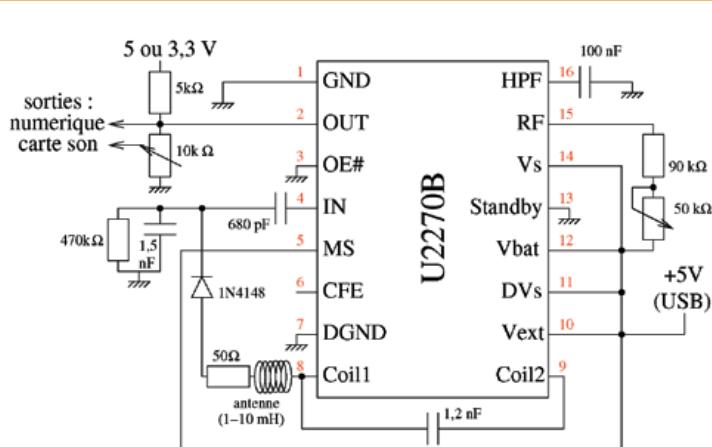


Figure 1 : La réalisation du circuit ne pose pas de problème majeur, puisque la fréquence de travail est de l'ordre de la centaine de kilohertz. Il s'agit d'un circuit simple face de petites dimensions dont la face inférieure, non visible sur cette photo, est totalement couverte de cuivre pour faire office de plan de masse. Le circuit est alimenté en 5 V, obtenu depuis un port USB pour rendre le montage mobile lorsqu'il est utilisé avec un ordinateur portable. Diverses antennes pourront être utilisées en fonction des rebuts disponibles : nous avons expérimenté avec une bobine faisant à l'origine office d'antenne de récepteur radio AM (bobine soudée au circuit, en bas), et avec une bobine extraite d'une carte d'accès RFID (bobine rouge à gauche). Le bobinage, de quelques centaines de tours, doit présenter une inductance de quelques millihenry [8].

Le bon fonctionnement du circuit est validé en observant à l'oscilloscope ou au compteur de fréquence les signaux attaquant

la bobine (broches 8 et 9) : ajuster la résistance sur la broche 15 jusqu'à atteindre la fréquence voulue de 134,2 kHz.

3

Acquisition et traitement du signal

En l'absence d'informations détaillées sur le protocole de communication du RFID avec l'interrogateur, nous désirons utiliser un outil aussi souple que possible pour acquérir et traiter les signaux. Nous allons donc acquérir les signaux

en sortie de l'U2270B sur la carte son d'un PC, pour ensuite traiter le fichier résultant et en extraire les valeurs des bits transmis. En effet, nous avons un a priori sur la forme des signaux attendus, information acquise grâce à quelques mots

clés qu'il nous faudra utiliser dans la majorité des recherches sur le web.

- Les protocoles de communication utilisés par les RFID implantés dans les animaux sont décrites dans deux standards que sont les ISO 11784 et 11785. L'obtention de ces documents est payante, mais les grandes lignes sont résumées à ref. [9].
- Ce même site web fournit le nom du protocole de communication – FDX-B – et la fréquence qui va nous intéresser pour une utilisation en Europe : 134,2 kHz. Il s'agit d'un protocole *full duplex* dans lequel l'émetteur fournit constamment de l'énergie à l'étiquette, qui transmet son information par modulation de l'impédance vue par le bobinage de l'émetteur (circuit ouvert ou fermé du côté de l'étiquette pour moduler le signal radiofréquence).
- Deux documents, refs. [10] et [11], vont nous fournir quelques informations sur la partie radiofréquence, mais surtout sur la partie numérique d'interprétation des données qui va nous intéresser par la suite (section 4).

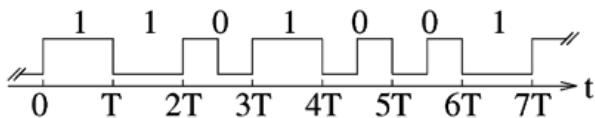


Figure 2 : Codage de Manchester

Nous apprenons dans ce document que le codage sera de type Manchester (Fig. 2) : une transition doit toujours survenir après une période d'horloge (période que nous ne connaissons pas encore), et un zéro est indiqué par deux transitions dans cette période d'horloge. Le décodage du signal issu de l'interrogateur consistera donc en une mesure de durée, nécessitant donc une base de temps stable du côté de l'interrogateur : cette fonctionnalité est fournie par la carte son.



Figure 3 : Gauche : acquisition des données. La principale difficulté de la mesure tient en l'immobilité du chien lors de la recherche de la position de l'étiquette. La portée réduite de notre circuit nécessite en effet de se placer parfaitement au-dessus de l'étiquette pour obtenir un signal stable et exploitable. Droite : exemple de signal acquis à 96 kHz, et gros plan sur une zone des points acquis présentant clairement les créneaux encodant les signaux sur les données brutes (points bleus) et après saturation des données brutes pour les convertir en valeurs binaires (traits rouges).

L'enregistrement du signal se fait par la carte son d'un Asus EEE PC 701 : afin de faciliter le traitement ultérieur du signal, nous échantillonnerons à la fréquence maximale de 96 kHz, sans nous inquiéter de l'espace disque occupé (Fig. 3). Tout logiciel d'acquisition capable de contrôler les options de la carte son conviendra : nous avons pour notre part utilisé Audacity afin de rapidement couper les segments de l'enregistrement sans intérêt et ne traiter que les parties pertinentes (Fig. 4). Les réglages sont un enregistrement à 96 kHz, mono, 16 bits/échantillons et sauvegarde au format PCM (c'est-à-dire fichier wav, sans compression, qui sera le plus simple à exploiter par la suite).

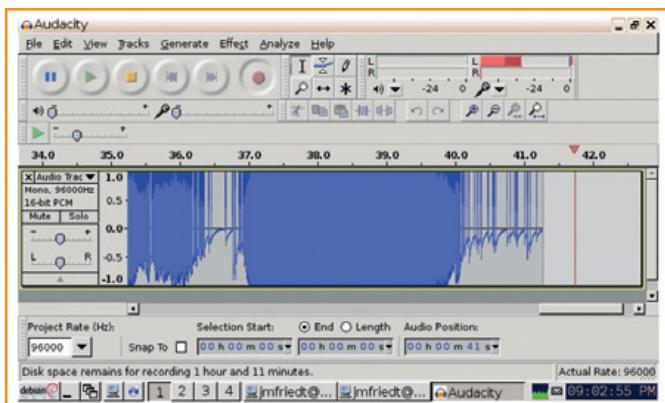
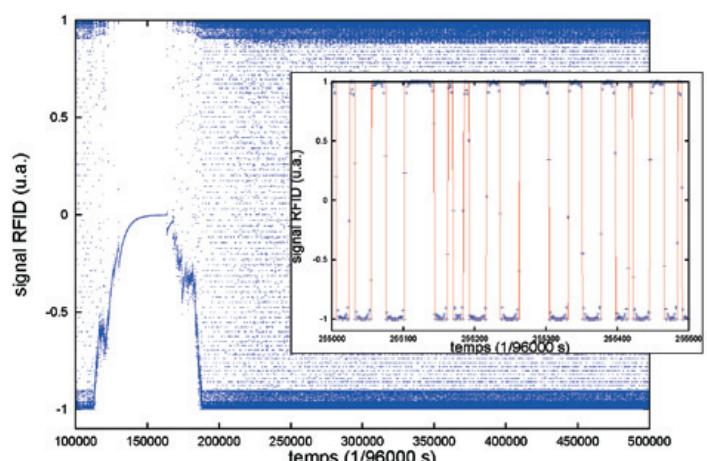


Figure 4 : Capture d'écran d'Audacity lors de l'acquisition d'un signal de RFID : le signal valable est disponible de 37 à 40 secondes, tandis que les recherches de la position de l'étiquette sur le cou du chien avant et après ces dates induisent des fluctuations d'amplitude du signal qui le rendent inexploitable.

Toute la suite des traitements se fait au moyen de GNU/Octave (version *open source* de Matlab), dont l'installation sous Debian est complétée par l'équivalent libre du *Signal Processing Toolbox* sous forme du paquet **octave-signal**.



Sous Octave, les données du fichier audio sont mises en mémoire par la fonction **wavread** qui renvoie dans **fs** la fréquence d'échantillonnage et surtout dans **a** les données ajustées pour tenir entre les bornes $[-1;+1]$:

```
[a,fs,siz]=wavread('adelie96k.wav');
x=find(a>=0.);a(x)=1;
x=find(a< 0.);a(x)=-1;
```

Le signal analogique est converti en signal binaire en saturant les valeurs négatives à -1 et les valeurs positives à $+1$, puis nous recherchons les transitions positives (variable **pos**) et négatives (variable **neg**) des créneaux en étudiant la valeur de la dérivée du signal. Ces deux ensembles de signaux, les dates des transitions positives et négatives des créneaux acquis vont être les bases sur lesquelles nous allons effectuer la suite de nos traitements.

```
pos=find(diff(a)> 0.5); % date montant
neg=find(diff(a)<-0.5); % date descendant
```

Sans savoir à quoi correspondent ces transitions, nous pouvons chercher à savoir si ces transitions sont représentatives d'un signal. Compte tenu de la longueur des mesures (plusieurs secondes), le signal transmis par le RFID doit nécessairement se répéter. Une transformée de Fourier doit donc présenter des pics correspondant au taux de répétition si le signal est périodique.

NOTE

Un signal contient N répétitions d'un motif de p points. Les $N \times p$ points au total représentent donc un signal périodique, de période p , c'est-à-dire de fréquence $1/p$. Cette fréquence caractéristique apparaît sur une transformée de Fourier sur $N \times p$ points comme un pic d'abscisse N . À titre d'illustration, prenons un signal de durée 1 s contenant un signal à 100 Hz échantillonné à la fréquence $f_{ech}=1000$ Hz. Alors $p=10$ (le signal à 100 Hz est échantillonné 10 fois par période), $N \times p=1000$ et la transformée de Fourier sur les $N \times p$ points s'étale sur un intervalle $\pm f_{ech}/2$, soit -500 Hz à $+500$ Hz. Le pic des 100 Hz se trouve à $1/5$ ème de l'abscisse maximum, soit le point d'indice $500/5=100$ qui est bien égal à N .

La transformée de Fourier est calculée sur un intervalle de temps qui semble « visuellement » intéressant, c'est-à-dire dans lequel le signal acquis par la carte son est saturé par la réponse du RFID. La transformée de Fourier (Fig. 6) est calculée sur les points 2000 à 8000 des données contenant les intervalles de temps (**pos** ou **neg**, Fig. 5), ce qui correspond aux points 240750 à 424477 dans le fichier audio original. 60 répétitions dans $(424477-240750)/96000=1,9$ s signifient qu'une réponse du RFID met 32 ms. Si nous faisons l'hypothèse d'un message de 128 bits, le débit est de 4013 bits/seconde ou 0,25 ms/bit, très proche de la valeur proposée sur la page [9] : nous sommes sur la bonne voie, nous avons été capables d'obtenir les bits individuels du message, qu'il nous faut interpréter comme un message complet.

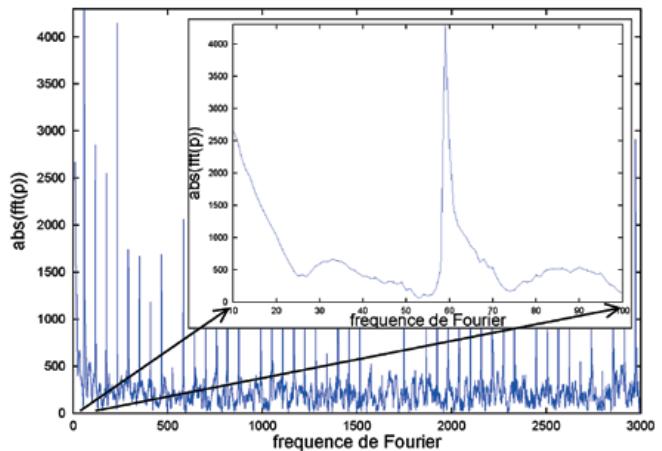


Figure 6 : Transformée de Fourier du signal pos calculé plus haut (Fig. 5), pour les points d'abscisses 2000 à 8000. Un pic d'abscisse 60 indique que l'ensemble des points sur lesquels cette opération a été effectuée contient 60 répétitions du signal. Nous supposerons que ce signal est la réponse du RFID que nous cherchons à décoder.

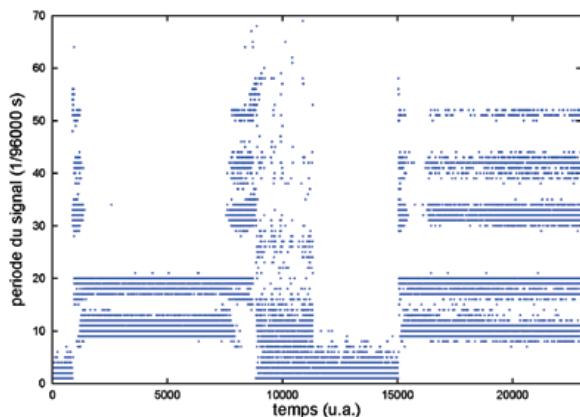
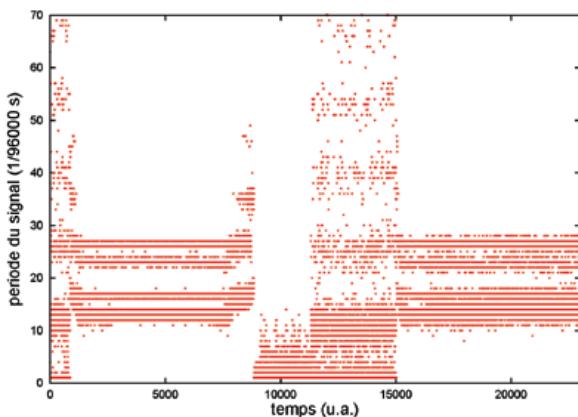


Figure 5 : Intervalles de temps entre deux transitions positives successives (droite, en bleu), et entre deux transitions négatives successives (gauche, en rouge). L'interrogateur était correctement placé entre les dates 1000 et 9000, puisque les deux distributions sont bimodales, tandis que les données de 15000 à la fin ne sont pas exploitables puisque la distribution des transitions positives (droite) n'est pas bimodale et ne peut donc pas représenter un signal en codage Manchester.

4

Interprétation des données

Ayant constaté que les données **pos** et **neg** calculées plus haut contiennent probablement une information pertinente, il nous reste à extraire cette information et l'interpréter.

Un signal périodique est formé d'alternances de fronts montants et de fronts descendants. Deux cas se présentent, selon que le premier front soit montant ou descendant :

```
pp=length(pos);pn=length(neg); % autant de fronts montant que descendant
if (pn<pp) p=pos(1:pn); clear pos;pos=p;
else p=neg(1:pp); clear neg;neg=p;
end

n=(neg-pos); % intervalle de temps entre descendant et montant
if (n(1)<0) % si negatif, c'est le mauvais sens
    n=neg(2:length(pos))-pos(1:length(neg)-1);
    p=abs(neg);
    premier=0;
else
    p=abs(pos)-neg(1:length(neg)-1);
    premier=1;
end
```

Lorsque nous calculons **neg-pos**, si la première transition est un front montant, alors la date de première transition de **pos** est inférieure à la date de première transition négative et la soustraction des tableaux ne contient que des intervalles de temps négatifs. Dans ce cas, il nous faut décaler le tableau **pos** afin de soustraire l'élément d'indice $n+1$ de **pos** à l'élément n de **neg**.

Les intervalles de temps sont alors seuillés afin de devenir des données binaires : l'histogramme de **pos** et **neg** nous indique où placer le seuil (au creux de la fonction bimodale) séparant le 1 du 0. Les deux ensembles de valeurs centrées sur 13 et 26 (alternance négative, Fig. 5 gauche) ou 10 et 20 (alternance positive, Fig. 5 droite) correspondent aux 0 et 1 respectivement, et placent le seuil vers 19 et 15 respectivement. Le fait que ces deux seuils soient différents indique que le signal en créneau initial n'est pas symétrique et que notre première étape de saturation du signal audio à ± 1 a induit cette asymétrie en prenant pour seuil 0.

```
SEUILp=19.5;SEUILn=14.5; % parametre a regler pour chaque montage

x=find(n<SEUILn);n(x)=0;
x=find((n>0)&(n<3*SEUILn));n(x)=1;

x=find(p<SEUILp);p(x)=0;
x=find((p>0)&(p<3*SEUILp));p(x)=1;
```

Dans cette opération, les seuils **SEUILn** et **SEUILp** sont des paramètres fondamentaux, a priori constants puisque fixés par la période T (Fig. 2), mais que nous avons constaté devoir adapter en pratique lors de chaque nouvelle acquisition.

Nous intercalons dans la variable **total** les intervalles de temps des créneaux positifs et des créneaux négatifs pour ainsi former la séquence du code de Manchester.

```
total=zeros(length(p)+length(n),1);
if (premier==1)
    total(1:2:2*length(n))=n;
    total(2:2:2*length(p))=p;
else
    total(1:2:2*length(p))=p;
    total(2:2:2*length(n))=n;
end
```

Chaque zéro est indiqué par deux transitions courtes successives. Il nous faut donc remplacer toute paire de 0 par un zéro unique. Par ailleurs, la présence dans les données initiales d'un zéro unique est indicateur d'une erreur de décodage : ce cas ne peut théoriquement jamais survenir.

```
sortie=zeros(length(total),1);

sta=5000;sto=7000; % plage de travail : debut et fin
pp=1;
k=sta*2+1;

while (k < sto*2) % elimine les "0" en double
    if (total(k)==1) % (un zero = deux transitions courtes)
        sortie(pp)=1;pp=pp+1;k=k+1;
    else
        pp=pp+1;k=k+1;
        if (total(k)==0) disp('erreur');k
        else k=k+1;
        end
    end
end

s=sortie(1:pp)' % afficher le resultat
save -text s s % attention, PAS compatible Matlab
```

En exécutant ces routines sur deux séries de points acquis sur deux chiens différents, nous obtenons les deux séries suivantes qui ont été validées par la présence de doublets de 0 dans **total**.

```
... 000010000001110000110111110001000000001000000001000000001
0000000001001000101100111111010011010001011101111101011110010000000010000000
11100001101111110001000000001000000001000000001
0000000001001000101100111111010011010001011101111101011110010000000010000000
11100001101111110001000000001000000001000000001
0000000001001000101100111111010011010001011101111101011110010000000010000000
11100001101111110001000000001000000001000000001 ...
...
```

et

```
... 11111101011101001110111101101111001000000010000000
1100011010110000101100000001000000001000000001
000000000110010111011111110101101001110111101011110010000000010000000
11000110101000101100000001000000001000000001
00000000011100101111011111110101110100111101011110010000000010000000
1100011010100010110011111110101110100111101011110010000000010000000
00000000011100101111011111110101110100111101011110010000000010000000
11000110101000101100000001000000001000000001 ...
...
```

```
0000000000111001011101111011111111010111010011101111101011111001000000010000000  
1100011010110000101100000001000000001000000001  
000000000011100101110111111110101110100111011111101011111101000000010000000  
110001010110000101100000001000000001000000001  
00000000001110010111011111111010111010011110110111110100000001000000001  
1100011010110000101100000001000000001000000001  
00000000001110010111011111111010111010011110110111110100000001000000001  
1100011010110000101100000001000000001000000001 ..
```

Ces données ont déjà été découpées pour les faire commencer par l'en-tête **000000000001** qu'on ne peut trouver qu'en début de trame. Nous constatons que les séquences se répètent et font bien 128 bits de longueur.

5

Calcul du code résultant

La série de bits obtenue précédemment s'interprète au moyen des informations fournies dans [11] :

- un en-tête formé de 10 zéros suivis de 1 un, suite unique de chiffres indiquant de façon univoque le début d'une trame ;
- 72 bits de données contenant l'identifiant qui va nous intéresser ;
- 18 bits de CRC ;
- 27 bits de données étendues.

- Les trois premiers chiffres sont le code du pays selon ISO 3166-1, 250 pour la France.
- Deux chiffres pour identifier la race de l'animal, avec **26** pour les chiens.
- Deux chiffres pour identifier le fabricant, en général pour les chiens un nombre entre **96** et **98**.
- Finalement, les 8 derniers chiffres identifient l'animal.

```
0000000001  
header v v v v v v v v  
00100010110011111101001101100010111011111 0101111100 1 0 00000010000000 1 1  
64-27 = Code d'identification national pays (ISO) 16 animal  
v v v v v  
100001101111110001 000000010000000100000001  
CRC data stream : que des 0 sépare de 1
```

Les symboles **v** identifient les 1 qui sont insérés dans la trame pour garantir que l'en-tête (incluant ses 10 zéros consécutifs) ne se retrouve nulle part dans le corps du message.

Ces « 1 » ne sont pas inclus dans le message et doivent être éliminés. Il nous reste donc un code de pays **0101111100** et un identifiant national **0010001010001111110100011010001011011111**. En prenant le bit de poids le plus faible à gauche et l'octet de poids le plus faible à gauche, ces valeurs binaires sont égales à **0xFA** et **0x3ED165F944**.

Le code national de la France est **250=0xFA** tel que défini dans la norme ISO 3166-1 [12].

L'identifiant national est égal en décimal à **269801093444**.

Le code d'identification fourni sur le passeport de ce chien est **250269801093444** : il y a bien concordance entre cette valeur et la concaténation du code de pays suivi de l'identifiant national que nous venons de calculer.

À titre d'exercice, le lecteur pourra confirmer que le second code fournit bien le même code de pays, tandis que l'identifiant national du second chien est **269700030163**.

Ces informations sont cohérentes avec la structure des tatouages électroniques, exprimée en décimal :

The image shows the cover of a magazine titled "MISC HORS-SÉRIE N°2". The main title is "SPÉCIAL CARTES À PUCE". The cover features several small images related to card technology, including a JavaCard chip, a YenCard, and a credit card. Text on the cover includes "2 NOV/DEC. 2008", "PROGRAMMATION JavaCard ou comment programmer une vraie carte à puce", "DOSSIER CARTES À PUCE DÉCOUVREZ LEURS FONCTIONNALITÉS ET LEURS LIMITES", "HISTORIQUE Retour sur la YenCard, un aperçu du système bancaire français", "SÉCURITÉ Mise en place d'infrastructures de sécurité de clés (PKI) utilisant des cartes à puce", and "TECHNOLOGIES MIFARE classic, comment une faille compromet la sécurité de milliards de cartes !". To the right of the magazine cover, there is promotional text: "Disponible dès le 14/11/2008*", "chez votre marchand de journaux", and "Comprenez et utilisez ces technologies pour assurer votre sécurité !". At the bottom, it says "Visitez www.ed-diamond.com pour en savoir plus !" and "sous réserve de toutes modifications".

6

Conclusion

Nous avons présenté pas à pas une démarche pour identifier la réponse d'une étiquette passive communiquant par radiofréquence (RFID tag). Nous avons identifié la fréquence de travail (134,2 kHz) et le codage de type Manchester, en accord avec les informations accumulées sur le protocole de communication avec les étiquettes implantées dans les animaux pour leur identification (FDX-B). Nous avons acquis avec du matériel communément disponible (PC équipé d'une carte son) des signaux caractéristiques de la réponse de l'étiquette, et avons décodé le contenu de cette information pour finalement trouver la valeur attendue pour l'identification de deux chiens.

Cette démarche mérite d'être étendue à d'autres types d'étiquettes [14], notamment celles équipant les badges d'accès ou les futurs passeports dits « biométriques ». Nous n'avons fait qu'effleurer un domaine riche en interrogeant une puce sans protection répondant systématiquement à toute sollicitation. Nous avons expérimenté avec une carte d'accès qui répond à la même fréquence, sans avoir cherché à en décoder le signal numérique. Afin d'étendre la gamme d'environnements dans laquelle le circuit d'interrogation fonctionne, les schémas plus complexes séparant l'alimentation de la bobine (jusqu'à 12 V) de la partie numérique (5 V) et effectuant une rétroaction de la fréquence reçue de l'étiquette sur la fréquence d'excitation sont fournis dans la datasheet de l'U2270B ([7], applications 2 et 3).

Finalement, nous n'avons jamais communiqué d'ordre à l'étiquette – option disponible en commandant la broche 6 (CFE) de l'U2270B sous contrôle d'un microcontrôleur afin de moduler la porteuse. Le circuit présenté ici mériterait d'être connecté à un microcontrôleur afin d'automatiser la phase de lecture de l'étiquette sans avoir à développer toute la procédure sous Octave présentée ici.



Adélie et Cézanne sont deux petits lévriers italiens, matricules **250269700030163** et **250269801093444** respectivement. Munies de puces d'identification radiofréquence, elles désiraient appréhender

les technologies de transfert des informations afin d'en comprendre les implications pour leur vie privée. Elles ont été assistées dans ce projet par J.-M. Friedt, membre de l'association Projet Aurore de Besançon et ingénieur dans la société SENSeOR.

RÉFÉRENCES

- [1] La traduction de RFID tag est inspirée de <http://fr.wikipedia.org/wiki/Radio-identification>
- [2] RFID, instrument de sécurité ou de surveillance, *MISC* 33, septembre/octobre 2007.

- [3] SMITH (J. R.), SAMPLE (A.P.), POWLEDGE (P.S.), ROY (S.) & MAMISHEV (A.), « *A Wirelessly-Powered Platform for Sensing and Computation* », Springer, Lecture Notes in Computer Science 4206, 2006, pp. 495-506 ou OPASJUMRUSKIT (K.), THANTHIPWAN (T.), SATHUSEN (O.), SIRINAMARATTANA (P.), GADMANEE (P.), POOTARAPAN (E.), WONGKOMET (N.), THANACHAYANONT (A.) & THAMSIRIANUNT (M.), « *Self-Powered Wireless Temperature Sensors Exploit RFID Technology* », IEEE Pervasive Computing 5 (1), 2006, pp. 54-61.
- [4] Une plateforme *open source* pour l'interrogation d'étiquettes HF : <http://2008.rmll.info/CRESITAG-Plateforme-opensource.html>. Cette présentation contient quelques références en français qui n'ont pas été consultées au cours de notre étude.
- [5] RYAN (R.), ANDERSON (Z.) & CHIESA (A.), *Anatomy of a Subway Hack*, DEFCON 16, 2008, disponible à http://www.tech.mit.edu/V128/N30/subway/Defcon_Presentation.pdf
- [6] FINKENZELLER (K.), *RFID Handbook – Fundamentals and Applications in Contactless Smart Cards and Identification, Second Edition*, John Wiley & Sons, 2003.
- [7] www.atmel.com/dyn/resources/prod_documents/doc4684.pdf
- [8] www.datasheetcatalog.org/datasheet/Temic/mXyzutqu.pdf
- [9] http://en.wikipedia.org/wiki/ISO_11784_%26_11785
- [10] Une présentation très complète sur le projet www.rfidiot.org, contenant notamment des informations sur les codages de l'information transmise par un RFID : <http://www.blackhat.com/presentations/bh-europe-07/Laurie/Presentation/bh-eu-07-laurie.pdf>
- [11] http://www.emmicroelectronic.com/webfiles/ref/h4005_ds.pdf décrit en détail les « 1 » ajoutés dans le signal à décoder pour ne pas retrouver l'en-tête dans les données, ainsi que l'organisation des 128 bits du message.
- [12] en.wikipedia.org/wiki/ISO_3166-1
- [13] HUNT (V. D.), PUGLIA (A.) & PUGLIA (M.), *RFID – A guide to radio frequency identification*, John Wiley & Sons, 2007 est un ouvrage ne contenant aucune information technique, que je ne mentionne ici que pour éviter au lecteur de gaspiller son argent.
- [14] GRAAFSTRA (A.), *RFID Toys: 11 Cool Projects for Home, Office and Entertainment*, Wiley ExtremeTech, 2006 se contente d'exploiter des lecteurs et étiquettes commerciaux, sans en expliquer le fonctionnement. Le seul chapitre intéressant est le premier, disponible gratuitement sur amazon.com. À éviter donc !

Avez-vous l'âme du collectionneur ?

**Boostez
votre
collection !**

LES 4 FAÇONS DE COMMANDER !

Par courrier

En nous renvoyant ce bon de commande.

Par le Web

Sur notre site : www.ed-diamond.com.

Par téléphone

(paiement C.B.) entre 9h-12h & 15h-18h au **03 88 58 02 08.**

Par fax

Au **03 88 58 02 09** C.B. et/ou bon de commande administratif.

Numéros GNU/Linux Magazine épuisés

N°01, N°02, N°03, N°04, N°05, N°08, N°20, N°21,
N°23, N°31, N°33, N°42, N°43, N°45, N°47, N°54 et
N°90

du 6 Février 2017

BON DE COMMANDE POWER PACKS à REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

Diamond éditions - GNU/Linux Magazine - BP 20142 - 67603 SÉLESTAT Cedex

Cochez ici ▲ POWER PACKS X5	OUI, je désire acquérir un POWER PACK X5		
	1 ^{er} 1PP* X5	2 ^{ème} 2PP* X5	3 ^{ème} 3PP* X5
	I, GNU/Linux Magazine N°		
	2, GNU/Linux Magazine N°		
	3, GNU/Linux Magazine N°		
	4, GNU/Linux Magazine N°		
5, GNU/Linux Magazine N°			
Total par série de POWER PACKS X5 :	15 €	30 €	45 €
Cochez ici ▲ POWER PACKS X10	OUI, je désire acquérir un POWER PACK X10		
	1 ^{er} 1PP* X10	2 ^{ème} 2PP* X10	3 ^{ème} 3PP* X10
	I, GNU/Linux Magazine N°		
	2, GNU/Linux Magazine N°		
	3, GNU/Linux Magazine N°		
	4, GNU/Linux Magazine N°		
	5, GNU/Linux Magazine N°		
	6, GNU/Linux Magazine N°		
	7, GNU/Linux Magazine N°		
	8, GNU/Linux Magazine N°		
9, GNU/Linux Magazine N°			
10, GNU/Linux Magazine N°			
Total par série de POWER PACKS X10 :	25 €	50 €	75 €
TOTAL :			
Frais de port : +3,81€			
TOTAL :			
Les hors-séries et numéros spéciaux sont exclus des POWER PACKS. Montant TOTAL 15€ + 3,81€ de frais de port . Le TOTAL s'élève à 18,81€ pour l'achat d'un POWER Pack x5. SEULEMENT EN FRANCE MÉTROPOLITAINE !			
*PP = POWER PACK			

Choisissez vos numéros dans le tableau ci-dessous*

* Seuls les numéros ci-dessous sont disponibles pour une commande de Power Packs par x5 et x10

N°06 GNOME - The Gimp	N°49 Après MySQL & PostgreSQL SAP DB : La base de données libre & puissante	N°81 Comment fonctionnent les générateurs de nombres pseudo-aléatoires
N°07 Dopez Linux	N°50 Créez un album Photo avec PHP ...et sans MySQL	N°82 eCos, une solution libre pour systèmes embarqués
N°09 Prêt pour le jeu !	N°51 Boostez votre site Web avec XML grâce à XSLT, CSS & XPath	N°83 Greylist Eliminez le SPAM à la racine
N°10 The HURD : 100% GNU	N°52 Linux Temps réel où en est-on aujourd'hui ?	N°84 Déploiement de hotspots Wifi sécurisés
N°11 Exclusif : l'avenir de G.N.O.M.E	N°53 Linux sur PDA : Linux dans votre poche !	N°85 Firewall : Netfilter & NuFW
N°12 NT et Linux : Guerre ou complément ?	N°55 Intelligence Artificielle : Principes & programmation de jeux de stratégie classique	N°86 Serveur SMTP: Routage des mails avec Postfix
N°13 Cryptage : la clé de la sécurité	N°56 Développez vos applications Mozilla avec XPFCE & XPCM	N°87 Le point sur Mono .NET Java et les Brevets
N°14 Xfree 4.0 : le futur à notre portée	N°57 Maitrisez la gestion... Slots & Signaux ... des événements en C++	N°88 Sécurité: Smartcards & Tokens
N°15 Passez à la vitesse supérieure	N°58 IDbnds enfin une alternative viable à BIND !	N°89 Utilisation avancée de XEN
N°16 OpenSources : Est-ce suffisant?	N°59 Zopix, Créez un CD "Live" Zope en 10 minutes !	N°90 Ajax Avancé
N°17 Linus : Système embarqué	N°60 JBoss serveur d'applications J2EE OpenSource	N°92 Paravirtualisation XEN & SLO Répartition de charge
N°18 Spécial interview : l'avenir de Linux	N°61 Découvrez MySQL 5 et les procédures stockées	N°93 Développez vos extensions Firefox
N°19 Dossier spécial : Postgre SQL 7.0	N°62 Créez votre OS, principe et implémentation	N°94 PBX Vidéo avec Asterisk
N°22 Le multi-threading : Une manière moderne de programmer le Multithéâtre	N°63 Les threads : kernel 2.6 et 2.4	N°95 Administration et configuration centralisées avec Clengine
N°24 Palm et Linux	N°64 Adamoto	N°96 Géolocalisation de photos numériques
N°25 Kernel 2.4.0	N°65 Théorie et pratique : Supervision avec Nagios	N°97 Haute-disponibilité & équilibrage de charge
N°26 <Dossier> XML </Dossier>	N°66 Créez votre Distribution Live	N°98 Supervision avec NAGIOS
N°27 Les systèmes de fichiers journalisés	N°67 C#/.NET	N°99 Java & JNI Tirez le meilleur de Java
N°28 Scripting : la force d'Unix	N°68 Le crash disque vous gêne	
N°29 LFS, Linux From Scratch	N°69 La réponse de Sun à Linux ? SOLARIS 10	
N°30 Le chiffrement des données	N°70 Découvrez et comprenez la technologie GRID	
N°32 Changez de coquille	N°71 Présentation et installation du Hurd	
N°34 XSL - FO : leX Killer ?	N°72 Services Web... C/C++ et gSOAP	
N°35 QoS et protôte : optimisation et contrôle du trafic IP	N°73 Compression librairie algorithmes et programmation	
N°36 Linux embrqué : Le projet mGlinux	N°74 VFS : Système de fichiers virtuel	
N°37 L'impression sous Linux	N°75 Tuning de code	
N°38 Le desktop Shell : Enlightenment	N°76 Algorithmes évolutionnistes	
N°39 Sécurité : Patchez votre noyau !	N°77 Systèmes de fichiers chiffrés	
N°40 MySQL : la base de données OpenSource	N°78 Bluetooth, Spécifications, protocoles et configuration	
N°41 Steganographie ou l'art de la dissimulation de données	N°79 Sécurisation du Noyau avec PAX	
N°44 Comprenez Netbios pour Maîtriser l'interopérabilité windows	N°80 Run in memory	
N°46 Debian : Utilisez Samba avec le support ACL		
N°48 Caudium, votre prochain serveur Web !		

I 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**



Abonnez-vous !

Économisez

Plus de

20%*

* Sur le prix de vente unitaire France Métropolitaine

11 Numéros de
GNU/Linux Magazine
pour
le prix de 9*

* Gain pour un abonnement France Métropolitaine, par rapport au prix unitaire France Métropolitaine



11 Numéros de GNU/Linux Magazine à

55 €
(Offre France Métro)

Soit votre GNU/Linux Magazine à

5,00 €
(Tarif au numéro dans le cadre d'un abonnement France Métro)

**Vous lisez d'autres magazines des Éditions Diamond ?
Des offres de couplage sont disponibles ci-contre.**

Les 3 bonnes raisons de vous abonner !

Ne manquez plus aucun numéro.

Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.

Économisez 16,50 €/an ! (soit plus de 2 magazines offerts !)

Offres d'abonnement

(Nos tarifs s'entendent TTC et en euros)

		F	D	T	E1	E2	EUC	A	RM
		France Métro	DOM	TOM	Europe 1	Europe 2	Etats-unis Canada	Afrique	Reste du Monde
1	Abonnement Linux Magazine	55 €	59 €	67 €	69 €	66 €	70 €	68 €	77 €
2	Linux Magazine + Hors-série	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
3	Linux Magazine + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
4	Linux Magazine + Linux Pratique	78 €	85€	96 €	99 €	95 €	101 €	98 €	111 €
5	Linux Magazine + Hors-série + Linux Pratique	110 €	119 €	134 €	138€	133 €	140 €	137 €	154 €
6	Linux Magazine + Hors-série + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
7	Linux Magazine + Hors-série + MISC + Linux Pratique	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
8	Linux Pratique Essentiel + Linux Pratique	57 €	62 €	69 €	71 €	69 €	73 €	71 €	79 €

• Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède

• Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande

• Zone Reste du Monde : Autre Amérique, Asie, Océanie

• Zone Afrique : Europe de l'Est, Proche et Moyen-Orient

**Toutes les offres d'abonnement : en exemple les tarifs ci-dessous correspondant à la zone France Métro (F)
(Vous pouvez également vous abonner sur : www.ed-diamond.com)**

Linux Magazine (11 n°) offre 1 par ABO : 55€ Economie : 16,50 € en kiosque : 71,50€	Linux Magazine (11 n°) + Linux Magazine hors-série (6 n°) offre 2 en kiosque : 110,50€ par ABO : 83€ Economie : 27,50 €	Linux Magazine (11 n°) + Misc (6 n°) offre 3 en kiosque : 119,50€ par ABO : 84€ Economie : 35,50 €	Linux Magazine (11 n°) + Linux Pratique (6 n°) offre 4 en kiosque : 107,20€ par ABO : 78€ Economie : 29,20 €	Linux Pratique Essentiel (6 n°) + Linux Pratique (6 n°)
Linux Magazine (11 n°) offre 5 par ABO : 110€ Economie : 36,20 € en kiosque : 146,20€	Linux Magazine (11 n°) + Linux Magazine hors-série (6 n°) offre 6 par ABO : 116€ Economie : 42,50 € en kiosque : 158,50€	Linux Magazine (11 n°) + Misc (6 n°) + Linux Magazine (11 n°) + Linux Pratique (6 n°) offre 7 en kiosque : 194,20€ par ABO : 143€ Economie : 51,20 €	Linux Pratique (6 n°) + Linux Pratique (6 n°) offre 8 par ABO : 57€ Economie : 17,70 €	

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous :

Je fais mon choix de la 1ère offre :

Je sélectionne le N° (1 à 8) de l'offre choisie :

Je fais mon choix de la 2ème offre :

Je sélectionne le N° (1 à 8) de l'offre choisie :

Je sélectionne ma zone géographique (F à RM) :

Je sélectionne ma zone géographique (F à RM) :

J'indique la somme due : (Total 1) **€**

J'indique la somme due : (Total 2) **€**

Exemple : je souhaite m'abonner à l'offre Linux Magazine + Hors-série + MISC (offre 6) et je vis en Belgique (E1), ma référence est donc 6E1 et le montant de l'abonnement est de 144 euros.

Montant Total à régler (Total 1 + Total 2) **€**

Je choisis de régler par :

Chèque bancaire ou postal à l'ordre de Diamond Editions

Carte bancaire n°

Expire le :

Cryptogramme visuel :

Date et signature obligatoires



Diamond Editions
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex

MISC HORS-SÉRIE N°2

SPÉCIAL CARTES À PUCE

Disponible dès le 14/11/2008*
chez votre marchand de journaux

2 NOV./DEC. 2008 France Métro : 3 € / DOM : 8,80 € / TOM Surface : 9,90 XPF / TOM Avion : 13,00 XPF / CH : 15,80 CHF / IRL : 18,80 LUX : PORT/CONT : 9 Eur / CAN : 15 \$CAD

MISC
Multi-System & Internet Security Cookbook
HORS - SÉRIE

DOSSIER

CARTES À PUCE
DÉCOUVREZ LEURS FONCTIONNALITÉS ET LEURS LIMITES

HISTORIQUE
Retour sur la YesCard, un aperçu du système bancaire français

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

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

L 16844 - 2 H - F : 8,00 € . RD



*Comprenez
et utilisez
ces technologies
pour assurer
votre sécurité !*

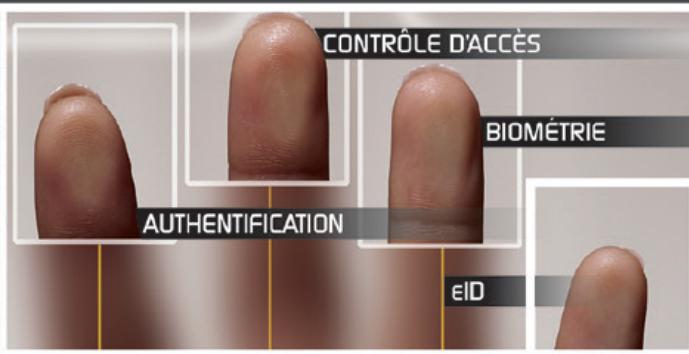
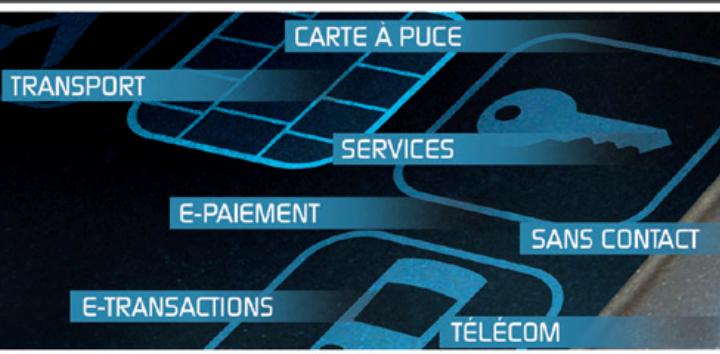
Visitez **www.ed-diamond.com** pour en savoir plus !

*sous réserve de toutes modifications



4-6 novembre 2008

Parc des Expositions de Paris-Nord Villepinte-FRANCE



Soyez prêt à aller plus loin !

- 520 exposants
- 20 000 visiteurs
- 35 000 m² d'exposition
- 250 conférenciers
- 1 700 congressistes



Votre badge **GRATUIT***
sur www.cartes.com
grâce à ce code : **LINUX**



ET PRÉPAREZ VOTRE VISITE :
Liste et actualités des exposants •
Animations • Inscription au Congrès
• Infos pratiques

Salons & Congrès

* Tarif pré-enregistrement : 50 € TTC - Sur place : 70 € TTC

L'événement leader mondial - Digital Security & Smart Technologies

