

Gestion de la Sécurité pour l'Extraction Parallèle Distribuée des Connaissances

Serge Chaumette*, Achraf Karray*,** et Damien Sauveron***

*LaBRI, UMR CNRS 5800 – Université Bordeaux 1
351 cours de la Libération, 33405 Talence CEDEX, FRANCE.

`serge.chaumette@labri.fr`

**ENIS – Université de Sfax

BP W.3038 Sfax, TUNISIE

`achraf.karray@labri.fr`

***XLIM, UMR CNRS 6172 – Université de Limoges

83 rue d'Isle, 87000 Limoges, FRANCE.

`damien.sauveron@unilim.fr`

Résumé. Dans le cadre de l'extraction de connaissances où les données et le code de fouille appartiennent à des propriétaires différents et qui ne se font pas nécessairement confiance, il semble difficile de concilier la confidentialité et l'intégrité des données d'une part et celles du code et de son exécution d'autre part. C'est une solution à ce problème non trivial que nous apportons grâce à l'utilisation d'entrepôts de données distribués sur la grille de cartes à puce que nous avons mise en place. Dans ce document nous décrivons l'exploitation de nos travaux dans le cadre d'une application particulière.

1 Introduction

Les grilles de calcul ou les architectures de type grappes ou clusters de stations sont aujourd'hui largement utilisées pour la fouille de données. Néanmoins, aucun système n'offre actuellement des garanties fortes de sécurité. Les objectifs que nous poursuivons sont : d'assurer la confidentialité et l'intégrité des données traitées ; de garantir la confidentialité et l'intégrité des algorithmes ou critères de fouille utilisés et de l'exécution de ces algorithmes. Pour assurer ces contraintes fortes de sécurité nous utilisons la grille de Java Cards^{TM1} (présentée Fig. 1) que nous avons mise en place dans le cadre de nos travaux sur la sécurité des systèmes distribués. Nous avons déjà déployé sur cette grille des applications sécurisées dans des domaines variés tels que la sécurité du calcul distribué, la sécurité d'une infrastructure PKI et maintenant la sécurité pour la fouille de données.

Nous aborderons dans la première section la problématique de la sécurité du code et des données dans le cadre de la fouille de données. Dans la seconde section, nous présenterons

¹Java and all Java-based marks are trademarks or registered trademarks of Sun microsystems, Inc. in the United States and other countries. The authors are independent of Sun microsystems, Inc. All other marks are the property of their respective owners.



FIG. 1 – *La grille de Java Cards.*

la grille de Java Cards (Chaumette et al., 2003; Chaumette et Sauveron, 2003; Chaumette et al., 2005; Atallah et al., 2005; Karray, 2004), son architecture matérielle, logicielle, et les outils d'administration actuellement disponibles. Nous exposerons dans la troisième section l'utilisation que nous faisons de cette grille dans le cadre d'une application spécifique que nous avons développée. Enfin nous présenterons brièvement les travaux en cours et nous conclurons.

2 Confidentialité et intégrité pour la fouille de données : impossible défi ?

Le contexte dans lequel nous nous plaçons est celui où le propriétaire des données et le propriétaire du code qui opère la fouille sont distincts. C'est évidemment un cas fréquent dans le domaine de l'extraction de connaissances. Bien souvent dans ce cadre, on souhaiterait pouvoir s'assurer que le jeu de données et le code de fouille ainsi que son exécution respectent certaines propriétés de sécurité telles que la confidentialité et l'intégrité. Toutefois, de telles contraintes posent beaucoup de problèmes puisqu'aucune des parties ne veut ni dévoiler ni mettre en péril ses biens (les données ou le code et son exécution). Ainsi aujourd'hui, faute de solution, soit les applications ne sont pas mises œuvre, soit les exigences de sécurité sont diminuées. Dans la section 4 nous illustrerons ce propos sur une application et nous montrerons comment à l'aide de notre grille de Java Cards nous pouvons assurer les propriétés de confidentialité et d'intégrité du jeu de données, du code et de son exécution. Toutefois, nous allons tout d'abord examiner comment sont classiquement assurées les propriétés que nous avons énoncées.

2.1 Confidentialité et intégrité du jeu de données

Une solution classique lorsqu'on a un jeu de données que l'on veut partager et dont on veut assurer certaines propriétés de sécurité consiste à définir une partie privée inaccessible (*i.e.* qui restera confidentielle et intègre) et une partie publique qui sera libre accès pour une simple consultation (*i.e.* pour assurer au minimum l'intégrité) ou en modification (*i.e.* pour assurer au minimum la confidentialité) ou en consultation et en modification (*i.e.* si on

ne souhaite assurer ni la confidentialité ni l'intégrité). On songera bien évidemment à assurer l'authenticité (ce n'est qu'une forme particulière de l'intégrité qui permet de s'assurer qu'on manipule les données originales) dans les cas de figure où cela est nécessaire, par exemple quand la responsabilité des fournisseurs de données risquerait d'être engagée. Bien souvent, l'implémentation de ces mécanismes se fait au travers d'une API publique qui offre un accès contrôlé aux diverses informations de la partie publique. Si le code de fouille de données est installé sur la machine partageant et hébergeant les données on utilise généralement un mécanisme de *sandbox* afin qu'il ne tente pas d'outrepasser ses droits par un accès direct au dépôt de données. On peut aussi envisager d'auditer soit même le code ou de le faire auditer par un tiers de confiance pour s'assurer de son respect de l'API et de son innocuité pour le reste des données confidentielles. Si le code de fouille accède à l'API via le réseau (*e.g.* pour l'interrogation d'une base de données) il est seulement nécessaire de garantir l'intégrité des messages échangés et l'authenticité des données reçues par le programme de fouille.

2.2 Confidentialité et intégrité du code et de son exécution

Les critères de fouille doivent rester secrets afin d'éviter par exemple que le fournisseur de données ne modifie ces dernières pour tromper le programme de recherche. On remarque d'ailleurs que la politique de confidentialité du code de fouille est en contradiction avec la solution proposée auparavant et consistant à permettre l'analyse du code pour s'assurer de son innocuité. De plus, pour assurer la viabilité de la fouille, il faut s'assurer que le code ne peut être modifié (*i.e.* garantir son intégrité) sans quoi les réponses ne seraient pas fiables. De la même façon il faut être capable de garantir la confidentialité de l'exécution et son intégrité.

En effet dans le cas où le code de fouille de données est installé sur une machine n'appartenant pas au propriétaire de ce code, il est facile de faire la rétro-conception du programme pour en analyser le contenu et donc de découvrir les critères de fouille. Il est aussi très facile de modifier ce code binaire pour perturber la recherche. De façon très simple, on peut aussi tracer l'exécution et donc connaître le code et les critères employés (*e.g.* l'administrateur peut faire une copie de la mémoire). Enfin il est facile de modifier l'exécution du programme, par exemple en utilisant des perturbations lumineuses (Govindavajhala et Appel, 2003).

2.3 Vers une solution : le tiers de confiance

Actuellement la seule véritable solution consiste à faire un compromis via un intermédiaire commun auquel les deux parties font confiance et qui assure la protection des données, du code et de son exécution.

Nous utilisons pour notre part une solution similaire mais notre intermédiaire est la carte à puce ou une émanation plus performante et aussi sécurisée. En effet les cartes à puce et plus particulièrement les Java Cards² (Sun microsystems, 2003; Chen, 2000) peuvent réellement permettre de résoudre ce défi grâce :

- au fait qu'elles sont évaluées par un CESTI (Centre d'Évaluation de la Sécurité des Technologies de l'Information) et certifiées en France par la DCSSI (Direction Centrale

²Java Card est une technologie permettant de faire fonctionner des programmes écrits en Java sur des cartes à puce. Cette technologie définit une plate-forme multi-applicative, portable et sécurisée pour les cartes à puce.

de la Sécurité des Systèmes d'Information) ce qui assure qu'on peut avoir une grande confiance dans les mécanismes sécuritaires qu'elles mettent en œuvre ;

- aux mécanismes sécurisés d'isolation des applications et de partages de données ;
- aux protections physiques offertes (*e.g.* blindage électromagnétique pour bloquer les émanations, pompe de charge pour lisser la consommation en courant, détecteur de conditions anormales de fonctionnement) qui protègent le code chargé et son exécution.

3 La plate-forme Java Card Grid

L'objectif du projet Java Card Grid consiste à définir et à mettre en œuvre une plate-forme matérielle et logicielle ainsi que les outils d'administration associés pour mener des expérimentations sur des applications ayant de fortes contraintes de sécurité.

3.1 La plate-forme matérielle

La plate-forme matérielle que nous avons déployée est représentée Fig. 2.

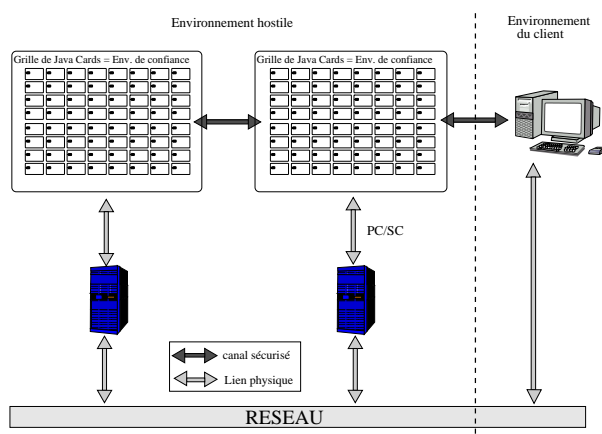


FIG. 2 – La plate-forme matérielle de la grille de Java Cards.

En pratique, nous avons déployé deux grilles interconnectées ensemble par un réseau filaire classique. Le matériel est assemblé dans une tour rackable d'une capacité de 19 unités. Chaque grille est composée de :

- un PC occupant 2U ;
- deux racks de SmartMount occupant 2U et comportant chacun 8 lecteurs de cartes de SCM Microsystems respectant le standard CCID, soit un total de 16 lecteurs CCID ;
- trois hubs USB de 7-ports (placés dans un rack vide de 2U) pour alimenter les lecteurs et les connecter au PC ;
- des Java Cards de différents fabricants insérées dans les lecteurs.

Nous avons aussi équipé un des PC d'un écran LCD et d'un clavier rackable spécial pour serveur (avec un touchpad intégré) que nous utilisons pour contrôler les machines. Une photo de la plate-forme est présentée Fig. 1.

3.2 L'infrastructure logicielle

L'infrastructure logicielle que nous avons conçue et implémentée comprend quant à elle deux couches : une couche de bas niveau qui gère les PC, les lecteurs et les cartes ; une couche de haut niveau qui gère l'infrastructure de calcul distribué que nous proposons.

À bas niveau, les PCs de contrôle tournent sous Linux et nous utilisons `pcsc-lite` (Corcoran et al.), une implémentation *open source* du standard PC/SC³, pour gérer les lecteurs. Pour les lecteurs CCID⁴ nous utilisons l'implémentation d'un pilote générique *open source* pour `pcsc-lite` (Rousseau). Par ailleurs comme nous voulions utiliser pour notre infrastructure une couche de haut niveau écrite en Java, nous avons choisi d'employer JPC/SC (IBM BlueZ Secure Systems), un wrapper JNI pour PC/SC, pour contrôler les lecteurs et gérer les applications embarquées dans les cartes (*cf.* Fig. 3).

Au niveau utilisateur, l'API de programmation se base sur Mandala (Chaumette et Vignéras, 2003). Mandala est un *framework* général qui a été développé dans notre équipe pour supporter le calcul distribué en Java. Il fournit une abstraction à la RMI mais avec un mécanisme supplémentaire d'invocation asynchrone de méthode (Vignéras et Grange) qui se révèle très utile dans un environnement où les ressources de calcul sont lentes.

L'ensemble de l'infrastructure logicielle déployée sur les deux machines est présentée Fig. 3.

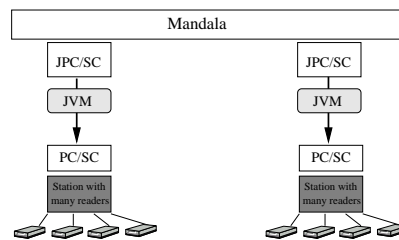


FIG. 3 – La plate-forme logicielle de la grille de Java Cards

3.3 Outils d'administration

Nous avons débuté la conception et l'implémentation de différents outils pour administrer la grille de Java Cards, *i.e.* pour surveiller et contrôler la topologie de la grille, pour détecter les cartes défectueuses, pour déployer de nouvelles applications, etc. Pour l'instant, seul l'outil de surveillance à distance de la topologie est disponible.

³PC/SC (PC/SC Workgroup) est un standard qui fournit une API de haut niveau pour communiquer avec les lecteurs de cartes à puce.

⁴CCID est un standard qui définit un protocole permettant à tout lecteur compatible CCID de dialoguer avec un hôte sans qu'il soit nécessaire d'installer un pilote spécifique.

4 Une application réelle : CBP/Air France

L'objectif de l'application que nous allons présenter est d'illustrer comment nous pouvons assurer la confidentialité et l'intégrité des données et du code (mais aussi de son exécution) quand ceux-ci appartiennent à deux entités distinctes ne se faisant pas confiance. Plus précisément l'application porte ici sur la recherche d'information dans les fichiers passagers d'une compagnie aérienne, par exemple Air France, par une agence gouvernementale, par exemple le bureau des douanes et de la protection des frontières des États-Unis, le CBP. Cette application correspond à des préoccupations réelles que nous décrivons ci-après.

4.1 Présentation du problème

Depuis le 11 septembre 2001, pour des raisons de sécurité, le CBP demande les informations (nom, numéro de carte de crédit, numéro de téléphone, nombre de bagages, etc.) contenues dans les *passenger name records* (PNR) (Transportation Security Administration), les fichiers de réservation des compagnies aériennes pour les vols à destination des États-Unis, afin de pouvoir effectuer certaines recherches. Selon Washington, ces données peuvent servir à identifier d'éventuels terroristes avant leur entrée sur le territoire américain. Le problème est que si une compagnie aérienne souhaitait préserver la confidentialité des données de ses passagers et donc ne pas communiquer son fichier au CBP, celui-ci pourrait annuler l'autorisation de la compagnie d'atterrir sur le territoire américain, mettant un terme à son activité commerciale vers les États-Unis.

Par ailleurs, dans le cas des compagnies aériennes opérant en Europe, les lois en vigueur dans la Communauté Européenne ont été récemment renforcées par des directives visant à contraindre ces compagnies à préserver la confidentialité des informations personnelles des passagers qu'elles transportent.

Ainsi, on se retrouve dans une situation assez étrange où les compagnies doivent à la fois préserver l'accès aux fichiers de passagers, comme la loi le leur impose, mais également les partager avec le CBP si elles veulent continuer leurs activités vers les États Unis. Pour légaliser cette situation ubuesque qui dans les faits a débuté en décembre 2003, le Conseil des ministres européen a entériné un accord de transfert avec les États-Unis en mai 2004 et ce contre l'avis du Parlement européen qui estimait que ce compromis représentait « une violation des droits fondamentaux » des citoyen européens (Parlement Européen, 2004). D'ailleurs les eurodéputés ont saisi pour arbitrage la Cour européenne de justice. En novembre 2005, l'avocat général indépendant nommé par la Cour européenne de justice a remis ses conclusions demandant l'annulation de la décision. Il s'agit maintenant d'attendre que les juges de la Cour se prononcent car ils ne sont pas tenus de suivre l'avis de l'avocat général (Cour de Justice des Communautés Européennes, 2005). De toutes façons quand le verdict définitif sera rendu l'accord sera quasi obsolète, puisque valable pour 3 ans (2004-2007). Afin de proposer une solution technique viable à cet imbroglio juridique nous avons mis en place une application tirant partie de la plate-forme de confiance que constitue la grille de Java Cards.

4.2 L'échec des solutions classiques

Une solution pourrait être que le CBP fournisse à Air France son application de fouille de données. Air France pourrait ainsi lancer cette application sur les fichiers de passagers

et ne retourner au CBP que la liste des passagers potentiellement suspects. Mais plusieurs problèmes se poseraient car comme nous l'avons déjà souligné le CBP ne souhaite pas dévoiler son algorithme, *e.g.* les critères discriminants mis en œuvre dans son analyse des données des passagers. Or, une analyse de rétro-conception du programme permettrait de les découvrir. Par ailleurs, le CBP ne souhaite pas non plus voir l'exécution de son code espionnée ou modifiée. Donc le code devrait être exécuté soit sur des machines du CBP soit sur une plate-forme de confiance. Enfin, si la compagnie aérienne acceptait de faire tourner le programme du CBP chez elle, elle ne pourrait pas être sûre que ce dernier n'a pas placé de code malicieux dans le programme afin d'avoir accès à toutes les données. Aussi, si la compagnie acceptait de faire tourner le code, cela pourrait seulement être sur une plate-forme de confiance dans laquelle le code ne pourrait pas faire plus que ce que le propriétaire de cette plate-forme lui a permis.

Il se dégage donc à l'évidence un fort besoin de disposer d'une plate-forme de confiance sur laquelle sera déployée le fichier de passagers et l'exécution du code de fouille de données du CBP.

4.3 Notre application

L'objectif de notre application est de permettre au CBP de rechercher des passagers suspects dans le fichier d'Air France, tout en garantissant les deux points suivants : le CBP ne doit pas pouvoir directement identifier les passagers ; Air France ne doit pas avoir connaissance des critères utilisés par le CBP. Pour ce faire, le fichier passagers d'Air France est déployé sur les processeurs (*i.e.* les cartes) accompagné d'une interface (API) permettant d'accéder à tout ou partie des informations contenues sans en violer la confidentialité. En particulier, un passager est identifié par une clef indépendante de son identité effective. Le code du CBP est lui aussi déployé sur les cartes ; il réalise une interrogation et centralise les clefs des passagers considérés comme suspects sur une carte particulière. Air France peut ensuite analyser ces clefs, effectuer une enquête plus poussée sur tel ou tel passager, et le cas échéant, donner les identités des personnes concernées au CBP.

4.4 Implémentation

Dans le cadre de notre application nous avons développé une interface graphique permettant à la compagnie Air France de saisir la liste de ses passagers et de la déployer sur les cartes. Dans cette application, un passager est représenté sur les cartes par la classe `Passenger` contenant toutes les informations le concernant, notamment son nom, son prénom, sa nationalité, etc. Sur chaque carte, nous disposons également d'une *applet* appelée `PassengerManager` qui permet à Air France de gérer la liste de ses passagers.

Cette *applet* `PassengerManager` fournit une interface réduite permettant au CBP de récupérer les informations spécifiées publiques par Air France. Pour cela, la classe `PassengerManager` implante une interface `PassengerInterface` (qui hérite de l'interface `Shareable`) qui donne seulement un accès partiel et contrôlé aux fichiers de passagers et ne permet pas d'identifier un passager particulier (*cf.* Fig. 4). Les fonctions de cette interface de type `Shareable` sont les seules fonctions de l'API d'Air France utilisables par les autres *applets*, donc par le CBP. Ainsi, Air France peut partager les données concernant ses passagers avec le CBP au sein d'un mécanisme sécurisé et contrôlé par le JCRE (Java Card Runtime Environment).

Gestion de la Sécurité pour l'Extraction Parallèle Distribuée des Connaissances

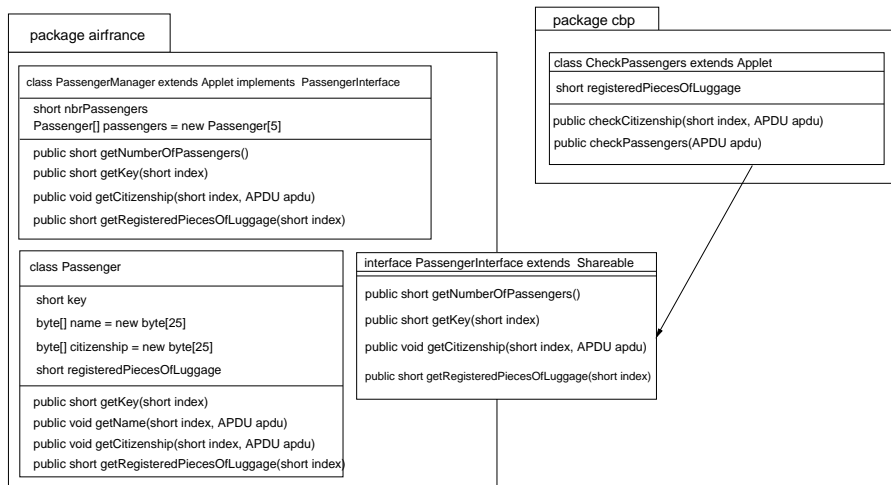


FIG. 4 – Le partage des données.

Une fois tous les passagers saisis par Air France, le CBP va tenter de déterminer quels sont ceux qui paraissent suspects. Pour cela, le CBP spécifie les critères nécessaires pour décider si tel passager est suspect ou non. Nous avons donc réalisé un outil graphique permettant une fois les critères saisis de les transmettre aux *applets* `CheckPassengers`, représentant le CBP sur les différentes cartes. Les *applets* `CheckPassengers` analysent alors la liste des passagers disponibles afin de trouver les passagers suspects correspondants aux critères de recherche. Chaque fois qu'un suspect est trouvé, la carte récupère son identifiant, `key` et l'envoie à une *applet* spéciale, appelée `CBPServer` qui centralise les résultats. Enfin, quand toutes les cartes ont fini la recherche, le CBP récupère auprès de l'*applet* `CBPServer` la liste des clefs des suspects trouvés et revient vers Air France pour demander les noms et prénoms de ces passagers, qu'Air France est libre de communiquer ou pas.

Pour être totalement exhaustif, on notera que dans notre application le CBP doit faire un minimum confiance à Air France, en ce sens qu'Air France aurait pu instrumenter son API pour tenter de découvrir les critères de recherche employés. De son côté s'il n'a pas totalement confiance le CBP peut aussi faire des requêtes inutiles pour tenter de leurrer Air France.

5 Travaux en cours

Nous travaillons actuellement sur le développement de nouvelles applications, le déploiement automatique des codes, l'augmentation significative de la taille de la plate-forme pour accroître la capacité de calcul et la gestion de *swap sécurisé off card* pour accroître la taille des données manipulables. Au delà du passage à un grand nombre de cartes, nous prévoyons l'utilisation des processeurs puissants issus des recherches actuelles et ayant une architecture de sécurité comparable (Trusted Computing Group), afin de les intégrer dès qu'ils arriveront sur le marché.

6 Conclusion

Dans cet article nous avons présenté nos travaux portant sur la sécurité et la confidentialité dans le cadre de l'extraction des connaissances. Nous avons montré les problématiques soulevées en ce qui concerne les données et les méthodes d'extraction.

L'application permettant le partage d'information entre les compagnies aériennes de l'Union Européenne et le CBP américain, que nous avons développée et décrite dans cet article, met en évidence l'apport à cette problématique d'une architecture distribuée à base de Java Cards.

Grâce à cette expérience, nous pensons pouvoir rapidement passer à l'échelle en terme d'une part de la quantité de données traitées et d'autre part de la complexité des traitements effectués, dès que des processeurs puissants adaptés seront présents sur le marché. Il n'en reste pas moins que l'architecture à base de cartes telle que nous l'avons mise en œuvre aujourd'hui offre une solution à des problèmes effectifs, comme nous l'avons vu dans cet article.

Remerciement

Notre projet est soutenu par Axalto, Gemplus et IBM BlueZ Secure Systems (pour les cartes), par SCM Microsystems et SmartMount (pour les lecteurs), et par Sun microsystems (pour la totalité de la plate-forme).

Nous remercions également : Fujitsu, Giesecke&Devrient, Oberthur Card Systems et Sharp pour les échantillons de Java Card ; David Corcoran et Ludovic Rousseau pour leur travaux sur `pcsc-lite` et le pilote générique CCID.

Références

- Atallah, E., S. Chaumette, F. Darrigade, A. Karray, et D. Sauveron (2005). A Grid of Java Cards to Deal with Security Demanding Application Domains. In *Proceedings of e-Smart 2005*, Nice, France.
- Chaumette, S., P. Grange, , A. Karray, D. Sauveron, et P. Vignéras (2005). Secure distributed computing on a Java Card grid. In *Proceedings of 7th International Workshop on Java for Parallel and Distributed Computing*, Denver, CO, USA.
- Chaumette, S., P. Grange, D. Sauveron, et P. Vignéras (2003). Computing with Java Cards. In *Proceedings of CCCT'03 and 9th ISAS'03*, Orlando, FL, USA.
- Chaumette, S. et D. Sauveron (2003). The Smart Cards Grid Project. <http://www.labri.fr/Person/~chaumett/recherche/cartesapuce/smartcardsgrid/documents/poster.pdf>. Poster presented at Cartes 2003.
- Chaumette, S. et P. Vignéras (2003). A framework for seamlessly making object oriented applications distributed. In *Parallel Computing 2003*, Dresden, Germany.
- Chen, Z. (2000). *Java Card™ Technology for Smart Cards : Architecture and Programmer's Guide*. The Java™ Series. Addison-Wesley.
- Corcoran, D., L. Rousseau, et D. Sauveron. `pcsc-lite` home page. <http://alioth.debian.org/projects/pcsc-lite/>.

- Cour de Justice des Communautés Européennes (2005). Conclusions de l'avocat général dans les affaires c-317/04 et c-318/04. <http://www.curia.eu.int/fr/actu/communiqués/cp05/aff/cp050098fr.pdf>.
- Govindavajhala, S. et A. Appel (2003). Using Memory Errors to Attack a Virtual Machine. In *Proceedings of IEEE Symposium on Security and Privacy*.
- IBM BlueZ Secure Systems. Java Wrappers for PC/SC. <http://www.musclecard.com/middleware/files/jpcsc-0.8.0-src.zip>.
- Karray, A. (2004). Calcul sécurisé sur grille de cartes à puce. Master's thesis, ENIS – University of Sfax.
- Parlement Européen (2004). Proposition de résolution de transfert des pnr. www.europarl.eu.int/meetdocs/committees/libe/20040309/524914fr.pdf.
- PC/SC Workgroup. PC/SC Workgroup Home. <http://www.pcscworkgroup.com/>.
- Rousseau, L. CCID free software driver. <http://pcsc-lite.alioth.debian.org/ccid.html>.
- Sun microsystems (2003). *Java Card™ 2.2.1 Specifications*. Sun microsystems.
- Transportation Security Administration. <http://www.tsa.gov/public/display?theme=5&content=09000519800cf3a7>, http://www.tsa.gov/public/interweb/assetlibrary/Secure_Flight_PRA_Notice_9.21.04.pdf.
- Trusted Computing Group. Trusted Computing Group Home. <https://www.trustedcomputinggroup.org/home>.
- Vignéras, P. et P. Grange. The Mandala website. <http://mandala.sourceforge.net/>.

Summary

Within the framework of data mining where the data and the code belong to different owners who do not necessarily trust each other, it seems difficult to reconcile the confidentiality and the integrity of the data on the one hand and those of the code and its execution on the other hand. It is a solution with this noncommonplace problem which we bring thanks to the use of data warehouses distributed on the grid of smart cards that we have developed. In this document we describe the exploitation of our work within the framework of a particular application.